ModelArts

最佳实践

文档版本	01
发布日期	2025-02-25

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 ModelArts 最佳实践案例列表

在最佳实践文档中，提供了针对多种场景、多种AI引擎的ModelArts案例，方便您通过如下案例快速了解使用ModelArts完成AI开发的流程和操作。

DeepSeek 模型推理场景

表 1-1

样例	场景	说明
DeepSeek模型基于ModelArts Lite Server适配MindIE推理部署指导	BF16推理、Int8量化推理	介绍DeepSeek R1和DeepSeek V3模型基于ModelArts Lite Server的推理部署过程，推理使用MindeIE框架和昇腾NPU计算资源。

LLM 大语言模型训练推理场景

样例	场景	说明
• 主流开源大模型基于DevServer适配ModelLink PyTorch NPU训练指导 • 主流开源大模型基于DevServer适配LlamaFactory PyTorch NPU训练指导	预训练、SFT全参微调训练、LoRA微调训练	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Lite Server的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。训练后的模型可用于推理部署，搭建大模型问答助手。

样例	场景	说明
• 主流开源大模型基于Standard+OBS适配PyTorch NPU训练指导 • 主流开源大模型基于Standard+OBS+SFS适配PyTorch NPU训练指导	预训练、SFT全参微调训练、LoRA微调训练	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Standard的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，搭建大模型问答助手。
主流开源大模型基于DevServer适配PyTorch NPU推理指导	推理部署、推理性能测试、推理精度测试、推理模型量化	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Lite Server的推理部署过程，推理使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可用于搭建大模型问答助手。
主流开源大模型基于Standard适配PyTorch NPU推理指导	推理部署、推理性能测试、推理精度测试、推理模型量化	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Standard的推理部署过程，推理使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可用于搭建大模型问答助手。

多模态模型场景

样例	场景	说明
Qwen-VL基于Standard+OBS+SFS适配PyTorch NPU训练指导（6.3.912） Qwen-VL模型基于Standard+OBS适配PyTorch NPU训练指导（6.3.912） Qwen-VL基于Lite Server适配PyTorch NPU的Finetune训练指导(6.3.912) Qwen-VL基于Lite Server适配PyTorch NPU的推理指导（6.3.909） MiniCPM-V2.6基于Lite Server适配PyTorch NPU训练指导（6.3.912） MiniCPM-V2.0推理及LoRA微调基于Lite Server适配PyTorch NPU指导（6.3.910） InternVL2基于LIte Server适配PyTorch NPU训练指导（6.3.912） LLaVA-NeXT基于Lite Server适配PyTorch NPU训练微调指导（6.3.912） LLaVA模型基于Lite Server适配PyTorch NPU预训练指导（6.3.912） LLaVA模型基于Lite Server适配PyTorch NPU推理指导（6.3.906） Llama 3.2-Vision基于Lite Server适配Pytorch NPU训练微调指导（6.3.912） LLaMA-VID基于Lite Server适配PyTorch NPU推理指导（6.3.910） moondream2基于Lite Server适配PyTorch NPU推理指导	Qwen-VL、MiniCPM-V2系列、InternVL2、LLaVA-NeXT、LLaVA、Llama 3.2-Vision、LLaMA-VID、moondream2等模型的训练或推理	介绍常见多模态模型使用PyTorch框架和昇腾NPU计算资源，基于ModelArts Lite Server或者Standard的训练或推理过程。

文生图模型训练推理场景

样例	场景	说明
• SDXL基于Standard适配PyTorch NPU的LoRA训练指导 • SD1.5&SDXL Diffusers框架基于DevServer适配PyTorch NPU训练指导 • SD1.5&SDXL Kohya框架基于DevServer适配PyTorch NPU训练指导	SDXL、SD1.5模型训练	介绍AIGC模型SDXL、SD1.5基于ModelArts Lite Server的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。训练后的模型可用于推理部署，应用于文生图场景。
SD3 Diffusers框架基于DevServer适配PyTorch NPU推理指导	SDXL、SD1.5模型推理	介绍AIGC模型SDXL、SD1.5基于ModelArts Lite Server的推理过程，推理使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可应用于文生图场景。
Open-Clip基于Lite Server适配PyTorch NPU训练指导	Open-Clip模型训练	介绍Open-Clip模型基于ModelArts Lite Server的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 应用于AIGC和多模态视频编码器。

文生视频场景

样例	场景	说明
• CogVideoX训练推理基于DevServer适配PyTorch NPU指导 • Open-Sora1.2基于DevServer适配PyTorch NPU训练推理指导 • Open-Sora-Plan1.0基于DevServer适配PyTorch NPU训练推理指导 • Open-Sora 1.0基于Lite Server适配PyTorch NPU训练指导（6.3.905）	CogVideo模型、Open-Sora模型训练推理	介绍CogVideo、Open-Sora-Plan、Open-Sora1.2模型基于ModelArts DevServer的训练推理过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，应用于文生视频场景。

数字人场景

样例	场景	说明
• Wav2Lip推理基于DevServer适配PyTorch NPU推理指导 • Wav2Lip训练基于DevServer适配PyTorch NPU训练指导	Wav2Lip，人脸说话视频模型，训练、推理	Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。 案例主要介绍如何基于ModelArts Lite Server上的昇腾NPU资源进行模型训练推理。

内容审核场景

样例	场景	说明
Bert基于Lite Server适配MindSpore Lite推理指导(6.3.910) Yolov8基于Lite Server适配MindSpore Lite推理指导（6.3.909） Paraformer基于Lite Server适配PyTorch NPU推理指导（6.3.911）	Bert、Yolov8、Paraformer等内容审核模型推理	案例主要介绍内容审核场景的相关模型如何基于ModelArts Lite Server上的昇腾NPU资源进行模型推理。

ModelArts Standard 权限配置

样例	对应功能	场景	说明
ModelArts Standard权限管理	IAM权限配置、权限管理	为子账号配置权限	当一个华为云账号下需创建多个IAM子账号时，可参考此样例，为IAM子账号赋予使用ModelArts所需的权限。避免IAM子账号因权限问题导致使用时出现异常。

ModelArts Standard 自动学习案例

表 1-2 自动学习样例列表

样例	对应功能	场景	说明
口罩检测	自动学习	物体检测	基于AI Gallery口罩数据集，使用ModelArts自动学习的物体检测算法，识别图片中的人物是否佩戴口罩。
垃圾分类	自动学习	图像分类	该案例基于华为云AI开发者社区AI Gallery中的数据集资产，让零AI基础的开发者完成“图像分类”的AI模型的训练和部署。

ModelArts Standard 开发环境案例

表 1-3 Notebook 样例列表

样例	镜像	对应功能	场景	说明
将Notebook的Conda环境迁移到SFS磁盘	-	环境迁移	开发环境	本案例介绍如何将Notebook的Conda环境迁移到SFS磁盘上。
使用ModelArts VSCode插件调试训练ResNet50图像分类模型	MindSpore	VS Code Toolkit工具	目标检测	本案例以Ascend Model Zoo为例，介绍如何通过VS Code插件及ModelArts Notebook进行云端数据调试及模型开发。

ModelArts Standard 模型训练案例

表 1-4 自定义算法样例列表

样例	镜像	对应功能	场景	说明
使用ModelArts Standard自定义算法实现手写数字识别	PyTorch	自定义算法	手写数字识别	使用用户自己的算法，训练得到手写数字识别模型，并部署后进行预测。

样例	镜像	对应功能	场景	说明
从0制作自定义镜像并用于训练（PyTorch+CPU/GPU）	PyTorch	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。
从0制作自定义镜像并用于训练（MPI+CPU/GPU）	MPI	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。
从0制作自定义镜像并用于训练（Tensorflow+GPU）	Tensorflow	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是Tensorflow，训练使用的资源是GPU。
从0制作自定义镜像并用于训练（MindSpore+Ascend）	MindSpore	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是NPU。

ModelArts Standard 推理部署

表 1-5 推理部署列表

样例	对应功能	场景	说明
基于ModelArts Standard一键完成商超商品识别模型部署	在线服务	物体检测	此案例以“商超商品识别”模型为例，完成从AI Gallery订阅模型，到ModelArts一键部署为在线服务的免费体验过程。
第三方推理框架迁移到ModelArts Standard推理自定义引擎	第三方框架推理部署	-	ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

第三方案例列表

第三方案例来源为**华为云开发者社区“云驻计划”**。由于ModelArts产品的持续更新和迭代，第三方案例中的界面和步骤可能因时效性而与最新产品有所差异，仅供学习和参考。

表 1-6 第三方案例列表

分类	文章名称	作者
Standard自动学习	**2步打通ModelArts和Astro实现AI应用落地**	**胡琦**
Standard开发环境	**想不想让一张静态的照片动起来**	**林欣**
	基于TensorFlow训练轻量化ssdlite_mbv2人脸手机检测模型	**AI练习生**
	基于ModelArts的手写数字识别	**AXYZdong**
	AI 文字编辑图片 instruct-pix2pix 案例	**AXYZdong**
Standard推理部署	**上线二维码检测识别服务**	**林欣**
	使用ModelArts对8类常见生活垃圾进行分类	**福州司马懿**
	使用ModelArts搭建"花卉种类识别"服务	**福州司马懿**

2 昇腾能力应用地图

ModelArts支持如下开源模型昇腾NPU进行训练和推理。

DeepSeek 系列模型

表 2-1 DeepSeek 系列模型

支持模型	应用场景	软件技术栈	指导文档
DeepSeek R1	推理	MindIE	DeepSeek模型基于ModelArts Lite Server适配MindIE推理部署指导
DeepSeek V3	推理	MindIE	

LLM 大语言模型

ModelArts针对以下主流的LLM大模型进行了基于昇腾NPU的适配工作，可以直接使用适配过的模型在NPU上进行推理训练。

表 2-2 LLM 模型推理能力

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
Llama	Llama-7b Llama-13b Llama-65b	推理	Ascend-vLLM	• 主流开源大模型基于Lite Server适配Ascend-vLLM PyTorch NPU推理指导（6.3.912） • 主流开源大模型基于Standard适配PyTorch NPU推理指导（6.3.911） • 主流开源大模型基于Lite Cluster适配PyTorch NPU推理指导（6.3.911）
Llama2	Llama2-7b Llama2-13b Llama2-70b	推理	Ascend-vLLM	
Llama3	Llama3-8b Llama3-70b	推理	Ascend-vLLM	
Yi	yi-6b yi-9b yi-34b	推理	Ascend-vLLM	
deepseek	deepseek-llm-7b deepseek-llm-67b deepseek-coder-instruct-33b	推理	Ascend-vLLM	
Qwen	qwen-7b qwen-14b qwen-72b	推理	Ascend-vLLM	
Qwen1.5	qwen1.5-0.5b qwen1.5-7b qwen1.5-1.8b qwen1.5-14b qwen1.5-32b qwen1.5-72b qwen1.5-110b	推理	Ascend-vLLM	
Qwen2	qwen2-0.5b qwen2-1.5b qwen2-7b qwen2-72b qwen2-57b-a14b	推理	Ascend-vLLM	

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
Qwen2.5	qwen2.5-0.5b qwen2.5-1.5b qwen2.5-3b qwen2.5-7b qwen2.5-14b qwen2.5-32b qwen2.5-72b	推理	Ascend-vLLM	
Baichuan2	baichuan2-7b baichuan2-13b	推理	Ascend-vLLM	
gemma	gemma-2b gemma-7b	推理	Ascend-vLLM	
ChatGLM	chatglm2-6b chatglm3-6b	推理	Ascend-vLLM	
GLMv4	glm4-9b	推理	Ascend-vLLM	
mistral	mistral-7b mistral-8x7b	推理	Ascend-vLLM	
falcon	falcon-11b	推理	Ascend-vLLM	
llama3.1	llama3.1-8b llama3.1-70b llama-3.1-405B	推理	Ascend-vLLM	
llama3.2	llama-3.2-1B llama-3.2-3B	推理	Ascend-vLLM	
deepseek	deepseek-v2-236b deepseek-v2-lite-16b	推理	Ascend-vLLM	
qwen-vl	qwen-vl qwen-vl-chat	推理	Ascend-vLLM	
MiniCPM-v2	MiniCPM-v2	推理	Ascend-vLLM	
gte-Qwen2-7B-instruct	gte-Qwen2-7B-instruct	推理	Ascend-vLLM	

表 2-3 LLM 模型训练能力

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
Llama2	Llama2-7b Llama2-13b Llama2-70b	预训练、SFT全参微调、LoRA微调	ModelLink LLaMAFactory	• 主流开源大模型基于DevServer适配ModelLink PyTorch NPU训练指导 • 主流开源大模型基于DevServer适配LlamaFactory PyTorch NPU训练指导 • 主流开源大模型基于Standard+OBS适配PyTorch NPU训练指导 • 主流开源大模型基于Standard+OBS+SFS适配PyTorch NPU训练指导
Llama3	Llama3-8b Llama3-70b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
Qwen	qwen-7b qwen-14b qwen-72b	预训练、SFT全参微调、LoRA微调	ModelLink	
Qwen1.5	qwen1.5-7b qwen1.5-14b qwen1.5-32b qwen1.5-72b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
Qwen2	qwen2-0.5b qwen2-1.5b qwen2-7b qwen2-72b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
Qwen2.5	qwen2.5-0.5b qwen2.5-7b qwen2.5-14b qwen2.5-32b qwen2.5-72b	预训练、SFT全参微调、LoRA微调	ModelLink	
Yi	yi-6b yi-34b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
ChatGLMv3	glm3-6b	预训练、SFT全参微调、LoRA微调	ModelLink	
GLMv4	glm4-9b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
Baichuan2	baichuan2-13b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
mistral	mistral-7b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
mixtral	mixtral-8x7b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
llama3.1	llama3.1-8b llama3.1-70b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
llama3.2	llama3.2-1b llama3.2-3b	预训练、SFT全参微调、LoRA微调	ModelLink LlamaFactory	
Qwen2_VL	qwen2_vl-2b qwen2_vl-7b qwen2_vl-72b	预训练、SFT全参微调、LoRA微调	LlamaFactory	
Falcon2	falcon-11B	预训练、SFT全参微调、LoRA微调	LlamaFactory	

多模态模型

ModelArts针对以下主流的多模态模型进行了基于昇腾NPU的适配工作，可以直接使用适配过的模型在NPU上进行推理或训练。

表 2-4 多模态模型基于 Ascend-vLLM 框架的推理

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
internVL2	internVL2-8B internVL2-26B internVL2-40B internVL2-Llama3-76B	推理	Ascend-vLLM	• 主流开源大模型基于Lite Server适配Ascend-vLLM PyTorch NPU推理指导（6.3.912） • 主流开源大模型基于Standard适配PyTorch NPU推理指导（6.3.911） • 主流开源大模型基于Lite Cluster适配PyTorch NPU推理指导（6.3.911）
MiniCPM	MiniCPM-v2.6	推理	Ascend-vLLM	
qwen2-vl	qwen2-vl-2B qwen2-vl-7B qwen2-vl-72B	推理	Ascend-vLLM	
llava	llava-1.5-7b llava-1.5-13b llava-v1.6-7b llava-v1.6-13b llava-v1.6-34b	推理	Ascend-vLLM	
llava-onevision-qwen2	llava-onevision-qwen2-0.5b-ov-hf llava-onevision-qwen2-7b-ov-hf	推理	Ascend-vLLM	

表 2-5 多模态模型基于 PyTorch 的训练推理

模型名称	应用场景	软件技术栈	指导文档
Qwen-VL	训练 推理	PyTorch	Qwen-VL基于Standard+OBS+SFS适配PyTorch NPU训练指导（6.3.912） Qwen-VL模型基于Standard+OBS适配PyTorch NPU训练指导（6.3.912） Qwen-VL基于Lite Server适配PyTorch NPU的Finetune训练指导(6.3.912) Qwen-VL基于Lite Server适配PyTorch NPU的推理指导（6.3.909）
MiniCPM-V2.6	训练	PyTorch	MiniCPM-V2.6基于Lite Server适配PyTorch NPU训练指导（6.3.912）
MiniCPM-V2.0	训练 推理	PyTorch	MiniCPM-V2.0推理及LoRA微调基于Lite Server适配PyTorch NPU指导（6.3.910）

模型名称	应用场景	软件技术栈	指导文档
InternVL2	训练	PyTorch	InternVL2基于LIte Server适配PyTorch NPU训练指导（6.3.912）
LLaVA-NeXT	训练	PyTorch	LLaVA-NeXT基于Lite Server适配PyTorch NPU训练微调指导（6.3.912）
LLaVA	训练 推理	PyTorch	LLaVA模型基于Lite Server适配PyTorch NPU预训练指导（6.3.912） LLaVA模型基于Lite Server适配PyTorch NPU推理指导（6.3.906）
LLama 3.2-Vision	训练	PyTorch	Llama 3.2-Vision基于Lite Server适配Pytorch NPU训练微调指导（6.3.912）
LLaMA-VID	推理	PyTorch	LLaMA-VID基于Lite Server适配PyTorch NPU推理指导（6.3.910）
moondream2	推理	PyTorch	moondream2基于Lite Server适配PyTorch NPU推理指导

文生图模型

ModelArts针对以下主流的AIGC文生图模型进行了基于昇腾NPU的适配工作，可以直接使用适配过的模型在NPU上进行推理或训练。

表 2-6 文生图模型

模型名称	应用场景	软件技术栈	指导文档
Stable Diffusion XL（SDXL）	Diffusers训练 Kohya训练 ComfyUI推理 Diffusers推理	PyTorch	SD1.5&SDXL Diffusers框架基于DevServer适配PyTorch NPU训练指导（6.3.908） SD1.5&SDXL Kohya框架基于DevServer适配PyTorch NPU训练指导（6.3.908） SDXL基于Standard适配PyTorch NPU的LoRA训练指导（6.3.908） SDXL&SD1.5 ComfyUI基于Lite Cluster适配NPU推理指导（6.3.906） SDXL基于Standard适配PyTorch NPU的Finetune训练指导（6.3.905） SDXL基于Lite Server适配PyTorch NPU的Finetune训练指导（6.3.905） SDXL基于Lite Server适配PyTorch NPU的LoRA训练指导（6.3.905）

模型名称	应用场景	软件技术栈	指导文档
Stable Diffusion 1.5（SD1.5）	ComfyUI推理 Diffusers推理	PyTorch	SD1.5&SDXL Diffusers框架基于DevServer适配PyTorch NPU训练指导（6.3.908） SD1.5&SDXL Kohya框架基于DevServer适配PyTorch NPU训练指导（6.3.908） SDXL&SD1.5 ComfyUI基于Lite Cluster适配NPU推理指导（6.3.906） SD1.5基于Lite Server适配PyTorch NPU Finetune训练指导（6.3.904）
Stable Diffusion 3（SD3）	训练 Diffusers推理	PyTorch	SD3基于Lite Server适配PyTorch NPU的训练指导（6.3.912） SD3 Diffusers框架基于Lite Server适配PyTorch NPU推理指导（6.3.912）
Stable Diffusion 3.5（SD3.5）	推理	PyTorch	SD3.5基于Lite Server适配PyTorch NPU的推理指导（6.3.912）
FlUX.1	训练 推理	PyTorch	FLUX.1基于DevSever适配PyTorch NPU Finetune&Lora训练指导（6.3.911） FlUX.1基于Lite Server适配PyTorch NPU推理指导（6.3.912）
Hunyuan-DiT	推理	PyTorch	Hunyuan-DiT基于Lite Server部署适配PyTorch NPU推理指导（6.3.909）
Open-clip	训练 推理	PyTorch	Open-Clip基于Lite Server适配PyTorch NPU训练指导

表 2-7 数字人模型

模型名称	应用场景	软件技术栈	指导文档
Wav2Lip	训练	PyTorch	Wav2Lip训练基于Lite Server适配PyTorch NPU训练指导（6.3.907）
	推理	PyTorch	Wav2Lip推理基于Lite Server适配PyTorch NPU推理指导（6.3.907）

表 2-8 文生视频模型

模型名称	应用场景	软件技术栈	指导文档
CogVideoX 1.5	训练	PyTorch	CogVideoX1.5 5b模型基于Lite Server适配PyTorch NPU全量训练指导（6.3.912）
CogVideoX	训练	PyTorch	CogVideoX模型基于Lite Server适配PyTorch NPU全量训练指导（6.3.911）
Open-Sora1.2	训练 推理	PyTorch	Open-Sora1.2基于Lite Server适配PyTorch NPU训练推理指导（6.3.910）
Open-Sora1.0	训练	PyTorch	Open-Sora 1.0基于Lite Server适配PyTorch NPU训练指导（6.3.905）
Open-Sora-Plan1.0	训练 推理	PyTorch	Open-Sora-Plan1.0基于Lite Server适配PyTorch NPU训练推理指导（6.3.907）

表 2-9 内容审核模型

模型名称	应用场景	软件技术栈	指导文档
Bert	推理	MindSpore Lite	Bert基于Lite Server适配MindSpore Lite推理指导(6.3.910)
Yolov8	推理	MindSpore Lite	Yolov8基于Lite Server适配MindSpore Lite推理指导（6.3.909）
Paraformer	推理	PyTorch	Paraformer基于Lite Server适配PyTorch NPU推理指导（6.3.911）

3 DeepSeek 系列模型推理

3.1 DeepSeek 模型基于 ModelArts Lite Server 适配 MindIE 推理部署指导

3.1.1 方案概述

场景描述

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展DeepSeek R1和DeepSeek V3模型推理部署的详细过程。推理框架使用MindIE。

资源规划

本方案推荐用户使用W8A8量化权重，需要2台Ascend Snt9B资源。Snt9B资源的单卡显存不低于64GB。

资源购买

使用Lite Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

由于镜像是在西南-贵阳一区域，资源购买建议在西南-贵阳一区域上完成。

3.1.2 准备权重

3.1.2.1 准备 BF16 权重

本章节介绍BF16权重转换操作。

BF16获取权重有2种方式：

方式一：直接获取HuggingFace社区已经转换完成的BF16权重。

方式二：基于DeepSeek官网提供的FP8权重转换为BF16权重。

方式一提供的权重是开发者在社区贡献的权重，如果是用于生产环境的业务，建议选择方式二，通过DeepSeek官方发布的FP8权重进行转换。

约束限制

由于模型权重较大，请确保您的磁盘有足够的空间放下所有权重，例如DeepSeek-V3在转换前权重约为640G左右，在转换后权重约为1.3T左右。

创建文件存放目录

在每台Server机器上创建一个目录${path-to-file}，例如：/home/data/，用于存放权重文件和rank_table_file.json文件。

方式一：直接获取 HuggingFace 社区已经转换完成的 BF16 权重

通过下述地址直接下载HuggingFace社区中开发者贡献的已经转换成功的BF16权重。建议在Server机器上创建${path-to-file}/deepseekV3-bf16或${path-to-file}/deepseekR1-bf16目录，并直接将权重文件下载到该目录中。

opensourcerelease/DeepSeek-V3-bf16

opensourcerelease/DeepSeek-R1-bf16

下载完成后，需要修改权重文件中config.json文件，把model_type字段值改为“deepseekv2”。

方式二：将 FP8 权重转换为 BF16 权重

介绍如何将DeepSeek官方发布的FP8权重转换为BF16的权重。用于生产环境的业务推荐使用此方式。具体操作步骤如下。

1. 下载FP8的权重，下载地址如下。建议在每台Server机器上创建${path-to-file}/deepseekV3或${path-to-file}/deepseekR1目录，并直接将权重文件下载到该目录中。

   deepseek-ai/DeepSeek-V3

   deepseek-ai/DeepSeek-R1

2. 准备FP8至BF16权重转换脚本fp8_cast_bf16.py，具体脚本内容参见**权重转换脚本文件fp8_cast_bf16.py**。权重转换需要使用有CPU资源的机器，建议直接登录Lite Server节点执行权重转换。
3. 在Server机器上创建权重转换后的存放目录${path-to-file}/deepseekV3-bf16或${path-to-file}/deepseekR1-bf16目录。
4. 执行转换命令python fp8_cast_bf16.py。

   FP8权重路径是${path-to-file}/deepseekV3，例如：/home/data/deepseekV3，将转换后的BF16权重输出到${path-to-file}/deepseekV3-bf16，例如：/home/data/deepseekV3-bf16，可以使用以下命令，此处以deepseekV3为例。

   ```
   python fp8_cast_bf16.py --input-fp8-hf-path ${path-to-file}/deepseekV3 --output-bf16-hf-path ${path-to-file}/deepseekV3-bf16
   ```

5. 修改转换后权重文件中config.json文件，把model_type字段值改为“deepseekv2”。
6. 转换后的权重文件再复制到其它三台机器的相同目录。

⚠ 注意

如果是新开的Server机器则需要安装torch、tqdm等软件包，具体命令如下

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple tqdm
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple torch
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple safetensors
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple numpy
```

权重转换脚本文件 fp8_cast_bf16.py

权重转换脚本fp8_cast_bf16.py文件内容如下。

```
import os
import json
from argparse import ArgumentParser
from glob import glob
from tqdm import tqdm

import torch
from safetensors.torch import load_file, save_file

#from kernel import weight_dequant

def weight_dequant(weight: torch.Tensor, scale: torch.Tensor, block_size: int = 128) -> torch.Tensor:

    #Get the original dimensions of weight
    M, N = weight.shape

    # Compute the effective block dimensions for scale
    scale_m, scale_n = scale.shape
    assert scale_m == (M + block_size - 1) // block_size, "Mismatch in scale rows and weight rows."
    assert scale_n == (N + block_size - 1) // block_size, "Mismatch in scale columns and weight columns."

    # Convert weight to float32 for calculations
    weight = weight.to(torch.float32)

    # Expand scale to match the weight tensor's shape
    scale_expanded = scale.repeat_interleave(block_size, dim=0).repeat_interleave(block_size, dim=1)

    # Trim scale_expanded to match weight's shape if necessary
    scale_expanded = scale_expanded[:M, :N]

    # Perform element-wise multiplication
    dequantized_weight = weight * scale_expanded

    # Convert the output to the default dtype
    dequantized_weight = dequantized_weight.to(torch.get_default_dtype())

    return dequantized_weight


def main(fp8_path, bf16_path):
    torch.set_default_dtype(torch.bfloat16)
    os.makedirs(bf16_path, exist_ok=True)
    model_index_file = os.path.join(fp8_path, "model.safetensors.index.json")
    with open(model_index_file, "r") as f:
        model_index = json.load(f)
    weight_map = model_index["weight_map"]

    # Cache for loaded safetensor files
    loaded_files = {}
    fp8_weight_names = []

    # Helper function to get tensor from the correct file
    def get_tensor(tensor_name):
        file_name = weight_map[tensor_name]
```

```
        if file_name not in loaded_files:
            file_path = os.path.join(fp8_path, file_name)
            loaded_files[file_name] = load_file(file_path, device="cpu")
        return loaded_files[file_name][tensor_name]

    safetensor_files = list(glob(os.path.join(fp8_path, "*.safetensors")))
    safetensor_files.sort()
    for safetensor_file in tqdm(safetensor_files):
        file_name = os.path.basename(safetensor_file)
        current_state_dict = load_file(safetensor_file, device="cpu")
        loaded_files[file_name] = current_state_dict

        new_state_dict = {}
        for weight_name, weight in current_state_dict.items():
            if weight_name.endswith("_scale_inv"):
                continue
            elif weight.element_size() == 1: #FP8 weight
                scale_inv_name = f"{weight_name}_scale_inv"
                try:
                    # Get scale_inv from the correct file
                    scale_inv = get_tensor(scale_inv_name)
                    fp8_weight_names.append(weight_name)
                    new_state_dict[weight_name] = weight_dequant(weight, scale_inv)
                except KeyError:
                    print(f"Warning: Missing scale_inv tensor for {weight_name}, skipping conversion")
                    new_state_dict[weight_name] = weight
            else:
                new_state_dict[weight_name] = weight

        new_safetensor_file = os.path.join(bf16_path, file_name)
        save_file(new_state_dict, new_safetensor_file)

        # Memory management: keep only the 2 most recently used files
        if len(loaded_files) > 2:
            oldest_file = next(iter(loaded_files))
            del loaded_files[oldest_file]
            # torch.cuda.empty_cache()

    # Update model index
    new_model_index_file = os.path.join(bf16_path, "model.safetensors.index.json")
    for weight_name in fp8_weight_names:
        scale_inv_name = f"{weight_name}_scale_inv"
        if scale_inv_name in weight_map:
            weight_map.pop(scale_inv_name)
    with open(new_model_index_file, "w") as f:
        json.dump({"metadata": {}, "weight_map": weight_map}, f, indent=2)


if __name__ == "__main__":
    parser = ArgumentParser()
    parser.add_argument("--input-fp8-hf-path", type=str, required=True)
    parser.add_argument("--output-bf16-hf-path", type=str, required=True)
    args = parser.parse_args()
    main(args.input_fp8_hf_path, args.output_bf16_hf_path)
```

3.1.2.2 准备 W8A8 权重

前提条件

已完成**准备BF16权重**。

W8A8 量化权重生成

介绍如何将BF16权重量化为W8A8的权重，具体操作步骤如下。

1. 在Server机器上创建权重量化后的存放目录${path-to-file}/deepseekV3-w8a8或${path-to-file}/deepseekR1-w8a8目录。
2. 下载msit源码，请下载指定分支br_noncom_MindStudio_8.0.0_POC_20251231。
   ```
   git clone -b br_noncom_MindStudio_8.0.0_POC_20251231 https://gitee.com/ascend/msit.git
   ```
3. 进入到msit/msmodelslim的目录；并在进入的msmodelslim目录下，运行安装脚本install.sh。
   ```
   cd msit/msmodelslim
   bash install.sh
   ```

   ⚠ 注意

   执行install过程会下载依赖包，因此需要确保能够访问到pip源。

4. 进入到msit/msmodelslim/example/DeepSeek目录，执行转换命令。

   BF16权重路径是${path-to-file}/deepseekV3-bf16，例如：/home/data/deepseekV3-bf16，将量化后的W8A8权重输出到${path-to-file}/deepseekV3-w8a8，例如：/home/data/deepseekV3-w8a8，可以使用以下命令，此处以deepseekV3为例。

   ```
   cd example/DeepSeek
   python3 quant_deepseek_w8a8.py --model_path ${path-to-file}/deepseekV3-bf16 --save_path ${path-to-file}/deepseekV3-w8a8
   ```
5. 量化后的权重文件再复制到另外一台机器的相同目录。

DeepSeek 量化 FAQ

- Q：报错 This modeling file requires the following packages that were not found in your environment： flash_attn. Run 'pip install flash_attn'
- A：当前环境中缺少flash_attn库且昇腾不支持该库，运行时需要注释掉权重文件夹中modeling_deepseek.py中的部分代码。

```
from transformers.modeling_utils import PreTrainedModel
from transformers.pytorch_utils import (
    ALL_LAYERNORM_LAYERS,
    is_torch_greater_or_equal_than_1_13,
)
from transformers.utils import (
    add_start_docstrings,
    add_start_docstrings_to_model_forward,
    is_flash_attn_2_available,
    is_flash_attn_greater_or_equal_2_10,
    logging,
    replace_return_docstrings,
)
from transformers.utils.import_utils import is_torch_fx_available
from .configuration_deepseek import DeepseekV3Config
import torch.distributed as dist
import numpy as np

if is_flash_attn_2_available():
    from flash_attn import flash_attn_func, flash_attn_varlen_func
    from flash_attn.bert_padding import index_first_axis, pad_input, unpad_input  # noqa
```

  - Q：modeling_utils.py报错 if metadata.get("format") not in ["pt", "tf", "flax", "mix"]: AttributeError: "NoneType" object has no attribute 'get';
  - A：说明输入的权重中缺少metadata字段，需安装更新transformers版本（>=4.48.2）

3.1.3 部署推理服务

3.1.3.1 自动化脚本快速部署推理服务（推荐）

场景描述

本方案提供了一键式安装脚本start.sh，用于快速部署推理服务。脚本中实现了以下步骤的自动化操作：

1. 环境检查
2. 拉取镜像
3. 根据实际值更新rank_table_file.json
4. 启动容器
5. 进入容器启动服务

前提条件

1. 已经完成**资源购买**。
2. 请确保多机之间能够正常通信。
3. 请确保每台机器都能访问到西南-贵阳一SWR镜像仓库。

约束限制

脚本中的镜像是在西南-贵阳一区域，建议在西南-贵阳一区域上部署推理服务。

脚本默认只支持在标准的欧拉或HCE 2.0操作系统上执行。

步骤一：检查环境

1. SSH登录机器后，检查NPU设备检查。如果驱动版本不是24.1.0，请先升级驱动和对应固件。

   ```
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   24.1.0版本驱动文件为**Ascend-hdk-910b-npu-driver_24.1.0_linux-aarch64.run**，对应固件文件为**Ascend-hdk-910b-npu-firmware_7.5.0.3.220.run**，请申请下载。

   安装固件命令如下，安装完后需要reboot重启机器。

   ```
   chmod 700 *.run
   ./Ascend-hdk-910b-npu-firmware_7.5.0.3.220.run --full
   reboot
   ```

   安装24.1.0驱动命令如下：

   ```
   ./Ascend-hdk-910b-npu-driver_24.1.0_linux-aarch64.run --full --install-for-all
   ```

   安装完成后再使用如下命令查看是否安装正确。

   ```
   npu-smi info -t board -i 1 | egrep -i "software|firmware"
   ```

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：自动化部署

1. 需要先安装yum和expect。

   ```
   sudo apt-get install yum
   yum install expect
   ```

2. 把rank_table_file.json、start.sh和权重文件放在同一目录下。其中rank_table_file.json和start.sh只需要在主节点即可。rank_table_file.json仅作为模板使用，会在start.sh脚本里自动更新，文件内容详见**附录：rank_table_file.json文件**和**一键部署脚本start.sh**。参考目录结构如下：

   ```
   ${mountPath}
   |---rank_table_file.json
   |---start.sh
   |---DeepSeekR1-w8a8  # 权重文件所在目录，每个节点都要有，且目录结构保持一致
   ```

3. 执行部署脚本start.sh。

   ```
   sh start.sh --model_name ${modelName} --mount_path ${mountPath} --maxSeqLen ${maxSeqLen} --node_ips ${nodeIps} --passwords ${nodePwds}
   ```

   - modelName：模型权重所在文件夹名称。注意：文件夹名称需要包含R1或V3。例如：DeepSeekR1-w8a8。
   - mountPath：容器挂载的路径，且不能为/home，该路径下包含权重文件所在目录。即为权重文件所在目录的父目录。
   - maxSeqLen：输入长度+输出长度的最大值。推荐默认16384。
   - nodeIps：节点IP列表，使用“,“分隔。填2个节点IP地址。
   - nodePwds：各节点的root用户登录密码，使用“,“分隔，和上述节点要一一对应。

步骤三：调用

执行请求调用模型

```
curl -ik -H 'Content-Type: application/json' -d '{"messages":[{"role":"user","content":"请讲一个笑话"}],"model":"${model_name}","temperature":0.6,"max_tokens":1024}' -X POST http://${ip}:${port}/v1/chat/completions
```

- model_name：为要调用的模型名称，即DeepSeek-V3或DeepSeek-R1
- ip：为**步骤二：自动化部署**中nodeIps中第一个IP
- port：为要访问的端口，默认1025

一键部署脚本 start.sh

```
#!/bin/bash

# 定义默认值
# 参数变量
model_name=""
mount_path=""
maxSeqLen=
image_name="swr.cn-southwest-2.myhuaweicloud.com/ei_ascendcloud_devops/mindie:2.0.T3.1-800I-A2-py311-openeuler24.03-lts-0220"
CONTAINER_NAME="mindie"
# 根据变量修改的值
MODEL_NAME=""
model_path=""
mount_path="${mount_path%/}"
```



```
RANK_TABLE_FILE="$mount_path/rank_table_file.json"

port=22
# 默认maxSeqLen为16K时的参数配置，如果maxSeqLen修改为32K时，需要修改maxInputTokenLen、maxPrefillTokens和maxIterTimes的值，具体修改方法参见附录：config.json文件介绍。
maxPrefillBatchSize=100
maxInputTokenLen=8192
maxPrefillTokens=8192
maxIterTimes=8192
httpsEnabled=false
multiNodesInferEnabled=true
interCommTLSEnabled=false
interNodeTLSEnabled=false


# 定义期望的软件和固件版本
expected_software_version="24.1.0"
expected_firmware_version="7.5.0.3.220"


# 定义节点IP列表（第一个IP为主节点）
node_ips=("")
NODE_IPS=""
echo "node_ips:   ${node_ips[*]}"

# 定义密码列表（如果只有一个密码，则所有节点使用该密码）
# node_passwords=("xxx" "xxx")  # 替换为实际密码
node_passwords=("")


# 检查expect是否安装
if ! command -v expect &> /dev/null; then
    echo "错误: expect未安装，请先执行yum install expect"
    exit 1
fi

# 解析命令行参数
ARGS=$(getopt -o n:p:l:i:k: -l model_name:,mount_path:,maxSeqLen:,node_ips:,passwords: -- "$@")
if [ $? -ne 0 ]; then
    echo "参数解析失败"
    exit 1
fi

eval set -- "$ARGS"

while true; do
    case "$1" in
        -n|--model_name)
            model_name=$2  # 忽略大小写
            MODEL_NAME=$model_name
            shift 2
            ;;
        -p|--mount_path)
            mount_path="$2"
            # 去除 mount_path 最后的 '/'
            mount_path="${mount_path%/}"
            shift 2
            ;;
        -l|--maxSeqLen)
            maxSeqLen="$2"
            shift 2
            ;;
        -i|--node_ips)
            IFS=',' read -r -a node_ips <<< "$2"
            shift 2
            ;;
        -k|--passwords)
            IFS=',' read -r -a node_passwords <<< "$2"
            shift 2
```

```
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "用法：$0 -n <model_name> -p <mount_path> -l <maxSeqLen> -i <node_ips> -k <passwords>"
            echo "  -n, --model-name: 模型名称，可选值为 r1 或 v3"
            echo "  -p, --mount-path: 挂载路径，必须提供，且不能为 /home"
            echo "  -l, --max-seq-len: 输入长度+输出长度的最大值，需要在8192上加2^的若干次方数"
            echo "  -i, --node-ips: 逗号分隔的节点IP地址"
            echo "  -k, --passwords: 密码字符串列表"
            exit 1
            ;;
    esac
done

# 检查 mount_path 是否为空或为 /home
if [ -z "$mount_path" ]; then
    echo "必须提供 mount_path 参数"
    exit 1
fi

if [ "$mount_path" = "/home" ]; then
    echo "mount_path 不能为 /home"
    exit 1
fi

if [ -z "$maxSeqLen" ]; then
    echo "必须提供 maxSeqLen 参数"
    exit 1
fi
model_path=$mount_path/$model_name
# 转换 model_name 为小写，以便忽略大小写
lower_model_name=$(echo "$model_name" | tr '[:upper:]' '[:lower:]')

# 检查 model_name 是否包含 r1 或 v3
if [[ "$lower_model_name" == *"r1"* ]]; then
    model_name="DeepSeek-R1"
    # model_path=$mount_path/deepseekR1-w8a8
    # echo "模型路径为：$model_path"

elif [[ "$lower_model_name" == *"v3"* ]]; then
    model_name="DeepSeek-V3"
    # model_path=$mount_path/deepseekV3-w8a8
    # echo "模型路径为：$model_path"
else
    echo "错误: model_name 参数中必须包含 r1 或 v3"
    exit 1
fi


RANK_TABLE_FILE="$mount_path/rank_table_file.json"
# 打印参数值
echo "maxSeqLen的值为：$maxSeqLen"
echo "模型名:$model_name"
echo "模型路径:$model_path"
echo "挂载路径:$mount_path"
echo "node_ips:  ${node_ips[*]}"
# 将数组元素以空格连接成字符串，再使用tr将空格替换为逗号
NODE_IPS=$(printf "%s " "${node_ips[@]}" | tr ' ' ',')
# 去除末尾多余的逗号
NODE_IPS=${NODE_IPS%,}
echo "NODE_IPS: ${NODE_IPS}"
echo "node_passwords:  ${node_passwords[*]}"
echo "rank_table_file文件路径:$RANK_TABLE_FILE"
# exit 0
```

```
echo "步骤1: 检查NPU的软件和固件信息..."
npu_smi_output=$(npu-smi info -t board -i 1 | egrep -i "software|firmware")

if [ $? -ne 0 ]; then
    echo "检查NPU软件和固件信息失败，退出脚本。"
    exit 1
fi


# 提取实际的软件和固件版本
actual_software_version=$(echo "$npu_smi_output" | grep -i "Software Version" | awk -F': ' '{print $2}' | tr -d
' ')
actual_firmware_version=$(echo "$npu_smi_output" | grep -i "Firmware Version" | awk -F': ' '{print $2}' | tr -
d ' ')

# 比较软件版本
if echo -e "$actual_software_version\n$expected_software_version" | sort -V | head -n1 | grep -q
"$actual_software_version"; then
    if [ "$actual_software_version" != "$expected_software_version" ]; then
        echo "软件版本不对，期望版本至少为 $expected_software_version，实际版本为
$actual_software_version"
        exit 1
    fi
fi

# 比较固件版本
if echo -e "$actual_firmware_version\n$expected_firmware_version" | sort -V | head -n1 | grep -q
"$actual_firmware_version"; then
    if [ "$actual_firmware_version" != "$expected_firmware_version" ]; then
        echo "固件版本不对，期望版本至少为 $expected_firmware_version，实际版本为
$actual_firmware_version"
        exit 1
    fi
fi

echo "NPU软件和固件版本检查通过。"

# 步骤2: 执行一系列检查命令，如果有报错则退出
echo "步骤2: 执行检查命令..."
commands=(
    "for i in {0..7}; do hccn_tool -i \$i -lldp -g | grep Ifname; done"
    "for i in {0..7}; do hccn_tool -i \$i -link -g ; done"
    "for i in {0..7}; do hccn_tool -i \$i -net_health -g ; done"
    "for i in {0..7}; do hccn_tool -i \$i -netdetect -g ; done"
    "for i in {0..7}; do hccn_tool -i \$i -gateway -g ; done"
    "for i in {0..7}; do hccn_tool -i \$i -tls -g ; done | grep switch"
    "for i in {0..7}; do hccn_tool -i \$i -tls -s enable 0; done"
)

echo "拉取镜像"
docker pull $image_name

# 步骤4: 获取enp67s0f5网卡下的IP地址
echo "步骤4: 获取本机IP地址..."
ip_addr_now=$(hostname -I | awk '{print $1}')
if [ -z "$ip_addr_now" ]; then
    echo "获取当前物理机IP地址失败。"
    exit 1
fi
echo "获取当前物理机IP地址成功，IP地址为: $ip_addr_now"


# 初始化标记位
is_first_node=false
echo "主节点ip地址为：${node_ips[0]}"
# exit 0
# 判断 ip_addr_now 是否是第一个节点
if [ "$ip_addr_now" = "${node_ips[0]}" ]; then
    is_first_node=true
```

```
fi

if $is_first_node; then
    echo "${ip_addr_now}作为主节点"

    # 步骤3: 获取每张卡的IP地址并打印（修正版）
    echo "步骤3: 获取每张卡的IP地址..."

    # 获取主节点的显卡IP
    echo "步骤3: 获取主节点的显卡IP..."
    declare -a main_npu_ips
    for i in {0..7}; do
        ip_info=$(hccn_tool -i $i -ip -g 2>&1)
        if [ $? -ne 0 ]; then
            echo "获取卡号 $i 的IP地址失败，退出脚本。"
            exit 1
        fi

        ip_addr=$(echo "$ip_info" | awk -F':' '/ipaddr/ {print $2}' | tr -d ' ')
        if [ -z "$ip_addr" ]; then
            echo "卡号 $i 未找到有效IP地址，退出脚本。"
            exit 1
        fi
        main_npu_ips[$i]=$ip_addr
        echo "主节点NPU卡$i IP地址: ${main_npu_ips[$i]}"
    done
    echo "主节点IP地址获取完成。"

    # 备份原始文件
    cp "$RANK_TABLE_FILE" "${RANK_TABLE_FILE}.bak"

    # 设置环境变量传递显卡IP
    export ips_0="${main_npu_ips[@]}"

    # 更新主节点的server_id和container_ip (使用Python)
    python3 <<EOF
import json
import os

def update_json(filename, server_id, container_ip):
    device_ips = os.getenv("ips_0").split()
    with open(filename, 'r') as f:
        data = json.load(f)
    # 更新主节点的server_id和container_ip
    data['server_list'][0]['server_id'] = server_id
    data['server_list'][0]['container_ip'] = container_ip
    # 更新主节点的device_ip
    for i in range(8):
        if i < len(device_ips):
            data['server_list'][0]['device'][i]['device_ip'] = device_ips[i]
        else:
            data['server_list'][0]['device'][i]['device_ip'] = ""
    # 删除多余的设备条目（如果有的话）
    for server in data['server_list']:
        if len(server['device']) > 8:
            server['device'] = server['device'][:8]
    with open(filename, 'w') as f:
        json.dump(data, f, indent=4)

update_json("$RANK_TABLE_FILE", "${node_ips[0]}", "${node_ips[0]}")
EOF

    # 远程登录其他节点，获取显卡IP并更新rank_table_file.json
    for (( node_index=1; node_index<${#node_ips[@]}; node_index++ )); do
        node_ip=${node_ips[$node_index]}
        node_password=${node_passwords[$node_index]}

        echo "步骤3: 获取节点 $node_ip 的显卡IP..."
```

```
    # 使用expect远程执行命令获取显卡IP
    declare -a remote_npu_ips
    for i in {0..7}; do
        ip_info=$(expect -c "
            set timeout 10
            spawn ssh -p $port -o StrictHostKeyChecking=no root@$node_ip \"hccn_tool -i $i -ip -g\"
            expect {
                \"password:\" {
                    send \"${node_password}\r\"
                    exp_continue
                }
                eof {
                    catch wait result
                    exit [lindex \$result 3]
                }
            }
        ")
        if [ $? -ne 0 ]; then
            echo "获取节点 $node_ip 卡号 $i 的IP地址失败，退出脚本。"
            exit 1
        fi

        ip_addr=$(echo "$ip_info" | awk -F':' '/ipaddr/ {print $2}' | tr -d ' ')
        if [ -z "$ip_addr" ]; then
            echo "节点 $node_ip 卡号 $i 未找到有效IP地址，退出脚本。"
            exit 1
        fi
        remote_npu_ips[$i]=$ip_addr
        echo "节点 $node_ip NPU卡$i IP地址: ${remote_npu_ips[$i]}"
    done
    echo "节点 $node_ip IP地址获取完成。"

    # 设置环境变量传递显卡IP
    export ips_1="${remote_npu_ips[@]}"

    # 更新当前节点的server_id和container_ip
    python3 <<EOF
import json
import os

def update_server_info(filename, server_index, server_id, container_ip):
    device_ips = os.getenv("ips_1").split()
    with open(filename, 'r') as f:
        data = json.load(f)

    # 确保 server_list 有足够的条目
    while server_index >= len(data['server_list']):
        new_server = {
            "device": [
                {"device_id": str(i), "device_ip": "", "rank_id": str(len(data['server_list']) * 8 + i)} for i in range(8)
            ],
            "server_id": "",
            "container_ip": ""
        }
        data['server_list'].append(new_server)

    # 更新指定服务器的信息
    data['server_list'][server_index]['server_id'] = server_id
    data['server_list'][server_index]['container_ip'] = container_ip

    # 更新设备 IP 和 rank_id
    for i in range(min(len(device_ips), 8)):
        data['server_list'][server_index]['device'][i]['device_ip'] = device_ips[i]
        data['server_list'][server_index]['device'][i]['rank_id'] = str(server_index * 8 + i)

    # 确保设备条目数量正确
    data['server_list'][server_index]['device'] = data['server_list'][server_index]['device'][:8]

    # 更新 server_count
```

```
    data['server_count'] = str(len(data['server_list']))

    with open(filename, 'w') as f:
        json.dump(data, f, indent=4)

update_server_info("$RANK_TABLE_FILE", $node_index, "$node_ip", "$node_ip")
EOF
    done


    # 传输给子节点
    # 传输 rank_table_file.json 到其他节点
    for (( node_index=1; node_index<${#node_ips[@]}; node_index++ )); do
        node_ip=${node_ips[$node_index]}
        node_password=${node_passwords[$node_index]}

        echo "传输 rank_table_file.json 到节点 $node_ip..."

        # 使用 expect 自动输入密码并执行 scp
        expect -c "
            set timeout 10
            spawn scp -P $port -ro StrictHostKeyChecking=no $mount_path/rank_table_file.json $mount_path/
start.sh root@$node_ip:$mount_path
            expect {
                \"password:\" {
                    send \"$node_password\r\"
                    exp_continue
                }
                eof {
                    catch wait result
                    exit [lindex \$result 3]
                }
            }
        "

        if [ $? -eq 0 ]; then
            echo "文件传输到节点 $node_ip 成功。"
        else
            echo "文件传输到节点 $node_ip 失败。"
            exit 1
        fi
    done

    echo "所有文件传输完成。"

    # 连接到远程节点并执行命令
    for (( node_index=1; node_index<${#node_ips[@]}; node_index++ )); do
        node_ip=${node_ips[$node_index]}
        node_password=${node_passwords[$node_index]}

        echo "连接到节点 $node_ip 并执行命令..."

        expect -c "
            set timeout 10
            spawn ssh -p $port -o StrictHostKeyChecking=no root@$node_ip \"bash $mount_path/start.sh -n
$MODEL_NAME -p $mount_path -l $maxSeqLen -i $NODE_IPS > $mount_path/log_$node_ip.txt 2>&1\"
            expect {
                \"password:\" {
                    send \"$node_password\r\"
                    exp_continue
                }
                eof {
                    catch wait result
                    exit [lindex \$result 3]
                }
            }
        "

        if [ $? -eq 0 ]; then
```

```
            echo "命令在节点 $node_ip 执行成功。"
        else
            echo "命令在节点 $node_ip 执行失败。"
            exit 1
        fi
    done

    echo "远程执行脚本成功"
sleep 32
fi

# 步骤5: 启动容器
echo "步骤5: 启动容器..."

# 启动容器
docker run -itd --privileged \
    -v /etc/localtime:/etc/localtime \
    -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
    -v /usr/local/Ascend/firmware:/usr/local/Ascend/firmware \
    --device=/dev/davinci_manager \
    --device=/dev/devmm_svm \
    --device=/dev/hisi_hdc \
    -v /var/log/npu/:/usr/slog \
    -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
    -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
    -v $mount_path:$mount_path \
    --net=host \
    --ipc=host \
    --shm-size 2g \
    --name ${CONTAINER_NAME} \
    ${image_name} \
    /bin/bash

# 检查容器是否启动成功
if [ $? -eq 0 ]; then
    echo "容器启动成功，容器名：${CONTAINER_NAME}"
else
    echo "容器启动失败！"
    exit 1
fi
# exit 0
# 步骤6: 将命令写入 ~/.bashrc 文件并执行
echo "步骤6: 将命令写入 ~/.bashrc 文件并执行..."

# 定义需要写入的命令列表（直接使用宿主机变量）
commands_to_write=(
    "cd /usr/local/Ascend/mindie/latest"
    "chmod 750 mindie-service"
    "chmod -R 550 mindie-service/bin"
    "chmod -R 500 mindie-service/bin/mindie_llm_backend_connector"
    "chmod 550 mindie-service/lib"
    "chmod 440 mindie-service/lib/*"
    "chmod 550 mindie-service/lib/grpc"
    "chmod 440 mindie-service/lib/grpc/*"
    "chmod -R 550 mindie-service/include"
    "chmod -R 550 mindie-service/scripts"
    "chmod 750 mindie-service/logs"
    "chmod 750 mindie-service/conf"
    "chmod 640 mindie-service/conf/config.json"
    "chmod 700 mindie-service/security"
    "chmod -R 700 mindie-service/security/*"
    "source /usr/local/Ascend/ascend-toolkit/set_env.sh"
    "source /usr/local/Ascend/nnal/atb/set_env.sh"
    "source /usr/local/Ascend/atb-models/set_env.sh"
    "unset HCCL_OP_EXPANSION_MODE"
    "export ATB_LLM_HCCL_ENABLE=1"
    "export ATB_LLM_COMM_BACKEND=\"hccl\""
    "export PYTORCH_NPU_ALLOC_CONF=\"expandable_segments:True\""
    "export MIES_CONTAINER_IP=${ip_addr_now}"
```

```
    "export RANKTABLEFILE=$mount_path/rank_table_file.json"
    "export HCCL_DETERMINISTIC=false"
    "export PARALLEL_PARAMS=\"[1,16,1,16,-1,-1]\""
    "export ATB_WORKSPACE_MEM_ALLOC_ALG_TYPE=3"
    "export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1"
    "export NPU_MEMORY_FRACTION=0.96"
    "export HCCL_CONNECT_TIMEOUT=7200"
    "export HCCL_EXEC_TIMEOUT=0"
    # "export MINDIE_LOG_TO_STDOUT=1"
)

# 将命令列表转换为单行字符串（避免引号嵌套问题）
commands_str=$(printf "%s\n" "${commands_to_write[@]}")

# 在容器内执行命令（使用单引号包裹）
docker exec ${CONTAINER_NAME} bash -c '
    # 逐行处理命令
    while IFS= read -r cmd; do
        # 检查命令是否已存在
        if ! grep -Fxq "$cmd" ~/.bashrc; then
            echo "$cmd" >> ~/.bashrc
        fi
    done <<< "$1"

    # 使配置生效
    # source ~/.bashrc
' _ "${commands_str}"  # 通过参数传递命令列表

# 检查命令是否执行成功
if [ $? -eq 0 ]; then
    echo "命令已成功写入 ~/.bashrc 并执行。"
else
    echo "命令写入或执行失败！"
    echo "删除容器${CONTAINER_NAME}"
    docker stop ${CONTAINER_NAME} && docker rm ${CONTAINER_NAME}
    echo "删除成功"
    exit 1
fi


# 容器内执行命令
echo "环境变量设置的当前节点ip为:"
docker exec ${CONTAINER_NAME} bash -c 'source ~/.bashrc && echo $MIES_CONTAINER_IP'
docker exec ${CONTAINER_NAME} bash -c "cat $RANK_TABLE_FILE"


docker exec ${CONTAINER_NAME} bash -c "
    cd /usr/local/Ascend/mindie/latest/mindie-service/conf && \
    sed -i 's|\"ipAddress\" : \".*\"|\"ipAddress\" : \"${node_ips[0]}\"|g' config.json && \
    sed -i 's|\"managementIpAddress\" : \".*\"|\"managementIpAddress\" : \"${node_ips[0]}\"|g' config.json
&& \
    sed -i 's|\"modelName\" : \".*\"|\"modelName\" : \"${model_name}\"|g' config.json && \
    sed -i 's|\"modelWeightPath\" : \".*\"|\"modelWeightPath\" : \"${model_path}\"|g' config.json && \
    sed -i 's|\"maxPrefillBatchSize\" : .*|\"maxPrefillBatchSize\" : ${maxPrefillBatchSize},|g' config.json && \
    sed -i 's|\"maxSeqLen\" : .*|\"maxSeqLen\" : ${maxSeqLen},|g' config.json && \
    sed -i 's|\"maxInputTokenLen\" : .*|\"maxInputTokenLen\" : ${maxInputTokenLen},|g' config.json && \
    sed -i 's|\"maxPrefillTokens\" : .*|\"maxPrefillTokens\" : ${maxPrefillTokens},|g' config.json && \
    sed -i 's|\"httpsEnabled\" : .*|\"httpsEnabled\" : ${httpsEnabled},|g' config.json && \
    sed -i 's|\"multiNodesInferEnabled\" : .*|\"multiNodesInferEnabled\" : ${multiNodesInferEnabled},|g'
config.json && \
    sed -i 's|\"interCommTLSEnabled\" : .*|\"interCommTLSEnabled\" : ${interCommTLSEnabled},|g'
config.json && \
    sed -i 's|\"interNodeTLSEnabled\" : .*|\"interNodeTLSEnabled\" : ${interNodeTLSEnabled},|g' config.json
&& \
    sed -i 's|\"maxIterTimes\" : .*|\"maxIterTimes\" : ${maxIterTimes},|g' config.json
"

# 检查修改后的结果
echo "修改后的值如下所示："
```

```
docker exec ${CONTAINER_NAME} bash -c "cd /usr/local/Ascend/mindie/latest/mindie-service/conf &&
    cat config.json"



docker exec ${CONTAINER_NAME} bash -c "source ~/.bashrc && rm -rf /root/mindie/cache/* && cd /usr/
local/Ascend/mindie/latest/mindie-service &&
    nohup ./bin/mindieservice_daemon &> $mount_path/start_log_$ip_addr_now.txt &"



# 查看容器日志
echo "查看容器日志："
docker logs -f ${CONTAINER_NAME}

echo "所有步骤执行完毕。"
```

3.1.3.2 手动部署推理服务

前提条件

已经完成**资源购买**。

约束限制

脚本中的镜像是在西南-贵阳一区域，请在西南-贵阳一区域上部署推理服务。

步骤一：检查环境

1. SSH登录机器后，检查NPU设备检查。如果驱动版本不是24.1.0，请先升级驱动和对应固件。

   ```
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   24.1.0版本驱动文件为**Ascend-hdk-910b-npu-driver_24.1.0_linux-aarch64.run**，对应固件文件为**Ascend-hdk-910b-npu-firmware_7.5.0.3.220.run**，请申请下载。

   安装固件命令如下，安装完后需要reboot重启机器。

   ```
   chmod 700 *.run
   ./Ascend-hdk-910b-npu-firmware_7.5.0.3.220.run --full
   reboot
   ```

   安装24.1.0驱动命令如下：

   ```
   ./Ascend-hdk-910b-npu-driver_24.1.0_linux-aarch64.run --full --install-for-all
   ```

   安装完成后再使用如下命令查看是否安装正确。

   ```
   npu-smi info -t board -i 1 | egrep -i "software|firmware"
   ```

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：获取推理镜像

镜像获取命令如下。

```
docker pull swr.cn-southwest-2.myhuaweicloud.com/ei_ascendcloud_devops/mindie:2.0.T3.1-800I-A2-py311-openeuler24.03-lts-0220
```

如果是权限导致的镜像拉取失败，请参考昇腾社区提供的**MindIE镜像**申请并下载2.0.T3.1-800I-A2-py311-openeuler24.03-lts版本的镜像。

步骤三：创建 rank_table_file.json

在启动容器前需要使用rank_table_file.json文件用于多机部署。

1. 检查机器网络情况

```
# 检查物理链接
for i in {0..7}; do hccn_tool -i $i -lldp -g | grep Ifname; done
# 检查链接情况
for i in {0..7}; do hccn_tool -i $i -link -g ; done
# 检查网络健康情况
for i in {0..7}; do hccn_tool -i $i -net_health -g ; done
# 查看侦测ip的配置是否正确
for i in {0..7}; do hccn_tool -i $i -netdetect -g ; done
# 查看网关是否配置正确
for i in {0..7}; do hccn_tool -i $i -gateway -g ; done
# 检查NPU底层tls校验行为一致性，建议全0
for i in {0..7}; do hccn_tool -i $i -tls -g ; done | grep switch
# NPU底层tls校验行为置0操作
for i in {0..7};do hccn_tool -i $i -tls -s enable 0;done
```

2. 获取每张卡的IP地址。

```
for i in {0..7};do hccn_tool -i $i -ip -g; done
```

3. 配置rank_table_file.json文件，并复制到每台机器上的${path-to-file}目录中。存放路径例如：/home/data/rank_table_file.json。详细样例参见**附录：rank_table_file.json文件**。注意：样例为4机部署配置，如果是2机部署则需要删除多余的配置，仅保留2机16卡的配置。
4. 设置rank_table_file.json文件权限。进入rank_table_file.json文件存放目录${path-to-file}，执行如下命令。

```
chmod 640 rank_table_file.json
```

步骤四：启动容器

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd --privileged \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /usr/local/Ascend/firmware:/usr/local/Ascend/firmware \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
```

```
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 表示需要在容器中挂载宿主机中文件在目录。dir为宿主机中的${path-to-file}目录，存放的是权重文件和rank_table_file.json文件，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将同一个NPU挂载给多个容器使用，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

步骤五：在每个节点进入容器并启动推理服务

1. 进入容器。
   ```
   docker exec -it -u root ${container-name} /bin/bash
   ```
2. 修改MindIE文件权限。
   ```
   cd /usr/local/Ascend/mindie/latest

   chmod 750 mindie-service
   chmod -R 550 mindie-service/bin
   chmod -R 500 mindie-service/bin/mindie_llm_backend_connector
   chmod 550 mindie-service/lib
   chmod 440 mindie-service/lib/*
   chmod 550 mindie-service/lib/grpc
   chmod 440 mindie-service/lib/grpc/*
   chmod -R 550 mindie-service/include
   chmod -R 550 mindie-service/scripts
   chmod 750 mindie-service/logs
   chmod 750 mindie-service/conf
   chmod 640 mindie-service/conf/config.json
   chmod 700 mindie-service/security
   chmod -R 700 mindie-service/security/*
   ```
3. 启动推理前需要先配置服务化环境变量。
   ```
   source /usr/local/Ascend/ascend-toolkit/set_env.sh
   source /usr/local/Ascend/nnal/atb/set_env.sh
   source /usr/local/Ascend/atb-models/set_env.sh
   unset HCCL_OP_EXPANSION_MODE
   export ATB_LLM_HCCL_ENABLE=1
   export ATB_LLM_COMM_BACKEND="hccl"
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
   export MIES_CONTAINER_IP=${container_ip}
   export RANKTABLEFILE=${RANKTABLEFILE}
   export HCCL_DETERMINISTIC=false
   export PARALLEL_PARAMS=[1,16,1,16,-1,-1]
   export ATB_WORKSPACE_MEM_ALLOC_ALG_TYPE=3
   export ATB_WORKSPACE_MEM_ALLOC_GLOBAL=1
   export NPU_MEMORY_FRACTION=0.96
   export HCCL_CONNECT_TIMEOUT=7200
   export HCCL_EXEC_TIMEOUT=0
   ```

- ${container_ip}：当前容器的IP地址，和rank_table_file.json文件中配置的container_ip保持一致。
- expandable_segments-使能内存池扩展段功能，即虚拟内存特性。
- ${RANKTABLEFILE}：rank_table_file.json文件挂载到容器中的地址${container_work_dir}/rank_table_file.json。
- PARALLEL_PARAMS=[dp,tp,moe_tp,moe_ep,pp,microbatch_size]，当前推荐配置为 tp=16, moe_ep=16。
- NPU_MEMORY_FRACTION：表示显存比。

4. 修改config.json文件中的服务化参数。config.json文件修改要求和样例参考**附录：config.json文件**。
```
cd /usr/local/Ascend/mindie/latest/mindie-service/
vim conf/config.json
```
5. 启动推理服务。
```
# 拉起服务化
cd /usr/local/Ascend/mindie/latest/mindie-service/
./bin/mindieservice_daemon
```
执行命令后出现“Daemon start success!”，表示服务成功启动。

步骤六：调用

调用DeepSeek-V3

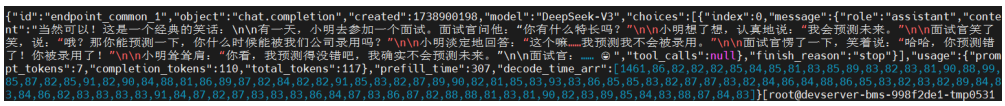
```
curl -ik -H 'Content-Type: application/json' -d '{"messages":[{"role":"user","content":"请讲一个笑话"}],"model":"DeepSeek-V3","temperature":0,"max_tokens":128}' -X POST http://${ip}:${port}/v1/chat/completions
```

调用DeepSeek-R1

```
curl -ik -H 'Content-Type: application/json' -d '{"messages":[{"role":"user","content":"请讲一个笑话"}],"model":"DeepSeek-R1","temperature":0,"max_tokens":128}' -X POST http://${ip}:${port}/v1/chat/completions
```

- ip：为**步骤五：在每个节点进入容器并启动推理服务**第4小步中配置的config.json中ipAddress值
- port：为**步骤五：在每个节点进入容器并启动推理服务**第4小步中配置的config.json中port字段的值

请求调用返回json参考如下：


{"id":"endpoint_common_1","object":"chat.completion","created":1738900198,"model":"DeepSeek-V3","choices":[{"index":0,"message":{"role":"assistant","content":"当然可以！这是一个经典的笑话：\n\n有一天，小明去参加一个面试。面试官问他：“你有什么特长吗？”\n\n小明想了想，认真地说：“我会预测未来。”\n\n面试官笑了笑，说：“哦？那你能预测一下，你什么时候能被我们公司录用吗？”\n\n小明淡定地回答：“这个嘛……我预测我不会被录用。”\n\n面试官愣了一下，笑着说：“哈哈，你预测错了！你被录用了！”\n\n小明耸耸肩：“你看，我预测得没错吧，我确实不会预测未来。”\n\n面试官：…… 😂","tool_calls":null},"finish_reason":"stop"}],"usage":{"prompt_tokens":7,"completion_tokens":110,"total_tokens":117},"prefill_time":307,"decode_time_arr":[1461,86,82,82,82,85,84,85,81,83,85,89,83,82,83,81,90,88,99,85,87,82,85,91,82,90,84,88,81,86,89,87,82,84,82,82,91,85,83,82,87,89,90,82,81,85,83,93,83,86,85,85,83,82,87,87,83,82,84,86,84,88,86,85,83,82,83,82,89,84,83,84,86,82,83,83,83,91,84,87,82,87,83,83,83,86,84,87,83,86,87,82,88,88,81,83,81,90,82,83,89,85,84,83,88,87,84,83]}[root@devserver-bms-998f2de1-tmp0531

3.1.4 附录：rank_table_file.json 文件

rank_table_file.json文件样例如下，需要根据实际修改server_count，device_ip，server_id，container_ip参数，每台机器上的rank_table_file.json文件内容一致。在**步骤三：创建rank_table_file.json**步骤中会用到。

- server_count：节点个数。当前默认为2。
- device_ip：当前卡的IP地址，2台机器共16张卡。device_ip查询命令
```
for i in {0..7};do hccn_tool -i $i -ip -g; done
```
- server_id：当前Server节点的IP地址，涉及2台机器。
- container_ip：容器IP地址，无特殊配置时与server_id保存一致。

```
{
   "server_count": "2",
```

```
"server_list": [
    {
        "device": [
            {
                "device_id": "0",
                "device_ip": "29.82.85.12",
                "rank_id": "0"
            },
            {
                "device_id": "1",
                "device_ip": "29.82.98.67",
                "rank_id": "1"
            },
            {
                "device_id": "2",
                "device_ip": "29.82.133.21",
                "rank_id": "2"
            },
            {
                "device_id": "3",
                "device_ip": "29.82.175.69",
                "rank_id": "3"
            },
            {
                "device_id": "4",
                "device_ip": "29.82.13.154",
                "rank_id": "4"
            },
            {
                "device_id": "5",
                "device_ip": "29.82.140.51",
                "rank_id": "5"
            },
            {
                "device_id": "6",
                "device_ip": "29.82.157.87",
                "rank_id": "6"
            },
            {
                "device_id": "7",
                "device_ip": "29.82.15.225",
                "rank_id": "7"
            }
        ],
        "server_id": "7.242.110.112",
        "container_ip": "7.242.110.112"
    },
    {
        "device": [
            {
                "device_id": "0",
                "device_ip": "29.82.177.28",
                "rank_id": "8"
            },
            {
                "device_id": "1",
                "device_ip": "29.82.41.231",
                "rank_id": "9"
            },
            {
                "device_id": "2",
                "device_ip": "29.82.16.3",
                "rank_id": "10"
            },
            {
                "device_id": "3",
                "device_ip": "29.82.154.20",
                "rank_id": "11"
            },
```



```
            {
               "device_id": "4",
               "device_ip": "29.82.56.73",
               "rank_id": "12"
            },
            {
               "device_id": "5",
               "device_ip": "29.82.177.138",
               "rank_id": "13"
            },
            {
               "device_id": "6",
               "device_ip": "29.82.29.230",
               "rank_id": "14"
            },
            {
               "device_id": "7",
               "device_ip": "29.82.1.176",
               "rank_id": "15"
            }
         ],
         "server_id": "7.242.104.54",
         "container_ip": "7.242.104.54"
      }
   ],
   "status": "completed",
   "version": "1.0"
}
```


3.1.5 附录：config.json 文件

config.json文件用于推理服务启动时，需要修改以下参数，2台机器的每个容器中config.json文件内容一致。

- ipAddress：主节点IP地址，即**rank_table_file.json文件**中的server_id。
- managementIpAddress：主节点IP地址，和ipAddress取值一致。
- httpsEnabled：取值需要修改为false。
- interCommTLSEnabled和interNodeTLSEnabled：如果不需要开启安全认证，这2个参数取值需要修改为false。
- multiNodesInferEnabled：取值需要修改true，表示开启多机推理。
- modelName：设置为DeepSeek-V3或DeepSeek-R1。
- modelWeightPath：权重文件在容器内的地址，例如：${container_work_dir}/deepseekV3-w8a8或$${container_work_dir}/deepseekR1-w8a8目录。${container_work_dir}在**步骤四：启动容器**时定义。
- maxPrefillBatchSize：最大prefill batch size。config.json文件中默认是50，并发请求数量超出设置，推理请求会被拒绝。用户可以根据实际修改。maxPrefillBatchSize和maxPrefillTokens谁先达到各自的取值就完成本次组batch。
- maxSeqLen：输入长度+输出长度的最大值。该值为maxInputTokenLen +maxIterTimes的和。config.json文件中默认是16k，用户可以根据自己的推理场景设置。
- maxInputTokenLen：输入最大长度。config.json文件中默认是15k，用户可以根据自己的推理场景设置。
- maxPrefillTokens：最大prefill token数。和maxInputTokenLen保持相同。
- maxIterTimes：最大输出长度。config.json文件中默认是1k，用户可以根据自己的推理场景设置。

说明

当前在W8A8量化权重、2台Ascend Snt9B资源下支持的maxSeqLen最大为32768。

```
{
    "Version" : "1.0.0",
    "LogConfig" :
    {
        "logLevel" : "Info",
        "logFileSize" : 20,
        "logFileNum" : 20,
        "logPath" : "logs/mindie-server.log"
    },

    "ServerConfig" :
    {
        "ipAddress" : "7.242.110.112",
        "managementIpAddress" : "7.242.110.112",
        "port" : 1025,
        "managementPort" : 1026,
        "metricsPort" : 1027,
        "allowAllZeroIpListening" : false,
        "maxLinkNum" : 1000,
        "httpsEnabled" : false,
        "fullTextEnabled" : false,
        "tlsCaPath" : "security/ca/",
        "tlsCaFile" : ["ca.pem"],
        "tlsCert" : "security/certs/server.pem",
        "tlsPk" : "security/keys/server.key.pem",
        "tlsPkPwd" : "security/pass/key_pwd.txt",
        "tlsCrlPath" : "security/certs/",
        "tlsCrlFiles" : ["server_crl.pem"],
        "managementTlsCaFile" : ["management_ca.pem"],
        "managementTlsCert" : "security/certs/management/server.pem",
        "managementTlsPk" : "security/keys/management/server.key.pem",
        "managementTlsPkPwd" : "security/pass/management/key_pwd.txt",
        "managementTlsCrlPath" : "security/management/certs/",
        "managementTlsCrlFiles" : ["server_crl.pem"],
        "kmcKsfMaster" : "tools/pmt/master/ksfa",
        "kmcKsfStandby" : "tools/pmt/standby/ksfb",
        "inferMode" : "standard",
        "interCommTLSEnabled" : false,
        "interCommPort" : 1121,
        "interCommTlsCaPath" : "security/grpc/ca/",
        "interCommTlsCaFiles" : ["ca.pem"],
        "interCommTlsCert" : "security/grpc/certs/server.pem",
        "interCommPk" : "security/grpc/keys/server.key.pem",
        "interCommPkPwd" : "security/grpc/pass/key_pwd.txt",
        "interCommTlsCrlPath" : "security/grpc/certs/",
        "interCommTlsCrlFiles" : ["server_crl.pem"],
        "openAiSupport" : "vllm"
    },

    "BackendConfig" : {
        "backendName" : "mindieservice_llm_engine",
        "modelInstanceNumber" : 1,
        "npuDeviceIds" : [[0,1,2,3]],
        "tokenizerProcessNumber" : 8,
        "multiNodesInferEnabled" : true,
        "multiNodesInferPort" : 1120,
        "interNodeTLSEnabled" : false,
        "interNodeTlsCaPath" : "security/grpc/ca/",
        "interNodeTlsCaFiles" : ["ca.pem"],
        "interNodeTlsCert" : "security/grpc/certs/server.pem",
        "interNodeTlsPk" : "security/grpc/keys/server.key.pem",
        "interNodeTlsPkPwd" : "security/grpc/pass/mindie_server_key_pwd.txt",
        "interNodeTlsCrlPath" : "security/grpc/certs/",
        "interNodeTlsCrlFiles" : ["server_crl.pem"],
        "interNodeKmcKsfMaster" : "tools/pmt/master/ksfa",
```

```
    "interNodeKmcKsfStandby" : "tools/pmt/standby/ksfb",
    "ModelDeployConfig" :
    {
       "maxSeqLen" : 16384,
       "maxInputTokenLen" : 15360,
       "truncation" : false,
       "ModelConfig" : [
          {
             "modelInstanceType" : "Standard",
             "modelName" : "DeepSeek-V3",
             "modelWeightPath" : "/data/model/DeepSeek-V3-w8a8",
             "worldSize" : 4,
             "cpuMemSize" : 5,
             "npuMemSize" : -1,
             "backendType" : "atb",
             "trustRemoteCode" : false
          }
       ]
    },

    "ScheduleConfig" :
    {
       "templateType" : "Standard",
       "templateName" : "Standard_LLM",
       "cacheBlockSize" : 128,

       "maxPrefillBatchSize" : 50,
       "maxPrefillTokens" : 15360,
       "prefillTimeMsPerReq" : 150,
       "prefillPolicyType" : 0,

       "decodeTimeMsPerReq" : 50,
       "decodePolicyType" : 0,

       "maxBatchSize" : 200,
       "maxIterTimes" : 1024,
       "maxPreemptCount" : 0,
       "supportSelectBatch" : false,
       "maxQueueDelayMicroseconds" : 5000
    }
  }
}
```

3.1.6 附录：部署常见问题

如何解决 DeepSeek-R1 概率不触发深度思考的问题

问题：DeepSeek-R1概率不触发深度思考

解决方法：在prompt最后面添加“<think>\n”。

```
{
  "messages": [{
     "role": "user",
     "content": "请讲一个笑话<think>\n"
  }]
}
```

如何解决“Available shared memory size is not enough“的问题

问题：容器共享内存不足

解决方法：在启动docker的命令中增加“--shm-size=${memSize}“，其中memSize为要设置的共享内存大小，如2g。

```
--shm-size 2g \
```

如何解决 MindIE 服务已退出情况下显存依然占用的问题

问题：服务退出显存未完全释放

解决方法：重启机器。

MindIE 和 vLLM 在深度思考返回接口差异

对于DeepSeek-R1这类模型，其返回信息包含深度思考的内容。

当前MindIE接口深度思考内容和问题回答都在content字段中，类似“xxx</think>xxx“，通常</think>前面的即为深度思考内容，后面的为问题回答。MindIE接口详见**MindIE推理接口**。

```
{
    "content":"xxx</think>xxx"
}
```

而vLLM框架则对深度思考内容做了处理，将深度思考的内容放在reasoning_content字段，content字段只有问题回答。

```
{
    "content":"xxx",
    "reasoning_content":"xxx"
}
```

如何解决“AttributeError：'IbisTokenizer' object has no atrribute 'cache_path' ”问题

若出现AttributeError：'IbisTokenizer' object has no atrribute 'cache_path' 问题。

详细解决方案参考**ModelZoo社区**。

3.2 基于 MaaS DeepSeek API 和 Dify 快速构建网站智能客服

本文介绍如何使用MaaS（大模型即服务平台）的免费Token额度的满血版DeepSeek-R1 API接入Dify（开源Agent平台），快速构建AI对话机器人并将其嵌入在网页页面中。当免费Token额度用完后，您也可以付费部署为我的服务使用。更多信息，请参见**计费说明**和**使用MaaS部署模型服务**。

背景介绍

客服系统是企业与消费者沟通的重要桥梁，然而传统的人工客服存在着人工成本高昂、数据收集和分析困难、难以24小时提供服务等痛点。

随着AI大模型的崛起，如何利用先进的大语言模型构建AI智能体并应用于智能客服系统，提供更加自然、流畅的对话体验，提高用户使用效率，实现24*7的无间断服务，降低企业在客服场景下的投入成本，提升用户满意度成为企业和用户关注的热门领域。

步骤一：在 MaaS 控制台领取免费体验额度

MaaS提供了一系列可以Severless方式免费调用的模型API。

1. 登录**ModelArts Studio控制台**，在顶部导航栏选择“西南-贵阳一”区域。
2. 在左侧导航栏，单击“模型部署”。
3. 在“模型部署”页面的“预置服务”页签，单击DeepSeek服务右侧的“领取”。

图 3-1 领取免费 Token 额度

当“领取”置灰时，表示该服务的免费额度已领取。您可以付费部署为我的服务使用，详情请参见**使用MaaS部署模型服务**。

步骤二：搭建网站智能客服

Dify是一个能力丰富的开源AI应用开发平台，为大型语言模型（LLM）应用的开发而设计。它巧妙地结合了后端即服务（Backend as Service）和LLMOps的理念，提供了一套易用的界面和API，加速了开发者构建可扩展的生成式AI应用的过程。

1. 基于Flexus云服务器X实例快速部署Dify平台。具体操作，请参见**快速搭建Dify-LLM应用开发平台**。
2. 部署完成后，登录Dify。

   首次登录需注册管理员账号，依次填写邮箱、账号、密码后，单击“设置”进行保存后，即可登录使用。

图 3-2 注册管理员账户

3. 将MaaS平台的模型服务接入Dify。
   a. 创建API Key。
      i. 登录**ModelArts Studio控制台**，在顶部导航栏选择“西南-贵阳一”区域。

ii. 在左侧导航栏，单击“鉴权管理”。

iii. 在“鉴权管理”页面，单击“创建API Key”，填写描述信息后，单击“确认”会返回“您的密钥”，请复制保存密钥，单击“关闭”后将无法再次查看密钥。

最多支持创建5个密钥，密钥只会在新建后显示一次，请妥善保存。当密钥丢失将无法找回，请新建API Key获取新的访问密钥。

b. 选择常驻模型API。

i. 方式一：使用预置服务接入。

1) 在**ModelArts Studio控制台**左侧导航栏，在“模型部署”页面的“预置服务”页签，在已领取额度的模型服务右侧，单击“调用”。

2) 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息。

图 3-3 获取 API 地址和模型名称

ii. 方式二：使用我的服务接入。

1) 在**ModelArts Studio控制台**左侧导航栏，单击“模型部署”。

2) 在“模型部署”页面，单击“我的服务”页签。

3) 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用”。

4) 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息。

图 3-4 获取 API 地址和模型名称

c. 配置Dify模型供应商。

i. 在Dify平台右上角，单击用户头像，选择“设置”。
ii. 在“设置”页面左侧导航栏，单击“模型供应商”。
iii. 在“模型供应商”页面，找到“OpenAI-API-compatible”供应商并单击“添加模型”。
iv. 在“添加 OpenAI-API-compatible”对话框，配置相关参数，然后单击“保存”。

图 3-5 配置 Dify 模型供应商

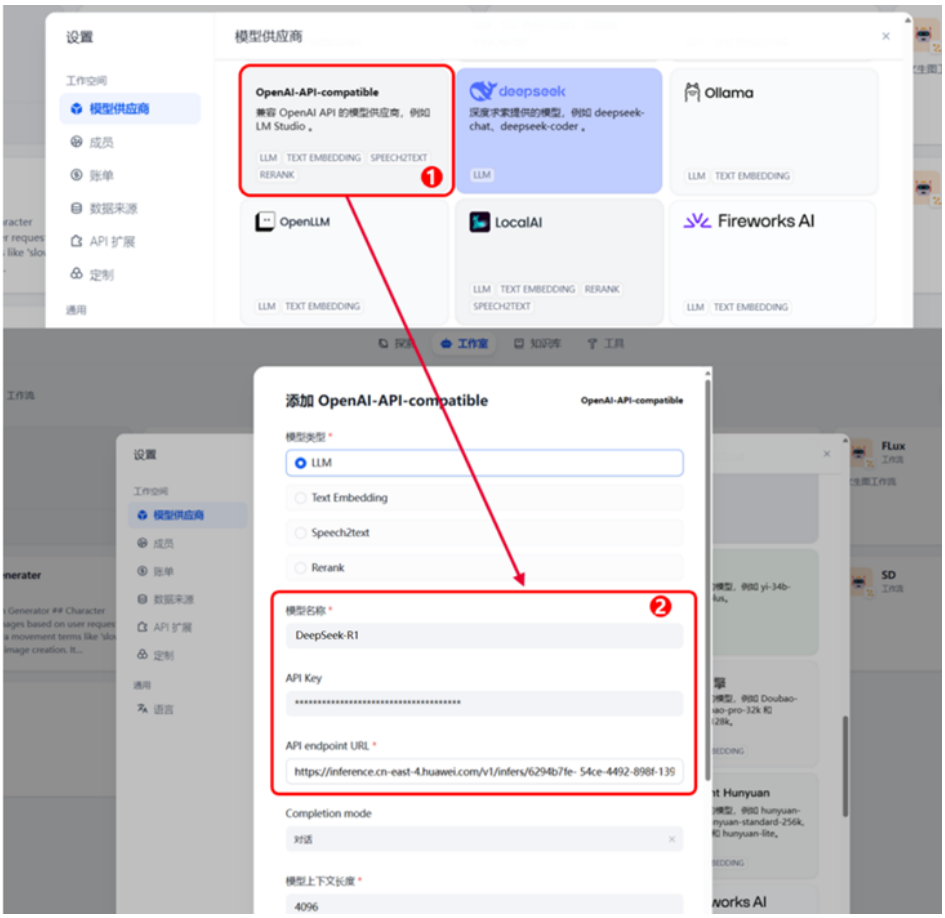


表 3-1 添加 OpenAI-API-compatible 参数说明

参数	说明
模型类型	选择“LLM”。
模型名称	步骤3.b显示的模型名称。
API Key	步骤3.a创建的贵阳一区域的API Key。
API Endpoint URL	步骤3.b获取的MaaS服务的基础API地址，需要去掉地址尾部的“/chat/completions”后填入。

参数	说明
Function calling	当前MaaS预置服务中仅Qwen2_5-7B-Instruct-1128、Qwen2_5-72B-Instruct-1128、Qwen2_5-72B-32K-1128可以配置Function calling为“Tool Call”，其余服务暂不支持。

步骤三：在 Dify 中创建客服机器人

如果想让智能客服能够基于企业内部的知识文档进行问答，可以在Dify中创建一个知识库。Dify能够协助进行数据的分段和清理工作。

Dify提供两种索引方式：高质量和经济。两种索引的区别如下表所示。下文使用经济索引进行演示。

表 3-2 高质量索引与经济索引的区别

比较项目	高质量索引	经济索引
索引原理	通过嵌入模型将文本块转换为数字向量，进行精确匹配，可使用向量检索、全文检索、混合检索。	每个文本块仅使用10个关键词，采用倒排索引方法选择相关文本块。
检索精度	精度高，可使用重排模型等优化结果。	相对较低。
资源消耗	启用重排模型等功能会消耗模型Tokens，对系统资源要求较高。	不消耗Tokens，对资源要求低。
适用场景	适用于对检索准确性和结果质量要求高的场景，例如专业知识查询、科研文献检索等自然语言文本场景。	适用于对检索速度要求较高、对精度要求相对较低的一般性场景，例如快速浏览大量文档获取大致信息。
检索设置	有向量检索、全文检索、混合检索三种检索设置，可配置Rerank模型等。	只提供倒排索引方法，无复杂检索设置和参数配置。

1. 创建知识库。
   a. 在Dify平台左侧导航栏，单击“知识库”，选择“创建知识库”。
   b. 在“选择数据源”页面，上传知识文档（可以同时上传多个文本文件），“索引方式”选择“经济”，配置其他信息，单击“保存并处理”。

图 3-6 创建知识库

2. 创建并调试客服机器人。
   a. 在“工作室”页签，单击“创建空白应用”，“应用类型”选择“Agent”，输入名称和描述，进入AI的编排页面。
   b. 进行AI人设设置，在左上角“提示词”处输入Prompt ，设定该AI与用户交谈时的语气和风格。

      智能客服提示词示例：你是一个友好的智能客服助手，负责解答用户提出的关于产品的问题、处理订单、提供技术支持。当用户提出的问题你没有找到合适答案的时候，要回答不知道。
   c. 在“上下文”处添加**步骤三.1**创建的知识库，让AI基于内部知识回答问题。
   d. 单击右下角功能的“管理”打开功能开关，配置开场白，即用户进入界面后AI的开场问候语，例如：您好，有什么可以帮到您的?
   e. 在“添加功能”处开启“下一步问题建议配置”功能，基于业务场景设置AI对于用户问题的指引，例如提供常见问题选项或引导用户进一步说明问题。
   f. 完成以上配置后，单击右上角“发布”菜单中“更新”保存设置。您可以在页面右侧窗口输入Prompt进行效果调试。

图 3-7 创建与调试客服机器人

3. 将客服机器人嵌入网页前端页面。
   a. 在Dify平台中完成客服机器人的创建与调试后，单击“发布”，选择“嵌入网站”。
   b. 选择嵌入方式。本文以第二种“悬浮窗式”为例，将下方的代码复制到您网站<head>或<body> 标签中，更新网站页面，即可与该客服机器人进行对话。

      Dify支持全屏界面式、悬浮窗式、浏览器插件式三种嵌入方式，请您根据需求选择合适的方式。

图 3-8 选择嵌入方式

3.3 基于 MaaS DeepSeek API 和 Cherry Studio 快速构建个人 AI 智能助手

本文介绍如何使用Cherry Studio调用部署在ModelArts Studio上的DeepSeek模型，构建个人AI助手。

背景介绍

Cherry Studio是一款开源的多模型桌面客户端，支持Windows、macOS和Linux系统。它集成了多种主流大语言模型（例如OpenAI、DeepSeek、Gemini等），并支持本地模型运行。此外，它还具备丰富的功能，例如对话知识库、AI绘画、翻译、多模型切换等。

ModelArts Studio（简称MaaS）服务将DeepSeek系列模型部署到平台中，支持广大开发者在线体验或端外调用。同时，MaaS提供免费Token支持开发者免费使用，帮助开发者快速验证和实现创新应用。更多信息，请参见**免费体验MaaS预置服务**。

步骤一：下载并安装 Cherry Studio

您可以通过**官方网站**或**开源地址**下载并安装Cherry Studio。

步骤二：MaaS 模型 API 接入准备

1. 创建API Key。
   a. 登录**ModelArts Studio控制台**，在顶部导航栏选择“西南-贵阳一”区域。
   b. 在左侧导航栏，单击“鉴权管理”。
   c. 在“鉴权管理”页面，单击“创建API Key”，填写描述信息后，单击“确认”会返回“您的密钥”，请复制保存密钥，单击“关闭”后将无法再次查看密钥。

      最多支持创建5个密钥，密钥只会在新建后显示一次，请妥善保存。当密钥丢失将无法找回，请新建API Key获取新的访问密钥。
2. 选择要接入的模型服务。
   - 方式一：使用预置服务接入。
     i. 在**ModelArts Studio控制台**左侧导航栏，单击“模型部署”。
     ii. 在“模型部署”页面的“预置服务”页签，单击DeepSeek服务右侧的“领取”。

     图 3-9 领取免费 Token 额度

     iii. 在已领取额度的模型服务右侧，单击“调用”。
     iv. 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Cherry Studio配置中使用。

图 3-10 获取 API 地址和模型名称

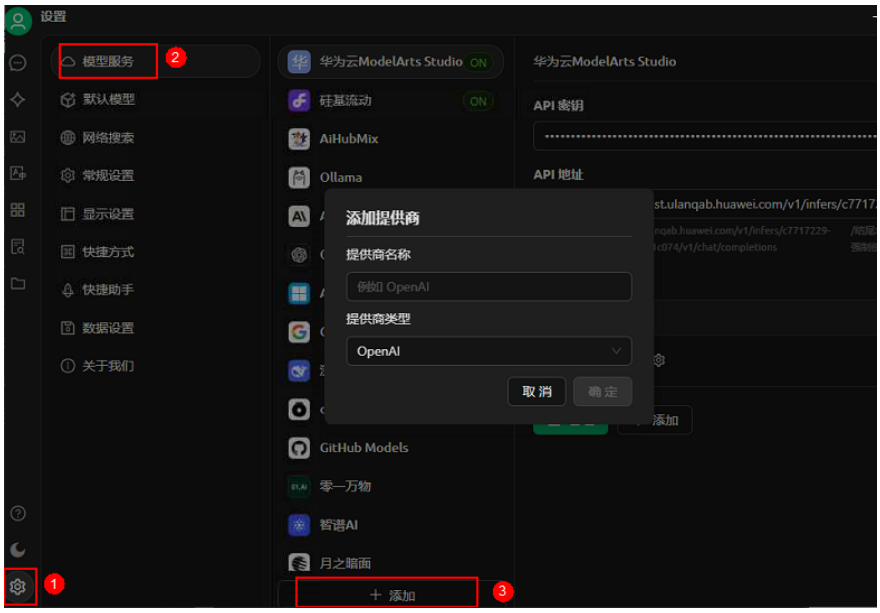
- 方式二：使用我的服务接入。
  i. 在**ModelArts Studio控制台**左侧导航栏，单击“模型部署”。
  ii. 在“模型部署”页面，单击“我的服务”页签，在右上角单击“模型部署服务”，创建模型部署服务。具体操作，请参见**使用MaaS部署模型服务**。
  iii. 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用”。
  iv. 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Cherry Studio配置中使用。

图 3-11 获取 API 地址和模型名称

步骤三：在 Cherry Studio 中配置 MaaS API

1. 添加MaaS提供商。
   a. 在Cherry Studio客户端左下角，单击设置图标，在“模型服务”中单击“添加”。

图 3-12 添加提供商



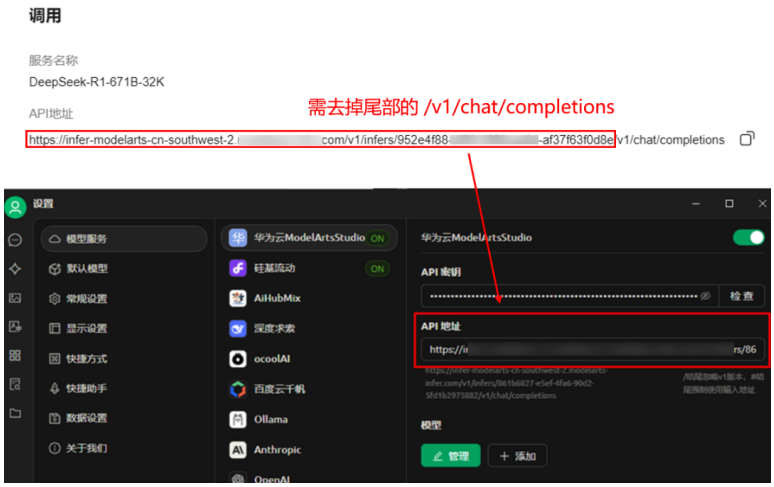
b. 在“添加提供商”对话框，配置提供商名称和提供商类型，然后单击“确定”。

表 3-3 添加提供商参数说明

参数	说明
提供商名称	配置为“华为云ModelArts Studio”，您可以按需修改。
提供商类型	配置为“OpenAI”。

2. 添加API密钥和API地址。
   a. 在Cherry Studio客户端左下角，单击设置图标。
   b. 在“设置”页面，找到“华为云ModelArts Studio”选项，配置API密钥和API地址。

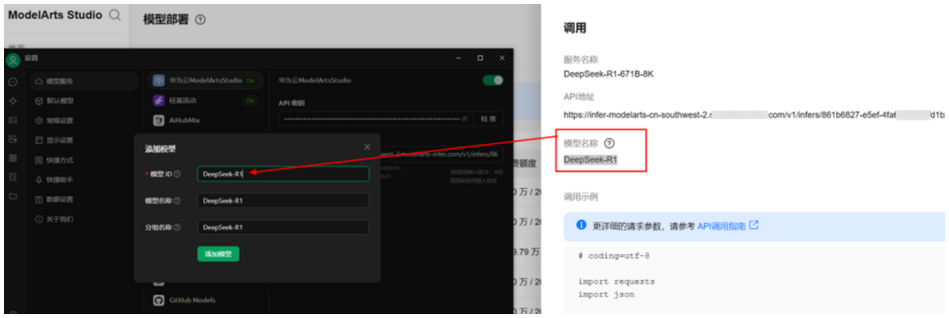
图 3-13 添加 API 密钥和 API 地址



参数	说明
API密钥	步骤二.1创建的API Key。
API地址	步骤二.2获取的MaaS服务的基础API地址，需要去掉地址尾部的“v1/chat/completions”后填入。

3. 添加模型。
   a. 在“模型”区域，单击“添加”。
   b. 在“添加模型”对话框，配置模型ID、模型名称和分组名称，单击“添加模型”。

图 3-14 添加模型

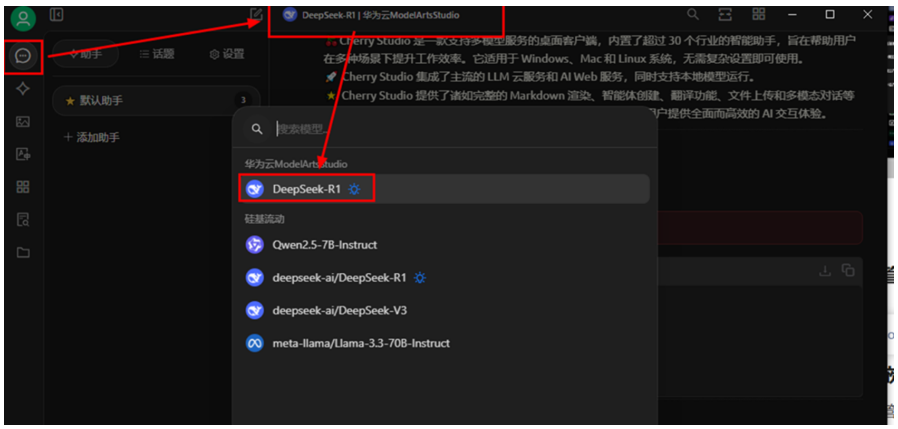


参数	说明
模型 ID	步骤二.2获取的模型名称。
模型名称	自定义模型名称。
分组名称	自定义分组名称。

步骤四：在 Cherry Studio 中使用 MaaS API

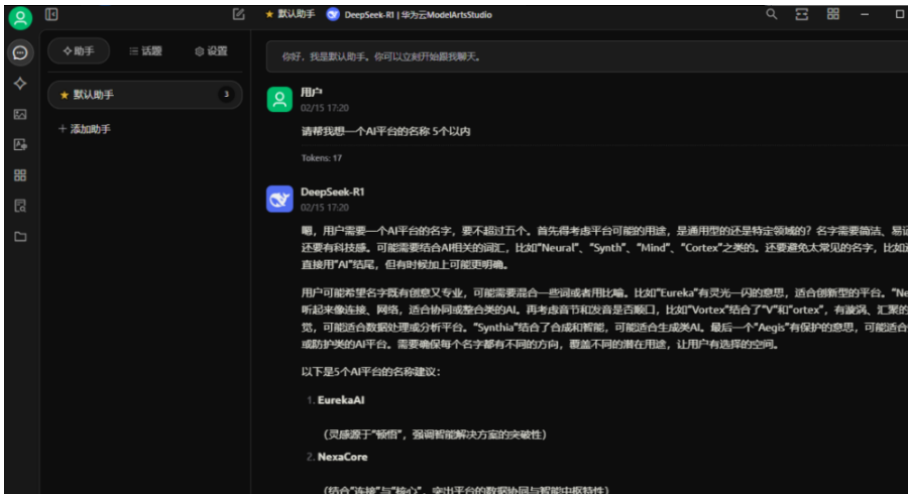
1. 在Cherry Studio左侧导航栏，单击  图标，选择已配置好的模型。

图 3-15 选择模型



2. 在文本框中输入文字，开始对话。
   您也可以选择顶部菜单中的模型名字切换模型。

   图 3-16 模型问答示例



3.4 基于 MaaS DeepSeek API 和 Chatbox 快速构建文案编辑器

本文介绍如何使用Chatbox调用部署在ModelArts Studio上的DeepSeek模型，构建文案编辑器。

背景介绍

Chatbox是一款开源的跨平台AI客户端应用，支持多种主流语言模型（例如OpenAI的GPT系列、DeepSeek、Claude、Google Gemini Pro等），并具备本地数据存储、图像生成、代码辅助、文档交互、联网搜索、Markdown和LaTeX支持等功能。它还支持团队协作，提供丰富的多语言交互能力，确保用户数据隐私和安全。

ModelArts Studio（简称MaaS）服务将DeepSeek系列模型部署到平台中，基于华为云昇腾云服务的全栈优化适配，可获得持平全球高端GPU部署模型的效果，提供稳定的生产级服务能力，满足业务商用部署需求，支持广大开发者在线体验或端外调用。同时，MaaS提供免费Token支持开发者免费使用，帮助开发者快速验证和实现创新应用。更多信息，请参见**免费体验MaaS预置服务**。

步骤一：下载并安装 Chatbox

您可以通过**官方网站**或**Github**下载并安装Chatbox。

步骤二：MaaS 模型 API 接入准备

1. 创建API Key。
   a. 登录**ModelArts Studio控制台**，在顶部导航栏选择“西南-贵阳一”区域。
   b. 在左侧导航栏，单击“鉴权管理”。
   c. 在“鉴权管理”页面，单击“创建API Key”，填写描述信息后，单击“确认”会返回“您的密钥”，请复制保存密钥，单击“关闭”后将无法再次查看密钥。

最多支持创建5个密钥，密钥只会在新建后显示一次，请妥善保存。当密钥丢失将无法找回，请新建API Key获取新的访问密钥。

2. 选择要接入的模型服务。
   - 方式一：使用预置服务接入。
     i. 在**ModelArts Studio控制台**左侧导航栏，单击“模型部署”。
     ii. 在“模型部署”页面的“预置服务”页签，单击DeepSeek服务右侧的“领取”。

     图 3-17 领取免费 Token 额度

     iii. 在已领取额度的模型服务右侧，单击“调用”。
     iv. 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Chatbox配置中使用。

     图 3-18 获取 API 地址和模型名称

   - 方式二：使用我的服务接入。
     i. 在**ModelArts Studio控制台**左侧导航栏，单击“模型部署”。
     ii. 在“模型部署”页面，单击“我的服务”页签，在右上角单击“模型部署服务”，创建模型部署服务。具体操作，请参见**使用MaaS部署模型服务**。
     iii. 在“状态”为“运行中”的模型服务右侧，单击操作列下的“更多 > 调用”。
     iv. 在“调用”页面，可以查看调用该服务需要的基础API地址和模型名称信息，在后续Chatbox配置中使用。

图 3-19 获取 API 地址和模型名称

步骤三：在 Chatbox 中配置 MaaS API

1. 在Chatbox平台左下角，单击“设置”。
2. 在“设置”对话框，单击模型提供方下拉框，选择“添加自定义提供方”，配置相关信息，单击“保存”。

图 3-20 添加自定义提供方

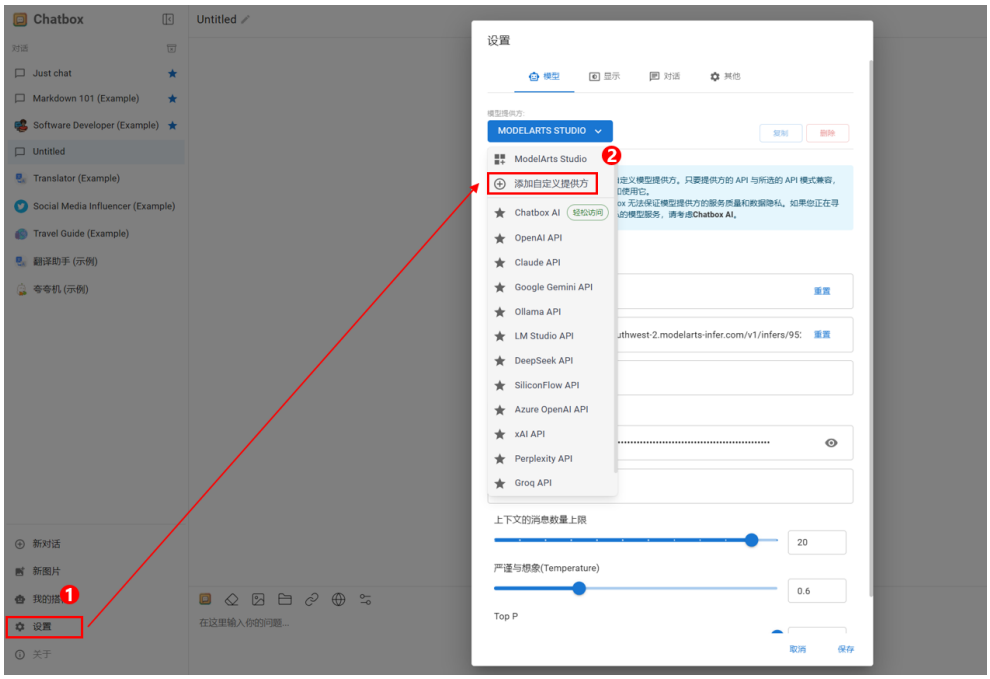


图 3-21 添加自定义提供方参数说明

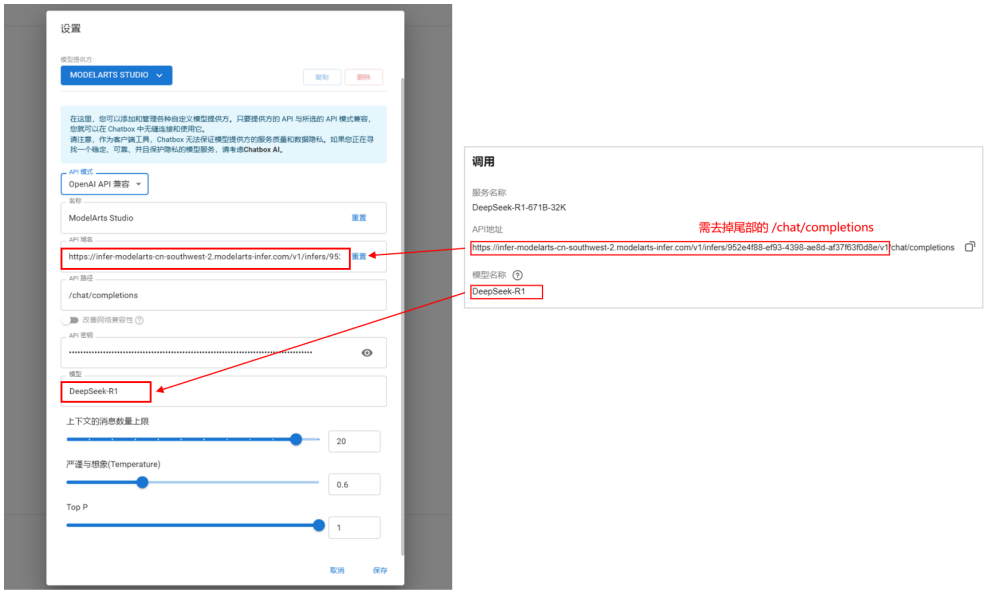


表 3-4 添加自定义提供方参数说明

参数	说明
API模式	默认为“OpenAI API兼容” 。
名称	填写“ModelArts Studio”，您可以自定义修改。
API域名	**步骤二.2**获取的API地址，需要去掉地址尾部的“/chat/completions” 后填入。
API路径	默认为“/chat/completions” 。
API密钥	**步骤二.1**创建的API Key。
模型	**步骤二.2**获取的模型名称。

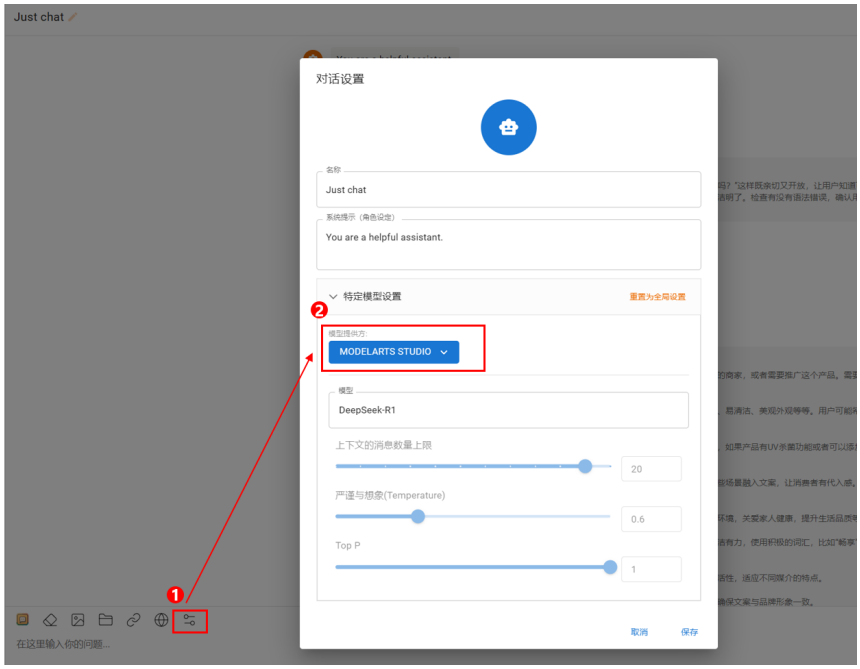
步骤四：在 Chatbox 中使用 MaaS API

1. 在Chatbox平台左下角，单击“我的搭档” 。
2. 在“我的搭档”页面，按需选择场景化模板。

图 3-22 选择场景化模板

3. 在新的对话窗口，单击左下方的配置图标，在“对话设置”对话框，选择已配置的ModelArts Studio提供方，单击“保存”。

图 3-23 对话设置



4. 使用MaaS的模型进行多轮对话、文案生成、摘要提取等操作。

图 3-24 文案生成示例

4 LLM 大语言模型训练推理

4.1 在 ModelArts Studio 基于 Qwen2-7B 模型实现新闻自动分类

说明

仅“华东二”和“西南-贵阳一”区域支持使用ModelArts Studio大模型即服务平台（MaaS）。

应用场景

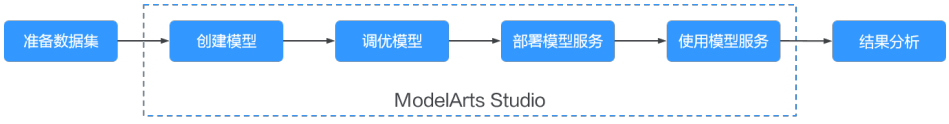
在数字化时代，新闻的生成与传播速度不断刷新记录。在ModelArts Studio大模型即服务平台（下面简称为MaaS），使用Qwen2-7B模型可以实现新闻自动分类，能够高效处理和分类大量新闻内容。

该解决方案可以应用于如下场景：

- 新闻门户网站： 自动将新闻内容归类到相应板块，如科技、体育或国际新闻，以提升用户体验和内容检索效率。
- 社交媒体平台： 对用户分享的新闻链接进行智能分类，帮助用户迅速定位到感兴趣的话题。
- 内容推荐系统： 根据用户的阅读偏好和历史行为，智能推荐相关新闻，增强用户粘性和满意度。
- 新闻分析工具： 为分析师提供自动分类的新闻数据，便于进行市场趋势和热点分析。

方案流程

图 4-1 方案实现流程



1. 准备数据集：获取新闻数据集，并上传到OBS。

2. 创建模型：选择Qwen2-7B基础模型，使用推荐权重创建个人专属模型。
3. 调优模型：使用不同的调优参数去训练模型。
4. 部署模型服务：将调优后的模型部署成模型服务。
5. 使用模型服务：在MaaS体验模型服务，测试推理结果。
6. 结果分析：分析模型的调优结果和推理结果，对比新闻分类效果。

方案优势

- 高准确性：利用模型强大的语义理解能力，系统能够准确识别新闻内容的主题和关键词，实现高准确率的自动分类。
- 快速响应：系统能够实时处理新闻内容，快速完成分类，满足新闻时效性的要求。
- 可扩展性：随着模型的不断训练和优化，系统能够适应不断变化的新闻内容和分类需求。
- 降低人力成本：减少人工分类的工作量，降低人力成本，提高工作效率。

操作步骤

步骤1 准备数据集。获取新闻数据集，并上传到OBS。

1. 下载新闻数据集。

   本文原始数据集来源：**https://github.com/aceimnorstuvwxz/toutiao-text-classfication-dataset**

   本文实验用数据集基于原始数据集处理而来，进行了简单的采样、清晰和prompt工程。

   实验数据集获取地址：**https://maas-operations.obs.myhuaweicloud.com/Sample-Dataset/maas_demo_news.jsonl**

   该实验数据集的总数据量5281条，随机分为训练集5120条和测试集161条。

   实验数据集文件“maas_demo_news.jsonl”的内容格式如下。

   ```
   {"conversation_id": 1, "chat": {"turn_1": {"Human":"text","MOSS":"text"},"turn_2": {"Human":"text","MOSS":"text"}}}
   ```

   “conversation_id”是样本编号，“chat”后面是多轮对话的内容，“turn_n”表示是第n次对话，每次对话都有输入（对应Human角色）和输出（对应MOSS角色）。其中Human和MOSS仅用于角色区分，模型训练的内容只有text指代的文本。
2. 将jsonl格式的数据集文件上传到“华东二”区域的OBS桶中，创建OBS桶和上传文件的操作指导请参见**OBS控制台快速入门**。本文以“/maas-test/news/maas_demo_news.jsonl”OBS路径为例。

步骤2 进入ModelArts Studio大模型即服务平台。

1. 登录ModelArts管理控制台。
2. 在左侧导航栏中，选择“ModelArts Studio”进入ModelArts Studio大模型即服务平台。

步骤3 创建Qwen2-7B模型。

1. 在ModelArts Studio左侧导航栏中，选择“我的模型”进入模型列表。
2. 在模型列表页，单击“创建模型”弹出创建模型页面。
3. 在“创建模型”页面，配置参数。

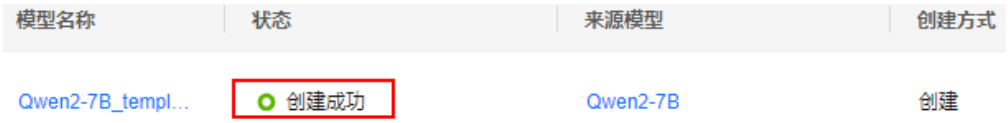
图 4-2 创建模型

表 4-1 创建模型

参数	说明	取值样例
来源模型	单击“选择基础模型”，在弹窗中选择模型，单击“确定”。	Qwen2-7B
模型名称	自定义模型名称。	Qwen2-7B_template
描述	自定义模型简介。	-
权重设置与词表	默认选择“使用推荐权重”，支持选择“自定义权重”。 使用平台推荐的权重文件，可提高模型的训练、压缩、部署和调优等服务的使用效率。 权重文件指的是模型的参数集合。	使用推荐权重

4. 参数配置完成后，单击“创建”，创建个人专属模型。
5. 在模型列表，单击模型名称可以进入详情页查看模型详细信息和任务。
   当模型“状态”变成“创建成功”时，表示模型创建完成。

图 4-3 查看我的模型状态



步骤4 调优模型，使用6种不同的调优参数去训练模型。

1. 模型创建成功后，在“我的模型”列表，单击操作列的“调优”。
2. 在“创建模型调优任务”页面，配置参数。

   由于需要分析模型调优效果，需要创建多个调优任务，不同调优任务的参数值配置请参见**表4-2**和**表4-3**。

表 4-2 创建调优任务

参数		说明	取值样例
任务设置	任务名称	自定义调优任务名称。	参见**表4-3**
	描述	自定义调优任务简介。	-
模型设置	来源模型	当从“我的模型”列表进入创建调优作业页面时，此处默认呈现选择的模型。	Qwen2-7B_template
	调优类型	- **全参微调**：直接在模型上训练，影响模型全量参数的微调训练，效果较好，收敛速度较慢，训练时间较长。 - **LoRA微调**：冻结原模型，通过往模型中加入额外的网络层，并只训练这些新增的网络层参数，效果接近或略差于全参训练，收敛速度快，训练时间短。 - **增量预训练**：在现有预训练模型基础上，利用新数据或特定领域的数据增强模型的能力和性能。允许模型逐步适应新的任务和数据，避免过拟合和欠拟合问题，进一步提高模型的泛化能力。	参见**表4-3**
	调优后模型名称	设置调优后产生的新模型的名称。	参见**表4-3**
	调优后模型权重存放路径	选择调优后模型权重文件的OBS存放路径。训练后将在指定路径下自动创建以作业ID命名的新文件夹进行权重存储。	/maas-test/news/out

参数		说明	取值样例
数据设置	选择数据集格式	支持选择MOSS、Alpaca和ShareGPT。训练数据需要按照对应格式，上传符合规范的数据集，以更好完成训练任务。关于数据集示例，请参见**支持的数据集格式**。 **说明** 如果数据集选择错误，您可以通过以下方式查看日志详情。 – 登录ModelArts Studio控制台，在“模型调优”页面单击目标作业，在作业详情的日志页签查看详情。 – 登录ModelArts控制台，在“模型训练 > 训练作业”页面单击目标作业，在日志页签查看详情。	MOSS
	添加数据集	选择存放训练数据集的OBS路径，必须选择到**步骤1**上传的jsonl文件。	/maas-test/news/maas_demo_news.jsonl
超参设置	数据条数	输入数据集中的总数据条数。	1000
	迭代轮次/Epoch	训练过程中模型遍历整个数据集的次数。不同量级数据集的建议值：百量集4~8；千量集2~4；更大数量级1~2。	4
	迭代步数/Iterations	计算得出的模型参数/权重更新的次数。在调优过程中，每一个Iterations会消耗32条训练数据。	参见**表4-3**
	学习率/learning_rate	设置每个迭代步数（iteration）模型参数/权重更新的速率。学习率设置得过高会导致模型难以收敛，过低则会导致模型收敛速度过慢。	参见**表4-3**
	Checkpoint保存个数	训练过程中保存Checkpoint的个数。最小值为1，最大值为“迭代步数/Iterations”的参数值，不超过10。Checkpoint会自动存储到“调优后模型权重保存路径”的OBS路径下。	1
资源设置	资源池类型	资源池分为公共资源池与专属资源池。 – 公共资源池供所有租户共享使用。 – 专属资源池需单独创建，不与其他租户共享。	公共资源池

参数		说明	取值样例
	规格	选择规格，规格中描述了服务器类型、型号等信息，仅显示模型支持的资源。	xxx
	计算节点个数	当计算节点个数大于1，将启动多节点分布式训练。详细信息，请参见**分布式训练功能介绍**。	1
更多选项	永久保存日志	选择是否打开“永久保存日志”开关。	关闭
	事件通知	选择是否打开“事件通知”开关。	关闭
	自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。	关闭
	自动重启	选择是否打开“自动重启”开关。	关闭

表 4-3 多个调优任务的参数配置

序号	任务名称	选择调优类型	调优后的模型名称	迭代步数/Iterations	学习率/learning_rate
1	job-lora-01	**LoRA微调**	Qwen2-7B_01-lora	160	3.00E-05
2	job-lora-02	**LoRA微调**	Qwen2-7B_02-lora	480	3.00E-05
3	job-lora-03	**LoRA微调**	Qwen2-7B_03-lora	800	3.00E-04
4	job-sft-01	**全参微调**	Qwen2-7B_01-sft	160	3.00E-05
5	job-sft-02	**全参微调**	Qwen2-7B_02-sft	800	3.00E-05
6	job-sft-03	**全参微调**	Qwen2-7B_03-sft	160	3.00E-06

3. 参数配置完成后，单击“提交”。

   “资源池类型”选择“公共资源池”时，会出现“计费提醒”对话框，请您仔细阅读预计调优运行时间和预计消耗费用信息，然后单击“确定”，创建调优作业。该预估费用不包含OBS存储费用。预估费用基于目录价和预估时长计算，估算存在波动性，最终以实际发生为准。

   在“模型调优”列表中，当模型调优作业的“状态”变成“已完成”时，表示模型调优完成。

步骤5 部署模型服务。将原始模型“Qwen2-7B_template”和调优后获得的6种模型都部署成模型服务。

1. 在ModelArts Studio左侧导航栏中，选择“模型部署”。
2. 在“模型部署”页面，单击“我的服务”页签，在右上角单击“部署模型服务”进入部署页面，完成创建配置。

图 4-4 资源设置

表 4-4 部署模型服务

参数		说明	取值样例
服务设置	服务名称	自定义模型服务的名称。	参见表4-5
	描述	自定义部署模型服务的简介。	-
模型设置	部署模型	单击“选择模型”，从“我的模型”列表中选择需要部署的模型。	参见表4-5
资源设置	资源池类型	资源池分为公共资源池与专属资源池。 – 公共资源池供所有租户共享使用。 – 专属资源池需单独创建，不与其他租户共享。	公共资源池
	实例规格	选择实例规格，规格中描述了服务器类型、型号等信息。	xxx
	流量限制（QPS）	设置待部署模型的流量限制QPS。	3
	实例数	设置服务器个数。 推荐实例数 = 流量限制 ÷ 推荐的单实例流量限制	1
更多选项	内容审核	选择是否打开内容审核，默认启用。启用此能力可阻止模型推理中有害内容的输入输出，但可能会对接口性能产生较大影响。	打开

参数		说明	取值样例
	事件通知	选择是否打开“事件通知”开关。	关闭
	自动停止	当使用付费资源时，可以选择是否打开“自动停止”开关。	关闭

表 4-5 多个部署服务的参数配置

序号	服务名称	部署模型
1	service--lora-01	Qwen2-7B_01-lora
2	service--lora-02	Qwen2-7B_02-lora
3	service--lora-03	Qwen2-7B_03-lora
4	service--sft-01	Qwen2-7B_01-sft
5	service--sft-02	Qwen2-7B_02-sft
6	service--sft-03	Qwen2-7B_03-sft
7	service-00	Qwen2-7B_template

3. 参数配置完成后，单击“提交”。

   “资源池类型”选择“公共资源池”时，会出现“计费提醒”对话框，请您仔细阅读预估费用信息，然后单击“确定”，创建部署任务。模型部署会基于资源占用时长进行计费。服务状态为运行中时会产生费用，最终实际费用以账单为准。

   在“我的服务”列表中，当模型部署服务的“状态”变成“运行中”时，表示模型部署完成。

步骤6 使用模型服务：在MaaS体验部署完成的7个模型服务，测试推理结果。

1. 在ModelArts Studio左侧导航栏中，选择“模型体验”进入体验页面。
2. 在“模型体验”页面，单击“请选择服务”，在模型列表中选择模型服务，单击“确定”。
3. 在“模型体验”页面右上角，单击“参数设置”，按需拖动或直接输入数值配置推理参数。单击“恢复默认”可以将参数值调回默认值。

图 4-5 设置推理参数

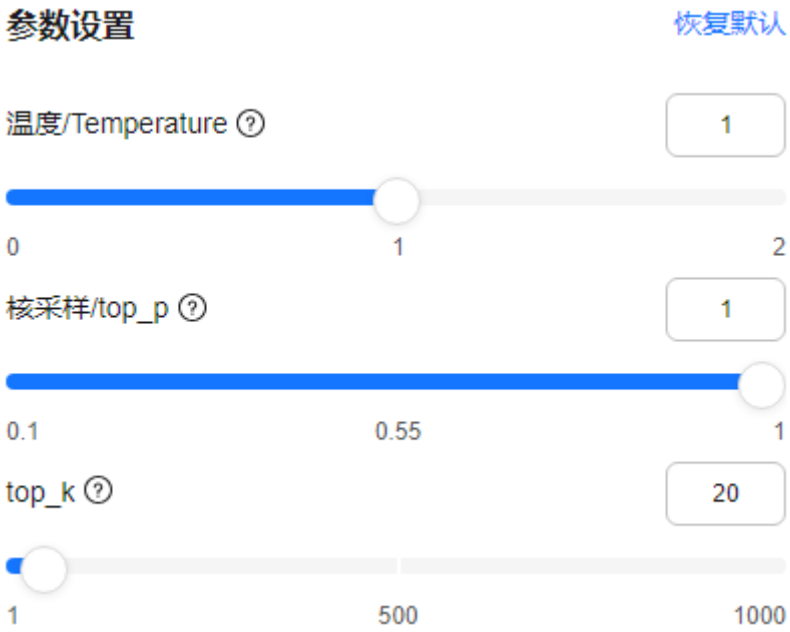


表 4-6 参数设置

参数	说明	取值样例
温度/Temperature	设置推理温度。 – 数值较高，输出结果更加随机。 – 数值较低，输出结果更加集中和确定。	1
核采样/top_p	设置推理核采样。调整输出文本的多样性，数值越大，生成文本的多样性就越高。	1
top_k	选择在模型的输出结果中选择概率最高的前K个结果。	20

4. 在对话框中输入问题，查看返回结果。

   输入的问题需要添加如下prompt，其中“{title}”要换成待判断的新闻标题，“{key_word}”要换成待判断的新闻关键词。

```
你是一位资深的新闻从业者，现在需要依据新闻标题和关键词对新闻进行分类。接下来你会收到<标题>和<关键词>，给出的分类结果需要严格按照以下要求：
1.使用json格式输出{"分类1":"中文标签","分类2":"英文标签"}
2.中文标签共15个，可选标签如下：民生、文化、娱乐、体育、财经、房产、汽车、教育、科技、旅游、国际、证券、农业、电竞等
3.英文标签共15个，可选标签如下：news_story、news_culture、news_entertainment、news_sports、news_finance、news_house、news_car、news_edu、news_tech、news_military、news_travel、news_world、stock、news_agriculture、news_game

示例：
<标题>：布达拉宫夜景，有缘人才能看见！太美了！
<关键词>：夜景,布达拉宫
<输出>：{"分类1":"旅游","分类2":"news_travel"}

现在请你判断以下新闻：
```

```
<标题>：{title}
<关键词>：{key_word}
<输出>：
```

说明

prompt需要满足如下要求：

- 设定合适的身份。
- 需求描述清晰：完整的上下文、任务内容要干什么和不要干什么、合理的符合标识。
- 指定输出结构。
- 给出一个示例。

返回结果如图4-6所示。

图 4-6 推理结果

步骤7 结果分析：分析模型的调优结果和推理结果。

推理结果的评分标准：

- 输出结构满足{"分类1":"xxx","分类2":"xxx"}，则格式正确，得1分。
- "分类1"和"分类2"的内容与标签相同，得1分。
- 从模型调优任务的日志中获取最后一个迭代的loss值，作为终止loss。

通过计算测试集161条数据的平均得分，作为最终得分。

表 4-7 模型服务的分析结果

序号	服务名称	选择调优类型	迭代步数/Iterations	学习率/learning_rate	格式正确得分	终止loss	最终得分
1	service--lora-01	LoRA微调	160	3.00E-05	161	0.0235	1.9005
2	service--lora-02	LoRA微调	480	3.00E-05	161	0.0338	1.907
3	service--lora-03	LoRA微调	800	3.00E-04	161	3.20E-05	1.9005
4	service--sft-01	全参微调	160	3.00E-05	0	0.0255	0
5	service--sft-02	全参微调	800	3.00E-05	1	1.70E-04	0.0124
6	service--sft-03	全参微调	160	3.00E-06	161	0.0235	1.8945
7	service-00	-	-	-	50	-	1.02

结果分析：

- 新闻自动分类是一个简单的分类任务，5281条数据的数据量也较小，因此LoRA微调和全参微调都能取得较好的调优结果。
- LoRA微调由于可调参数少，所以学习率设置比全参微调大。
- 全参微调用相同的“3.00E-05”学习率就出现了过拟合的现象。

由结果可知第1个和第3个模型服务的训推效果较好。

----结束

4.2 主流开源大模型基于 Lite Server 适配 Ascend-vLLM PyTorch NPU 推理指导（6.3.912）

4.2.1 Ascend-vLLM 介绍

Ascend-vLLM 概述

vLLM是GPU平台上广受欢迎的大模型推理框架，因其高效的continuous batching和pageAttention功能而备受青睐。此外，vLLM还具备投机推理和自动前缀缓存等关键功能，使其在学术界和工业界都得到了广泛应用。

Ascend-vLLM是华为云针对NPU优化的推理框架，继承了vLLM的优点，并通过特定优化实现了更高的性能和易用性。它使得在NPU卡上运行大模型变得更加高效和便捷，为用户带来了极大的便利和性能提升。Ascend-vLLM可广泛应用于各种大模型推

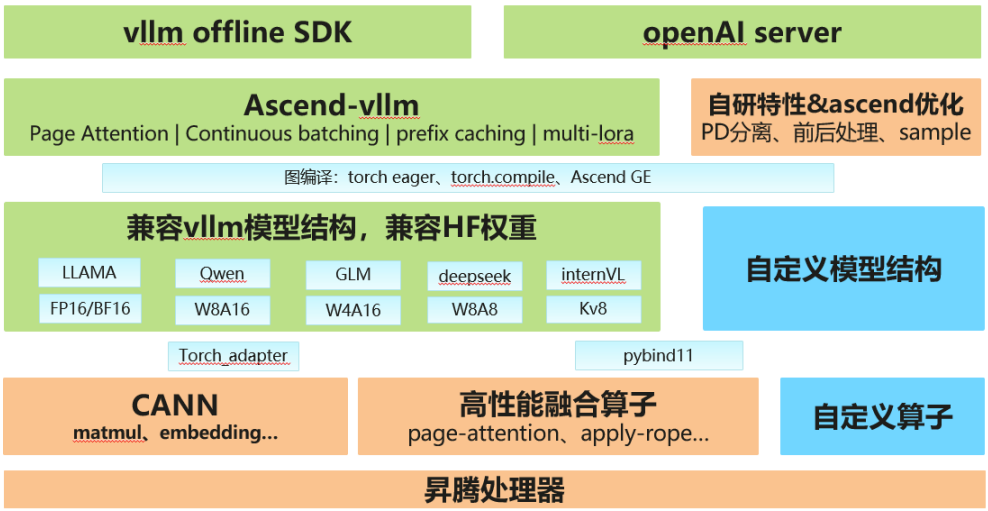
理任务，特别是在需要高性能和高效率的场景中，如自然语言处理、图像生成和语音识别等。

Ascend-vLLM的主要特点

1. **易用性**：Ascend-vLLM简化了在大模型上的部署和推理过程，使开发者可以更轻松地使用它。
2. **易开发性**：提供了友好的开发和调试环境，便于模型的调整和优化。
3. **高性能**：通过自研特性和针对NPU的优化，如PD分离、前后处理、sample等，实现了高效的推理性能。

Ascend-vLLM 架构

Ascend-vLLM架构图如下所示。



- 算子：使用CANN基础算子和高性能融合算子，同时支持用户自定义算子，持续迭代优化，提高推理效率。
- 模型：结构实现和社区一致，Huggingface模型开箱即用，同时可以快速适配新模型。
- 调用：提供高性能算子下发和图模式两种方案，兼顾性能和灵活性。
- 特性：服务调度、特性实现和社区一致，针对昇腾硬件做亲和替换和优化。
- 接口：离线SDK、在线OpenAI Server和社区完全一致，无缝迁移。

Ascend-vLLM 支持的特性介绍

表 4-8 Ascend-vLLM 支持的特性

特性名称		特性说明
调度	Page-attention	分块管理kvcache，提升吞吐。
	Continuous batching	迭代级调度，动态调整batch，降低延迟，提升吞吐。

特性名称		特性说明
	Multi-step	一次调度多次推理，降低调度上的cpu-overhead。
量化	W4A16-AWQ、GPTQ	权重Int4量化，降低显存消耗和时延。小并发时延提升80%，精度损失2%以内。
	W8A8-smoothQuant	权重Int8量化，降低显存消耗，吞吐提升30%；精度损失1.5%以内。
	W8A16-GPTQ	Int8量化，降低显存消耗，提高吞吐20%。精度损失1%以内。
	Kv8	Kv-cache量化，提高吞吐，支持更长序列。
高效解码	Auto-prefix-caching	前缀缓存，降低首token时延。在system prompt较长或多轮对话场景收益明显
	Chunked-prefill	又名split-fuse。全量增量同时推理，提高资源利用率，提升吞吐。
	Speculative Decoding	支持大小模型投机推理和eager模式投机，提升推理性能。
图模式	Cuda-graph/cann-graph	记录算子执行的依赖关系构图；消除python host耗时；且支持动态shape。
	Torch.compile	Torch.dynamo构图，转ascend-GE后端推理；使用静态分档。
实例复用	Multi-lora	多lora挂载，多个不同微调模型共用一份权重同时部署。
控制输出	Guided Decoding	通过特定模式控制模型输出。
	Beam search	通过beamsearch输出多个候选结果。
分离部署	PD分离部署	全量、增量分离部署，提高资源利用率，提升体验。
剪枝	FASP (Fast and Accurate Structured Pruning) 剪枝	FASP剪枝是一种结构化稀疏剪枝方法，能有效降低模型显存以及需要部署的资源依赖，减小推理过程中的计算量，降低增量推理时延，提升吞吐。

4.2.2 支持的模型列表

表 4-9 支持的大语言模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
53	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite
54	qwen-vl	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL
55	qwen-vl-chat	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL-Chat

56	MiniCPM-v2	√	x	x	x	x	https://huggingface.co/HwwwH/MiniCPM-V-2
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() # 新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

57	gte-Qwen2-7B-instruct	√	x	x	x	x	Alibaba-NLP/gte-Qwen2-7B-instruct at main (huggingface.co)

表 4-10 支持的多模态模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	internvl2-8B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-8B/tree/main
2	internvl2-26B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-26B/tree/main
3	internvl2-40B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-40B/tree/main
4	internVL2-Llama3-76B	√	√	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B/tree/main https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B-AWQ
5	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
6	qwen2-vl-2B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
7	qwen2-vl-7B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
8	qwen2-vl-72B	√	√	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct/tree/main https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct-AWQ

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
9	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
10	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
11	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
12	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
13	llava-v1.6-34b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-34b-hf/tree/main
14	llava-onevision-qwen2-0.5b-ov-hf	√	x	x	x	x	https://huggingface.co/llava-hf/llava-onevision-qwen2-0.5b-ov-hf
15	llava-onevision-qwen2-7b-ov-hf	√	x	x	x	x	https://huggingface.co/llava-hf/llava-onevision-qwen2-7b-ov-hf

说明

各模型支持的卡数请参见**各模型支持的最小卡数和最大序列**章节。

4.2.3 版本说明和要求

资源规格要求

本文档中的模型运行环境是ModelArts Lite的弹性节点Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

Ascend-vLLM 版本

本方案支持vLLM的v0.6.3版本。

镜像版本

本方案中用到的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-11 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-12**所示。

表 4-12 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

软件包结构说明

本教程需要使用到的AscendCloud-6.3.912中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
```

```
├──ascend_vllm
    ├── vllm_npu       # 推理源码
    ├── ascend_vllm-0.6.3-py3-none-any.whl   # 推理安装包
    ├── build.sh                # 推理构建脚本
    ├── vllm_install.patch       # 社区昇腾适配的补丁包
    ├── Dockerfile              # 推理构建镜像dockerfile
    ├── build_image.sh          # 推理构建镜像启动脚本
├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
    ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
    ├── autosmoothquant_ascend          # 量化代码
    ├── build.sh                        # 安装量化模块的脚本
  ├──AutoAWQ            # W4A16量化工具
    ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──quantize.py   # 昇腾适配的量化转换脚本
    ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
    ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
    ├── benchmark_parallel.py   # 评测静态性能脚本
    ├── benchmark_serving.py    # 评测动态性能脚本
    ├── benchmark_utils.py      # 抽离的工具集
    ├── generate_datasets.py    # 生成自定义数据集的脚本
    ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
    ├──opencompass.sh       #运行opencompass脚本
    ├──install.sh           #安装opencompass脚本
    ├──vllm_api.py          #启动vllm api服务器
    ├──vllm.py              #构造vllm评测配置脚本名字
```

4.2.4 推理服务部署

4.2.4.1 准备推理环境

前提条件

- 已准备Lite Server资源，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一：检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-11**。

```
docker pull {image_url}
```

步骤三：上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.912-xxx.zip和算子包AscendCloud-OPP-6.3.912-xxx.zip到主机中，包获取路径请参见**表4-12**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**支持的模型列表**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**大模型训练相关文档**。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下。

   ```
   df -h
   ```

步骤四：制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.912-xxx.zip和算子包AscendCloud-OPP-6.3.912-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && cd ./AscendCloud && unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-2.1.0-py39-*.zip -d ./AscendCloud-OPP && cd .. && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

多模态场景下，如果推理需要使用NPU加速图片预处理（仅适配了llava-1.5模型），启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

步骤五：启动容器

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

步骤六：进入容器

1. 进入容器。
   ```
   docker exec -it -u ma-user ${container-name} /bin/bash
   ```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
   ```
   npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```
   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

可以通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-7 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 92.5        51                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+====================================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

启动推理服务的具体操作步骤请参见**启动推理服务**。

4.2.4.2 启动推理服务

本章节主要介绍大语言模型的推理服务启动方式，包括离线推理和在线推理2种方式。

离线推理

编辑一个python脚本，脚本内容如下，运行该脚本使用ascend-vllm进行模型离线推理。

```
from vllm import LLM, SamplingParams

def main():
    prompts = [
        "Hello, my name is",
        "The president of the United States is",
        "The capital of France is",
        "The future of AI is",
    ]
    sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

    model_path = "/path/to/model"
    llm = LLM(model=model_path, tensor_parallel_size=1, max_model_len=8192)

    outputs = llm.generate(prompts, sampling_params)

    # Print the outputs.
    for output in outputs:
        prompt = output.prompt
        generated_text = output.outputs[0].text
        print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")

if __name__=="__main__":
    main()
```

启动在线推理服务

此处提供OpenAI服务API接口启动方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

推荐通过OpenAI服务的API接口启动推理，单机单卡和单机多卡场景下的具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code \
--enforce-eager
```

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**表4-25**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。模型并行与流水线并行的乘积取值需要和启动的NPU卡数保持一致，可以参考**表4-25**。此处举例为1，表示使用单卡启动服务。
- --pipeline-parallel-size：流水线并行数。模型并行与流水线并行的乘积取值需要和启动的NPU卡数保持一致，默认为1。
- --block-size：kv-cache的block大小，推荐设置为128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于multi-step调度。每次调度生成多个token，可以降低时延。开启投机推理后无需配置该参数，否则会导致投机推理启动报错。
- --multi-step-stream-outputs: 设置false后，multi-step会关闭流式输出提升性能，一次将返回num-scheduler-steps个token。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。
- --enforce-eager：未设置INFER_MODE环境变量时，部分模型会默认使用CANNGraph图模式启动来提升性能，设置该参数后将关闭图模式。CANNGraph图模式目前支持llama和qwen2系列大语言模型单卡场景，包含该系列AWQ量化模型，其他场景(如Multi-lora)暂未支持。小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
- --disable-async-output-proc：关闭异步后处理特性，关闭后性能会下降。

多机部署启动推理服务（可选）

当单机显存无法放下模型权重时，可选用多机方式部署；多机部署方式，需要机器在同一个集群，NPU卡之间IP能够ping通方可，具体步骤如下：

1. 查看卡IP，在其中一个宿主机上执行。
```
for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
```
2. 检查卡之间的网络是否通。
```
# 在另一个机器上执行，29.81.3.172是上一步输出的ipaddr的值
hccn_tool -i 0 -ping -g address 29.81.3.172
```
3. 在每个节点容器内，启动Ray集群。
```
# 指定通信网卡，使用ifconfig查看，找到和主机IP一致的网卡名
export GLOO_SOCKET_IFNAME=enp67s0f5
export TP_SOCKET_IFNAME=enp67s0f5
export RAY_EXPERIMENTAL_NOSET_ASCEND_RT_VISIBLE_DEVICES=1
# 指定可使用的卡
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
# 将其中一个机器设为头节点
ray start --head --num-gpus=8
# 在其他机器执行
ray start --address='10.170.22.18:6379' --num-gpus=8
```
   - --num-gpus：要跟ASCEND_RT_VISIBLE_DEVICES指定的可用卡数一致。
   - --address：头节点IP+端口号，头节点创建成功后，会有打印。

   说明
   - 环境变量每个节点都要设置。
   - 更新环境变量需要重启Ray集群。
4. 选择其中一个节点，添加指定分布式后端参数【--distributed-executor-backend=ray】，其他参数与正常启服务一致即可。具体参考本文单机场景下OpenAI服务的API接口**启动在线推理服务**方式。

推理请求测试

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**启动在线推理服务**。

通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

OpenAI Completions API with vLLM
```
curl http://${docker_ip}:8000/v1/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
```



```
    "prompt": "hello",
    "max_tokens": 7,
    "temperature": 0
}'
```

OpenAI Chat Completions API with vLLM

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
   "stream": false
}'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

说明

e5-mistral-7B和gte-Qwen2-7B-instruct模型，使用OpenAI启动服务，发送推理请求使用的接口如下。

```
curl -X POST http://${docker_ip}:8080/v1/embedding
```

OpenAI服务相关请求参数说明请参照**表4-13**。

表 4-13 OpenAI 服务请求参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0，1] 范围内。设置为1表示考虑所有tokens。

参数	是否必选	默认值	参数类型	描述
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0，2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0，2.0]。

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数，详细配置参照**guided-decoding**。

4.2.5 推理关键特性使用

4.2.5.1 量化

4.2.5.1.1 W4A16 量化

大模型推理中，模型权重数据类型（weight），推理计算时的数据类型（activation）和kvcache一般使用半精度浮点FP16或BF16。量化指将高比特的浮点转换为更低比特的数据类型的过程。例如int4、int8等。

模型量化分为weight-only量化，weight-activation量化和kvcache量化。

量化的一般步骤是：1、对浮点类型的权重镜像量化并保存量化完的权重；2、使用量化完的权重进行推理部署。

什么是 W4A16 量化

W4A16量化方案能显著降低模型显存以及需要部署的卡数（约75%）。大幅降低小batch下的增量推理时延。

约束限制

支持AWQ W4A16、 per-group（group-size=128）和perchannel量化。

支持AWQ量化的模型列表请参见**表4-9**。

步骤一：模型量化

可以在Huggingface开源社区获取量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：**从开源社区下载发布的AWQ量化模型**。

方式二：使用AutoAWQ量化工具进行量化。

AutoAWQ量化工具的适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/AutoAWQ目录下。

1、使用该量化工具，需要切换conda环境，运行以下命令。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

步骤二：权重格式离线转换（可选）

在GPU上AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

步骤三：启动量化服务

参考**启动在线推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.2.5.1.2 W8A8 量化

什么是 W8A8 量化

W8A8量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。

约束限制

- 支持SmoothQuant（W8A8）量化的模型列表请参见**支持的模型列表**。
- 激活量化支持动态per-token和静态per-tensor，支持非对称量化。
- 权重量化支持per-channel，支持非对称量化。
- Deepseek-v2系列模型的W8A8量化需要使用llm-compressor工具。

SmoothQuant 量化模型

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

1. SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

   代码目录如下:

   ```
   AutoSmoothQuant  #量化工具
           ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
           ├── autosmoothquant                 # 量化代码
           ├── build.sh                        # 安装量化模块的脚本
           ...
   ```

2. 执行如下命令进行权重转换。

   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
   - --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
   - --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 参考**启动推理服务**，启动推理服务时添加如下命令。
   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

使用 llm-compressor 工具量化 Deepseek-v2 系列模型

本章节介绍如何在GPU的机器上使用开源量化工具**llm-compressor**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   git clone https://github.com/vllm-project/llm-compressor.git
   cd llm-compressor
   pip install -e .
   ```
2. 修改examples/quantizing_moe/deepseek_moe_w8a8_int8.py中的代码：

   1）如果本地已有权重，请将MODEL_ID修改为权重路径；
   ```
   MODEL_ID = "deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct"
   ```
   2）如果量化Deepseek-V2-236B模型，请将num_gpus改为8；
   ```
   device_map = calculate_offload_device_map(
       MODEL_ID,
       reserve_for_hessians=True,
       num_gpus=8,
       torch_dtype=torch.bfloat16,
       trust_remote_code=True,
   )
   ```
   3）为减少量化时间，建议将以下参数设置为512；
   ```
   NUM_CALIBRATION_SAMPLES = 512
   ```
3. 执行权重量化：
   ```
   python deepseek_moe_w8a8_int8.py
   ```

说明

1、执行权重量化过程中，请保证使用的GPU卡上没有其他进程，否则可能出现OOM；

2、如果量化Deepseek-v2-236b模型，大致需要10+小时。

4.2.5.1.3 W8A16 量化

什么是 W8A16 量化

使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

约束限制

- 只支持GPTQ W8A16 perchannel量化，只支持desc_act=false。
- GPTQ W8A16量化支持的模型请参见**支持的模型列表**。

步骤一：量化模型权重

在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。

```
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
model_id = "meta-llama/CodeLlama-34b-hf"
tokenizer = AutoTokenizer.from_pretrained(model_id)
gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。

```
dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。

```
quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
quantization_config=gptq_config)
```

5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。

```
quantized_model.save_pretrained("CodeLlama-34b-hf")
tokenizer.save_pretrained("CodeLlama-34b-hf")

# if quantized with device_map set
quantized_model.to("cpu")
quantized_model.save_pretrained("CodeLlama-34b-hf")
```

步骤二：启动量化服务

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**启动在线推理服务**。

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

4.2.5.1.4 kv-cache-int8 量化

什么是 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。

约束限制

- 当前支持per-token动态量化（推荐）， per-tensor静态量化以及per-tensor+per-head静态量化。
- 支持kv-cache-int8量化和FP16、BF16、AWQ、SmoothQuant的组合。
- kv-cache-int8量化支持的模型请参见**支持的模型列表**。

per-token 动态量化场景

使用该场景量化方法，无需提前生成量化权重。推理前向会自动计算kv-cache量化系数，并进行kv的量化。

在**启动推理服务时**添加如下参数，启动kv-cache-int8-per-token量化服务。

```
--kv-cache-dtype int8_pertoken  #只支持int8，表示kvint8 per-token量化
```

per-tensor 静态量化场景

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数。

1. 使用tensorRT量化工具进行模型量化。

   在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

   量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

   执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

   ```
   python convert_checkpoint.py \
   --model_dir ./llama-models/llama-7b-hf   \
   --output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
   --dtype float16  \
   --int8_kv_cache
   ```

   运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

2. 抽取kv-cache量化系数。

   该步骤的目的是将上一步中生成的scale系数提取到单独文件中，供推理时使用。

   使用的抽取脚本由vllm社区提供：

   ```
   python3 examples/fp8/extract_scales.py \
   --quantized_model <QUANTIZED_MODEL_DIR> \
   --tp_size <TENSOR_PARALLEL_SIZE> \
   --output_dir <PATH_TO_OUTPUT_DIR>
   ```

   运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

a. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。

b. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

3. 启动kv-cache-int8-per-tensor量化服务。

   在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

   ```
   --kv-cache-dtype int8_pertensor  #只支持int8，表示kvint8 per-tensor量化
   --quantization-param-path kv_cache_scales.json  #输入2. 抽取kv-cache量化系数生成的json文件路径;如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
   ```

per-tensor+per-head 静态量化场景

如需使用该场景量化方法，请自行准备kv-cache量化系数，格式和per-tensor静态量化所需的**2. 抽取kv-cache量化系数**生成的json文件一致，只需把每一层的量化系数修改为列表，列表的长度为kv的头数，列表中每一个值代表每一个kv头使用的量化系数。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": [0.09965550899505615, 0.20214074850082397, 0.16
                "1": [0.07757135480642319, 0.15182086825370789, 0.13
```


在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-tensor+per-head量化服务。


```
--kv-cache-dtype int8_pertensor_perhead  #只支持int8，表示kvint8 per-tensor+per-head量化
--quantization-param-path kv_cache_scales.json  #输入生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.2.5.2 剪枝

什么是剪枝

剪枝是一种大模型压缩技术的关键技术，旨在保持推理精度的基础上，减少模型的复杂度和计算需求，以便大模型推理加速。

剪枝的一般步骤是：1、对原始模型调用不同算法进行剪枝，并保存剪枝后的模型；2、使用剪枝后的模型进行推理部署。

常用的剪枝技术包括：结构化稀疏剪枝、半结构化稀疏剪枝、非结构化稀疏剪枝。

FASP 剪枝

FASP剪枝是一种结构化稀疏剪枝方法，能有效降低模型显存以及需要部署的资源依赖，减小推理过程中的计算量，降低增量推理时延，提升吞吐。

FASP (Fast and Accurate Structured Pruning) 一种针对LLM进行结构化剪枝的算法，可以减少大模型对于内存和计算资源的需求，提升推理速度，同时其具备比较高的剪枝速度。使用FASP对大模型进行稀疏化剪枝，可以在几乎不影响推理精度情况下，可以有效提升推理性能（吞吐等）。

本文主要应用FASP对LLM进行剪枝压缩。

约束限制

FASP剪枝当前仅支持Llama系列、Llama2系列、Llama3系列、Qwen2系列的NLP模型。

安装 AscendModelNano

AscendModelNano是FASP剪枝工具，适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/ModelNano目录下。

AscendModelNano工具需要安装，执行命令如下。

```
cd ModelNano  # 进入ModelNano工具目录
bash build.sh AscendModelNano  # 编译
pip install dist/AscendModelNano-0.1.0-py3-none-any.whl  #安装
```

模型剪枝

可以在Huggingface开源社区获取需剪枝的模型权重或者获得业务上已预训练好的模型权重，通过AscendModelNano工具进行FASP剪枝。

```
CUDA_VISIBLE_DEVICES=0 python3 -m model_nano.prune_llama \
 --model $MODEL_FILE \
 --dataset $DATASET \
 --nsamples $SAMPLE \
 --sparsity $SPARSITY \
 --save $SAVE_DIR \
 --seed 0 \
 --memory_efficient \
 --eval
```

参数介绍：

- model：必选，要进行压缩的原始模型地址
- save：必选，压缩后模型的保存的地址
- dataset：可选，压缩模型所用的校准数据，可选范围["wikitext2"，"c4"]，默认wikitext2。
- nsamples：可选，压缩模型所用的校准数据样本数量，默认128。
- seed：可选，随机数种子。
- sparsity：可选，剪枝稀疏度，稀疏度越大剪枝压缩率越高，默认0.1。
- memory_efficient：可选，优化剪枝过程中的显存使用，推荐传入。
- eval：可选，是否进行压缩后模型的PPL评估。如果输入此参数，在wikitext2以及c4数据上进行PPL计算。

具体的代码示例如下。

```
GPU=0
SPARSITY=0.1
MODEL="llama-3-8b"
SAMPLE=128
DATASET="wikitext2"
MODEL_FILE="/mnt/models/$MODEL" SAVE_DIR="/mnt/save_models/${MODEL}_${SPARSITY}_$
{DATASET}_n${SAMPLE}"
CUDA_VISIBLE_DEVICES=$GPU
python3 -m model_nano.prune_llama \
  --model $MODEL_FILE \
  --dataset $DATASET \
  --nsamples $SAMPLE \
  --sparsity $SPARSITY \
  --save $SAVE_DIR \
  --seed 0 \
  --memory_efficient \
  --eval
```

启动剪枝模型推理服务

使用剪枝特性时，启动推理服务时的model_path请使用剪枝处理后的模型。具体参考**启动推理服务**。

4.2.5.3 分离部署

4.2.5.3.1 PD 分离部署使用说明

什么是 PD 分离部署

大模型推理是自回归的过程，有以下两阶段：

- Prefill阶段（全量推理）

  将用户请求的prompt传入大模型，进行计算，中间结果写入KVCache并推出第1个token，属于计算密集型。

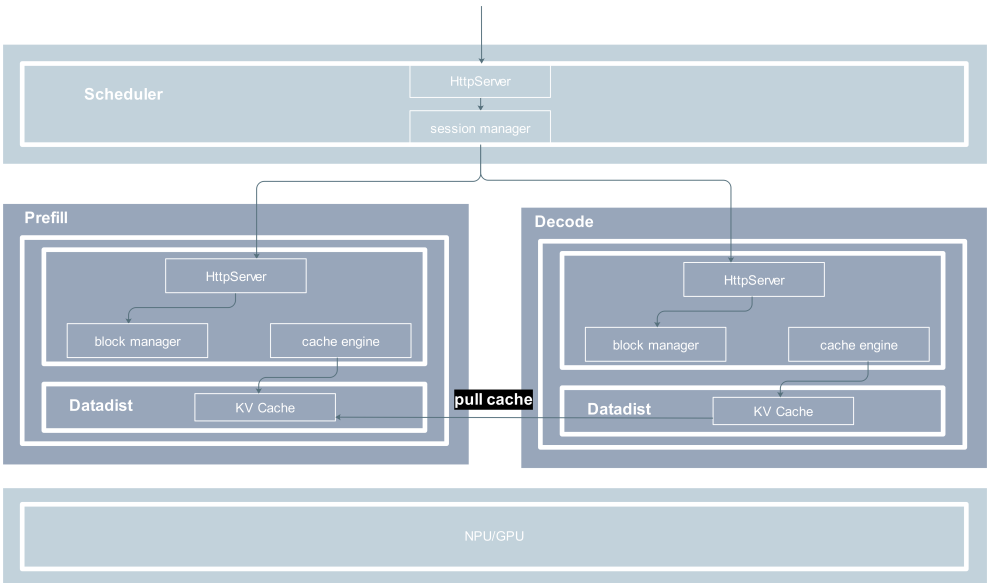
- Decode阶段（增量推理）

  将请求的前1个token传入大模型，从显存读取前文产生的KVCache再进行计算，属于访存密集型。

PD分离部署场景下，大模型推理的Prefill阶段（全量推理）和Decode阶段（全量推理）分别实例化部署在不同的推理卡资源上同时进行推理，用于提高资源利用效率。

PD分离结合Prefill阶段的计算密集型特性，以及Decode阶段的访存密集型特性，通过调节PD节点数量配比来提升Decode节点的batch size来充分发挥NPU卡的算力，进而提升集群整体吞吐。

此外，在Decode平均低时延约束场景，PD分离相比PD混合部署，更加能够发挥性能优势。



分离部署的实例类型启动分为以下三个阶段：

1. **步骤二：启动全量推理实例**：必须为NPU实例，用于启动全量推理服务，负责输入的全量推理。全量推理占用至少1个容器。
2. **步骤三：启动增量推理实例**：必须为NPU实例，用于启动增量推理服务，负责输入的增量推理。增量推理占用至少1个容器。
3. **步骤四：启动scheduler实例**：可为CPU实例，用于启动api-server服务，负责接收推理请求，向全量或增量推理实例分发请求，收集推理结果并向客户端返回推理结果。服务调度实例不占用显卡资源，建议增加1个容器，也可以在全量推理或增量推理的容器上启动。

约束限制

- 全量和增量节点的local rank table必须一一对应。
- 全量和增量节点不能使用同一个端口。
- scheduler实例中NODE_PORTS=8088,8089；端口设置顺序必须与global rank table文件中各全量和增量节点顺序一致，否则会报错。
- 确保scheduler实例和P、D实例之间网络通畅，检查代理设置例如no_proxy环境变量，避免scheduler访问P、D实例时走不必要的网关。

前提条件

已完成推理环境镜像制作，具体参见**准备推理环境**。

步骤一：生成 ranktable

介绍如何生成ranktable，以1p1d-tp2分离部署模式为例。当前1p1d分离部署模式，全量节点和增量节点分别占用2张卡，一共使用4张卡。

1. 配置tools工具根目录环境变量

   使用AscendCloud-LLM发布版本进行推理，基于AscendCloud-LLM包的解压路径配置tool工具根目录环境变量：

   ```
   export LLM_TOOLS_PATH=${root_path_of_AscendCloud-LLM}/llm_tools
   ```

   其中，${root_path_of_AscendCloud-LLM}为AscendCloud-LLM包解压后的根路径。

   当使用昇腾云的官方指导文档制作推理镜像时，可直接基于该固定路径配置环境变量：

   ```
   export LLM_TOOLS_PATH=/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools
   ```

2. 获取每台机器的rank_table

   在每个机器生成global rank_table信息与local rank_table信息。

   ```
   python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 4,5 --decode-server-list 6,7 --api-server --save-dir ./save_dir
   ```

   执行后，会生成一个global_ranktable.json文件和使用实例个数的local_ranktable.json文件；如果指定了--api-server，还会生成一个local_ranktable_host.json文件用于确定服务入口实例。

   ./save_dir生成ranktable文件如下（假设本地主机ip为10.**.**.18）。

   ```
   global_ranktable_10.**.**.18.json       # global rank_table
   local_ranktable_10.**.**.18_45.json     # 全量节点local rank_table
   local_ranktable_10.**.**.18_67.json     # 增量节点local rank_table
   local_ranktable_10.**.**.18_host.json   # api-server
   ```

   如果要启动多P多D服务，则需要修改--prefill-server-list和--decode-server-list参数，每个实例之间用空格隔开，例如2p2d-tp2：

   ```
   python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 0,1 2,3 --decode-server-list 4,5 6,7 --api-server --save-dir ./save_dir
   ```

3. 合并不同机器的global rank_table(可选)

   如果分离部署在多台机器，获取每台机器的rank_table后，合并各个机器的global rank_table得到完整的global rank_table。

   ```
   python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode merge --global-ranktable-list ./ranktable/global_ranktable_0.0,0,0.json ./ranktable/global_ranktable_1.1.1.1.json --save-dir ./save_dir
   ```

- pd_ranktable_tools.py的入参说明如下。
  - --mode：脚本的处理模式，可选值为gen或者merge。gen模式表示生成rank_table文件，merge模式表示合并global rank_table文件。
  - --save-dir：保存生成的rank_table文件的根目录，默认为当前目录。
  - --api-server：仅在gen模式有效，可选输入，当存在该输入时，表示分离部署的服务入口在该机器。注意，在多台机器启动分离部署时，只能有一台机器存在服务入口。当存在该输入时，会生成local_ranktable_xx_host.json文件，用于在启动推理服务时确定服务入口实例。
  - --prefill-server-list：仅在gen模式有效，可选输入，后续入参表示若干个vllm全量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用英文逗号`,`分隔开。当存在该输入时，会生成对应全量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定全量实例。
  - --decode-server-list：仅在gen模式有效，可选输入，后续入参表示若干个vllm增量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用英文逗号`,`分隔开。当存在该输入时，会生成对应增量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定增量实例。

- --global-ranktable-list：仅在merge模式有效，必选输入，后续入参表示需要合并的global rank_table，使用空格分隔开。

合并不同机器的global rank_table后，会生成新合并的global_ranktable_merge.json文件。

- global_rank_table.json格式说明

server_group_list的长度必须为3，第一个元素(group_id="0")代表Scheduler实例的ip信息，只能有一个实例。

第二个元素(group_id="1")代表全量实例信息，长度即为全量实例个数。其中需要配置每个全量实例的ip信息以及使用的device信息。rank_id为逻辑卡号，必然从0开始计算，device_id为物理卡号，device_ip则通过上面的hccn_tool获取。

第三个元素(group_id="2")代表增量实例信息，长度即为增量实例个数。其余信息和全量类似。

global_rank_table.json具体示例如下：

```
{
   "version": "1.0",
   "status": "completed",
   "server_group_list": [
      {
         "group_id": "0",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost"
            }
         ]
      },
      {
         "group_id": "1",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "4",
                     "device_ip": "10.**.**.22",
                     "rank_id": "0"
                  },
                  {
                     "device_id": "5",
                     "device_ip": "10.**.**.23",
                     "rank_id": "1"
                  }
               ]
            }
         ]
      },
      {
         "group_id": "2",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "6",
                     "device_ip": "29.**.**.56",
                     "rank_id": "0"
                  },
                  {
```

```
                    "device_id": "7",
                    "device_ip": "29.**.**.72",
                    "rank_id": "1"
                }
            ]
        }
    ]
}
```

- local_rank_table.json格式说明

  每个全量/增量实例都需要local_rank_table.json。下面以某一个增量实例为例，需要和global_rank_table.json中的增量信息完全对应，group_id为0。

```
{
   "version": "1.0",
   "status": "completed",
   "group_id": "0",
   "server_count": "1",
   "server_list": [
      {
         "server_id": "localhost",
         "server_ip": "localhost",
         "device": [
            {
               "device_id": "6",
               "device_ip": "29.**.**.56",
               "rank_id": "0"
            },
            {
               "device_id": "7",
               "device_ip": "29.**.**.72",
               "rank_id": "1"
            }
         ]
      }
   ]
}
```

步骤二：启动全量推理实例

以下介绍如何启动全量推理实例。

1. 启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci4、davinci5。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明：

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
  - 如果需要多个全量实例，每个全量都需要启动一个容器，只挂载对应的NPU
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动全量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_45.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8088 \
   --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量或增量推理实例启动的--port参数相关。--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。

- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：OpenAI服务的model入参名称，仅在环境变量USE_OPENAI=1时生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**量化**章节对模型做量化处理。
- --prefill-batching-policy：针对PD分离场景下的P实例调度策略选择，支持fcfs（先来先服务，vllm默认）及gtsf（Group Time Short First，组内等待时间最短优先）两种策略，若负载比较高且请求长度差异大可以选择gtsf进行尝试。

参数定义和使用方式与vLLM0.6.3版本一致，此处介绍关键参数。详细参数解释请参见**https://github.com/vllm-project/vllm/blob/main/vllm/engine/arg_utils.py**。

步骤三：启动增量推理实例

1. 启动增量推理容器

   启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci6、davinci7。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，$

{container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- 如果需要多个增量实例，每个增量都需要启动一个容器，只挂载对应的NPU

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动增量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_67.json

export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8089 \
   --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用,（英文逗号）分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP地址
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号

- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：OpenAI服务的model入参名称，仅在环境变量USE_OPENAI=1时生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**量化**章节对模型做量化处理。

步骤四：启动 scheduler 实例

建议在PD服务（即全量推理和增量推理服务）启动后，再启动scheduler服务。

1. 启动scheduler容器。启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了0张卡。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动scheduler实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1
export no_proxy=localhost,127.0.0.1,10.**.**.18

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=9000 \
    --served-model-name ${served-model-name}

# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。当前端口9000是对外服务端口，而8088、8089则为scheduler调度推理服务端口。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。
- no_proxy：可选，避免scheduler实例和P、D实例之间访问时走不必要的网关。

其中常见的参数如下，

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号。分离部署对外服务使用的是scheduler实例端口，在后续推理性能测试和精度测试时，服务端口需要和scheduler实例端口保持一致。
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量USE_OPENAI=1时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**量化**章节对模型做量化处理。
- --prefill-routing-policy：全量节点路由策略，支持RoundRobin（轮询，默认）、FreeKVFirst（优先调度到空闲KV最多的节点）、BLB（优先调度到排队请求数量最少的节点）三种。

4.2.5.4 Prefix Caching

什么是 Prefix Caching

在LLM推理应用中，经常会面临具有长system prompt的场景以及多轮对话的场景。长system prompt的场景，system prompt在不同的请求中但是相同的，KV Cache的计算也是相同的；多轮对话场景中，每一轮对话需要依赖所有历史轮次对话的上下文，历史轮次中的KV Cache在后续每一轮中都要被重新计算。这两种情况下，如果能把system prompt和历史轮次中的KV Cache保存下来，留给后续的请求复用，将会极大地降低首Token的耗时。如果Prefix Cache和Generated KV Cache都可以缓存，在多轮对话的应用中，忽略边界情况，基本上可以认为其消除了历史轮次中生成对话的recompute。

Ascend vllm提供prefix caching关键特性能力，能够显著降低长system prompt和多轮对话场景首token时延，提升用户体验。其优势主要包括：

- **更短的prefill时间**：由于跨请求的重复token序列对应的KV Cache可以复用，那么就可以减少一部分前缀token的KV Cache计算时间，从而减少prefill的时间。
- **更高效的显存使用**：当正在处理的请求相互之间存在公共前缀时，公共前缀部分的KV Cache可以共用，不必重复占用多份显存。

约束限制

- 该特性不能和Chunked-prefill、KV Cache量化特性同时使用。
- 该特性暂不支持与LoRA特性配合。
- 多模态模型暂不支持prefix cache。
- LLaMa系列、Qwen系列模型支持此特性。
- 当跨请求公共前缀token数大于等于Page Attention中的block size，才会进行公共前缀token的KV Cache复用。

Prefix Caching 参数设置

启动推理服务时，使用Prefix Cache特性需要配置的补充参数如表4-14所示，对应的代码样例如表4-15所示。

表 4-14 Prefix Cache 特性参数

服务启动方式	配置项	取值类型	取值范围	配置说明
offline	enable_prefix_caching	bool	• True • False	• True：会开启Prefix Cache特性。 • False：不会开启Prefix Cache特性。

服务启动方式	配置项	取值类型	取值范围	配置说明
online	--enable-prefix-caching	-	-	• 设置：会开启Prefix Cache特性。 • 不设置：不会开启Prefix Cache特性。 须知： 启用Prefix Cache特性是在起服务时指定，属于action类型参数。

表 4-15 开启 Prefix Cache 特性服务的代码样例

服务启动方式	接口	服务启动基础命令
offline	-	`LLM(model="facebook/opt-125m", enable_prefix_caching=True)`

online	vllm	`python -m vllm.entrypoints.api_server \`
		`--model=facebook/opt-125m \`
		`--enable-prefix-caching`

online	openai	`python -m vllm.entrypoints.openai.api_server \`
		`--model=facebook/opt-125m \`
		`--enable-prefix-caching`

执行推理参考

1. 配置服务化参数。Ascend vllm使用该特性需参考**表4-14**和**表4-15**，其它参数请**启动推理服务**。
2. 启动服务。具体请参考**启动推理服务**。
3. 精度评测和性能评测。具体请参考**推理服务精度评测**和**推理服务性能评测**。

4.2.5.5 multi-step

什么是 multi-step

vLLM的调度和输入准备的CPU开销可能会导致NPU利用率不足，开启multi-step调度可以有效解决这个问题，开启multi-step调度后会在执行一次调度和输入准备后，连续n步运行模型。通过NPU在n步之间连续处理，而无需等待CPU，可以将CPU开销分散到n步中，从而显著减少NPU空闲时间，提升整体性能。

约束限制

暂不支持Multi-Lora和投机推理场景。

multi-step 参数设置

启动推理服务时，使用multi-step调度需要配置的参数如下表所示。

表 4-16 开启 multi-step 调度参数配置

服务启动方式	配置项	取值类型	配置说明
offline	num_scheduler_steps	int	连续运行模型的步数。 默认为1，推荐设置为8
offline	multi_step_stream_outputs	bool	设置false后，mult-step会关闭流式输出提升性能，一次将返回num_scheduler_steps个token。 默认true
online	--num-scheduler-steps	int	连续运行模型的步数。 默认为1，推荐设置为8
online	--multi-step-stream-outputs	bool	设置false后，mult-step会关闭流式输出提升性能，一次将返回--num-scheduler-steps个token。 默认true

4.2.5.6 投机推理

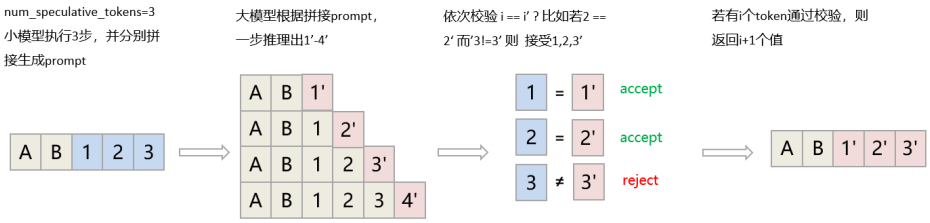
4.2.5.6.1 投机推理使用说明

什么是投机推理

传统LLM推理主要依赖于自回归式（auto-regressive）的解码（decoding）方式，每步解码只能够产生一个输出token，并且需要将历史输出内容拼接后重新作为LLM的输入，才能进行下一步的解码。为了解决上述问题，提出了一种投机式推理方式，其核心思想是通过计算代价远低于LLM的小模型替代LLM进行投机式地推理（Speculative Inference）。即每次先使用小模型试探性地推理多步，再将这些推理结果收集到一起，一次交由LLM进行验证。

如下图所示，在投机模式下，先由小模型依次推理出token 1、2、3，并将这3个token一次性输入大模型LLM推理，得到1'、2'、3'、4'，将1、2、3与1'、2'、3'依次校验，即可用三次小模型推理（相较于大模型，耗时极短），以及一次大模型推理的时间，得到1~4个token，大幅提升推理性能。

图 4-8 大小模型投机示意图



如此一来，投机推理可以带来如下优势：

更短的decode平均时间：以qwen2-72b作为LLM大模型、qwen2-0.5b作为小模型为例，小模型推理一次的时间不足大模型的1/5，加上校验后，执行一次完整投机流程的

时间也仅为大模型的1.5倍左右（投机步数设置为3步）。而这一次投机流程，平均可以生成3个有效token，即用1.5倍的时间代价，生成了3倍的token数量，性能提升了100%。

投机推理参数设置

在启动离线或在线推理服务时参考**表4-17**所示配置参数，使用投机推理功能。

表 4-17 投机推理相关参数

服务启动方式	配置项	取值类型	配置说明
offline	speculative_model	str	小模型权重地址，目前支持相对基础模型（如llama2-13b-chat）较小的LLM模型（如llama1.1b）或者基础模型对应的eagle模型（如EAGLE-llama2-chat-13B）。 Eagle小模型可以通过开源模型获取，也可以参考**Eagle投机小模型训练**训练获取。
offline	num_speculative_tokens	int	小模型投机步数，即小模型生成几个token来交给大模型进行推理，取值通常在2~6之间，步数越小，每次校验的token越少，投机token与大模型一致的概率越高；反之，步数过大会导致与大模型无法保持一致，造成资源浪费。
offline	speculative_draft_tensor_parallel_size	int	小模型所使用的设备数量，由于小模型通常较小，所以此处建议设置为1，如果使用eagle作为小模型，此处必须设置为1
offline	speculative_disable_by_batch_size	int	投机推理batch上限，即当输入batch大于此值，将不进行投机推理。其使用原因是投机推理在大batch下收益会显著下降，故需要在batch过大时关闭该特性。
online	--speculative-model	str	小模型权重地址，目前支持相对基础模型（如llama2-13b-chat）较小的LLM模型（如llama1.1b）或者基础模型对应的eagle模型（如EAGLE-llama2-chat-13B） Eagle小模型可以通过开源模型获取，也可以参考**Eagle投机小模型训练**训练获取。
online	--num-speculative-tokens	int	小模型投机步数，即小模型生成几个token来交给大模型进行推理，取值通常在2~6之间，步数越小，每次校验的token越少，投机token与大模型一致的概率越高；反之，步数过大会导致与大模型无法保持一致，造成资源浪费。
online	--speculative-draft-tensor-parallel-size	int	小模型所使用的设备数量，由于小模型通常较小，所以此处建议设置为1，如果使用eagle作为小模型，此处必须设置为1

服务启动方式	配置项	取值类型	配置说明
online	--speculative-disable-by-batch-size	int	投机推理batch上限，即当输入batch大于此值，将不进行投机推理。其使用原因是投机推理在大batch下收益会显著下降，故需要在batch过大时关闭该特性。

投机推理端到端推理示例

以**llama-2-13b-chat-hf**模型作为LLM大模型，**llama1.1b**作为小模型，启用openai接口服务为例。

1. 使用下面命令启动推理服务。

```
base_model=/path/to/base_model
spec_model=/path/to/spec_model
spec_step=3
spec_parallel=1
python -m vllm.entrypoints.openai.api_server \
--model=${base_model} \   # 大模型权重地址
--speculative-model=${spec_model} \   # 投机小模型权重地址
--num-speculative-tokens=${spec_step} \ #投机步数
--speculative-draft-tensor-parallel-size=${spec_parallel} \ #投机小模型使用的卡数，通常设置为1
--tensor-parallel-size=1 \
--host 0.0.0.0 \
--port 9999 \
--dtype auto \
--gpu-memory-utilization=0.9 \
--served-model-name test_server_demo \
--trust-remote-code
```

2. curl请求

```
curl http://0.0.0.0:9999/v1/chat/completions \    -H "Content-Type: application/json" \    -d '{        "model": "test_server_demo",        "messages": [            {"role": "system", "content": "You are a helpful assistant."},            {"role": "user", "content": "Who won the world series in 2020?"},            {"role":"assistant","content":"The World Series in 2020 was won by the Los Angeles Dodgers. They defeated the Tampa Bay Rays in a six-game series, which was delayed and played in October and November due to the COVID-19 pandemic."},            {"role": "user", "content": "Do you like NewYork?"}        ],    "max_tokens": 100,    "top_k": -1,    "top_p": 1,    "temperature": 0,    "ignore_eos": false,    "stream": false    }'
```

执行推理参考

1. 配置服务化参数。Ascend vllm使用该特性需参考**表4-17**，其它参数请参考**启动推理服务**。
2. 启动服务。具体请参考**启动推理服务**。
3. 精度评测和性能评测。具体请参考**推理服务精度评测**和**推理服务性能评测**。

4.2.5.6.2 Eagle 投机小模型训练

什么是 Eagle 投机小模型训练

2013年12月滑铁卢大学、加拿大向量研究院、北京大学等机构联合发布Eagle，旨在提升大语言模型的推理速度，同时保证模型输出文本的分布一致。这种方法外推LLM的第二顶层特征向量，能够显著提升生成效率。

Eagle训练了一个单层模型，使用input token和基模型推理出的hidden-state作为输入，输出hidden-state。然后根据这个输出的hidden-state使用基模型的原始LLM的分

类头来预测下一个词。hidden-state比input token包含更多信息，使得回归hidden-state的任务比预测词的任务简单得多。总之，Eagle在hidden-state层面上进行外推，使用一个小型单层Eagle模型，然后利用基模型的冻结的分类头生成预测的token。

如此一来，Eagle投机推理可以带来如下优势：

- **更小的训练成本得到小模型：**相较于训练独立的LLM大模型，Eagle仅需训练一个自回归层。这使得其训练成本相较于训练一个独立的LLM模型要小得多。
- **为每个模型提供针对性的投机模型：** Eagle的模型大小及结构，与基模型的某一层完全相同，这使得它的大小远远小于其基模型。解决了对于部分原始LLM模型，找不到合适的投机模型的问题。

投机小模型训练端到端示例

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据训练eagle小模型，并使用自行训练的小模型进行eagle推理。支持llama1系列、llama2系列和Qwen2系列模型。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装Eagle。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的如下格式，则需要执行convert_to_sharegpt.py文件将数据集转换为share gpt格式。

```
{
"prefix":  "AAA"
"input": "BBB",
"output": "CCC"
}
```

执行convert_to_sharegpt.py文件。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

参数解释如**表4-18**所示。

当转换为sharegpt格式时，prefix和input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

表 4-18 数据集转换为 sharegpt 格式阶段（可选）

py文件名称	配置项	取值类型	配置说明
convert_to_sharegpt.py	--input_file_path	str	预训练json文件地址。
	--out_file_name	int	输出的sharegpt格式文件地址。
	--prefix_name	str	预训练json文件的前缀字段名称，例如：您是一个xxx专家，您需要回答下面问题。prefix_name可设置为None，此时预训练数据集只有input和output两段输入。
	--input_name	str	预训练json文件的指令输入字段名称，例如：请问苹果是什么颜色。
	--output_name	str	预训练json文件的output字段名称，例如：苹果是红色的。
	--code_type	str	预训练json文件编码，默认utf-8。

步骤三：sharegpt 格式数据生成为训练 data 数据集

设置环境变量。

```
export EAGLE_TARIN_MODE=1
```

如果使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

如果使用其他数据集，需要先执行**步骤二：非sharegpt格式数据集转换（可选）**转换数据集格式为sharegpt格式。

执行如下脚本将sharegpt格式数据生成为训练data数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--npu_indices "0,1,2,3,4,5,6,7" \
--used_npus 8  \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048
--dtype bfloat16
```

其中具体参数解释如**表4-19**所示。

表 4-19 数据生成阶段

py文件名称	配置项	取值类型	配置说明
allocation.py	--outdir	str	生成的训练所需数据的输出地址
	--end_num	int	生成的训练数据总条数
	--npu_indices	str	使用哪些NPU进卡行数据生成，用逗号隔开，如"0,1,2,3,4,5,6,7"
	--used_npus	int	拉起的每个py脚本使用几个NPU，如果为70b则填写4或8，7b 13b则填1。
	--model_type	str	使用模型类型，目前支持llama系列（填写llama）及qwen2（填写qwen2）
	--model_name	str	LLM的基模型地址，如./Llama2-7b
	--data_path	str	预训练数据集地址，如sharegpt.json
	--seed	int	生成训练数据所使用的seed，默认为42，42为开源训练设定参数。
	--max_length	int	模型的最大长度，默认为2048
	--dtype	str	模型dtype，默认为bfloat16

执行完成后，记得unset环境变量，否则会导致后续推理服务启动出错。

```
unset EAGLE_TARIN_MODE
```

执行完成后，如果used_npus>1，则需要将训练生成data数据重新分配为8个文件夹，分配脚本为reassign_data_num.py。

```
python reassign_data_num.py --old_folder "./sharegpt_0_199_mufp16/" \
--new_folder "./sharegpt_0_199_mufp16/" \
--tp 8
```

- old_folder为上一步生成data的地址，填写到卡号的文件夹之前。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- new_folder为需要存储新的data的地址。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- tp为需要切分成的文件夹数量，默认为8。

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

具体的参数解释如表4-20所示，

表 4-20 执行训练阶段

py文件名称	配置项	取值类型	配置说明
eagle.train.main	--tmpdir	str	生成的训练所需数据的输出地址，即数据生成中的outdir
	--cpdir	str	训练出的Eagle模型权重存放地址
	--configpath	str	Eagle模型config文件地址，通常将对应LLM原始模型中的num_layers改为1即可。
	--basepath	str	LLM原始模型权重地址
	--bs	int	训练模型的batch

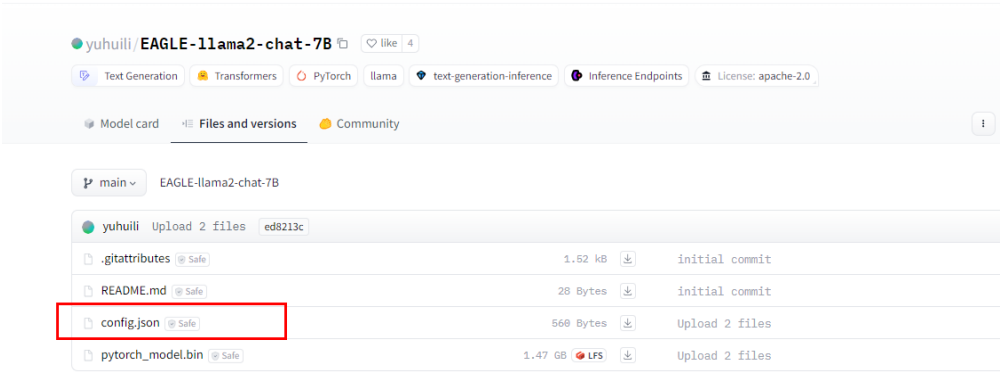
其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应Eagle模型地址。

图 4-9 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。

图 4-10 eagle config 文件



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即**步骤四：执行训练**中config文件所在目录）。

然后在llm_tools/spec_decode/EAGLE文件夹，执行

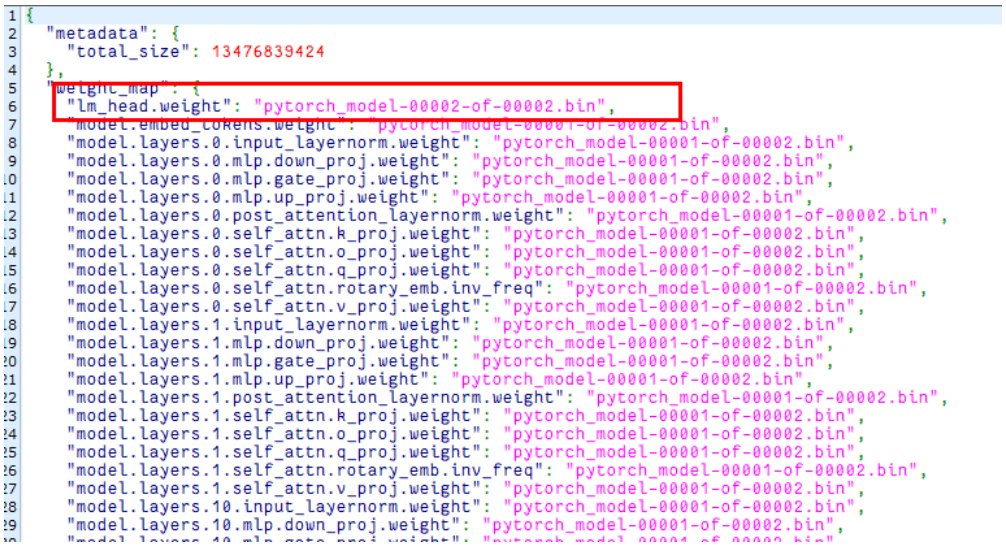
```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体参数解释如**表4-21**所示。

表 4-21 训练后权重转换适配 vllm 阶段

py文件名称	配置项	取值类型	配置说明
convert_eagle_ckpt_to_vllm_compatible.py	--base-path	str	LLM原始模型权重地址，例如 ./llama2-7b-chat
	--draft-path	str	Eagle模型权重存放地址，即**步骤四：执行训练**中config文件所在目录，例如 ./eagle_llama2-7b-chat
	--base-weight-name	str	为大模型包含lm_head的权重文件名，可以在base-path目录下的model.safetensors.index.json文件获取，例如llama2-7b-chat的权重名为pytorch_model-00001-of-00002.bin
	--draft-weight-name	str	Eagle小模型权重文件名，其格式为.bin文件或者.safetensors文件，即刚才移动的.bin文件或者.safetensors文件。

图 4-11 权重文件名



4.2.5.7 图模式

什么是 PTA 图模式

PTA图模式使用TorchAir框架（继承自PyTorch框架Dynamo模式）在昇腾NPU上进行图模式推理，可达到最大化消除算子下发瓶颈的目的。推荐在小模型以及MOE模型的场景开启PTA图模式，如Qwen2-1.5B，Qwen2-0.5B，mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8。另外，当前MoE模型的PTA图模式启动不支持multi step。

PTA 图模式参数配置

如果要开启PTA图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1   # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32    # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500    # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV    #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服

务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

什么是 CANN-GRAPH

CANNGraph图模式是一种Capture-Replay架构的Host图，可以有效消除Host瓶颈，支持模型输入动态shape，无需分档构图，构图较快。未设置INFER_MODE环境变量时，即默认模式下，部分模型会默认使用CANNGraph图模式启动来提升性能。

CANN-GRAPH 使用限制

CANNGraph图模式目前仅支持llama和qwen2系列架构的大语言模型单卡场景，包含该系列AWQ量化模型。由于部分算子暂未适配，其他场景(如Multi-lora)暂未支持。

CANN-GRAPH 参数设置

相关参数如下表所示：

服务启动方式	配置项	配置说明
offline	enforce_eager	设置该参数为True将关闭CANNGraph图模式，小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
online	--enforce-eager	设置该参数将关闭CANNGraph图模式，小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。

4.2.5.8 多模态

什么是多模态

多模态（Multimodality）是集成和处理两种或两种以上不同类型的信息或数据的方法和技术。具体来说，在机器学习和人工智能领域，多模态涉及的数据类型通常包括但不限于文本、图像、视频、音频和传感器数据。

多模态的主要目标是利用来自多种模态的信息来提升任务的表现力，提供更丰富的用户体验，或是获取更全面的数据分析结果。例如，在实际应用场景中，可以通过结合图像和文本信息来进行更好的对象识别或情感分析。

此外，多模态还可以细分为以下几个方面：

- 多模态理解：如何让计算机从不同种类的数据源中抽取有用的信息，并将其综合起来形成有意义的知识。
- 视觉大模型：这类模型专门针对图像和其他视觉数据设计，帮助计算机更好地理解和解释视觉世界。
- 多模态检索：这是指利用多种数据模态(如文本、图像、视频、音频等)进行信息检索的技术，旨在通过整合不同形式的数据，提供更精准的结果。

综上所述，多模态不仅仅是简单的特征融合，而是涵盖了广泛的理论基础及实践应用。这里的多模态是指多模态理解。

约束限制

机器以及最大卡数见**各模型支持的最小卡数和最大序列**。

启动多模态推理服务

当前支持Llava(llava-1.5系列模型)、llava-next(llava-v1.6系列模型)、InternVL2、MiniCPM、qwen-vl、qwen2-vl、llava-onevision模型，具体模型和权重地址参见**支持的模型列表**，推荐显卡数量参见**表4-25**。

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--quantization=${quantization} \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下（其余参数设置参考4.2启动推理服务基础参数说明）：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**表4-25**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

- --tensor-parallel-size：模型并行数。模型并行与流水线并行的乘积取值需要和启动的NPU卡数保持一致，可以参考**表4-25**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于multi-step调度。每次调度生成多个token，可以降低时延。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --chat-template：对话构建模板，可选参数。如：llava chat-template：${vllm_path}/examples/template_llava.jinja
- --quantization：为量化选项，可选参数，当前只支持awq，不传入默认为None即不启用量化。

多模态推理请求

通过online_serving.py方式发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如**表1 脚本参数说明**所示。

```
import base64
import requests
import argparse
import json
from typing import List
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

def get_stop_token_ids(model_path):
    with open(f"{model_path}/config.json") as file:
        data = json.load(file)
        if data.get('architectures')[0] == "InternVLChatModel":
            return [0, 92543, 92542]
    return None

def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    stop_token_ids = args.stop_token_ids if args.stop_token_ids is not None else
get_stop_token_ids(args.model_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
```

```
            "text": args.text
          },
          {
            "type": "image_url",
            "image_url": {
              "url": f"{image_base64}"
            }
          }
        ]
      }
    ],
    "max_tokens": args.max_tokens,
    "temperature": args.temperature,
    "ignore_eos": args.ignore_eos,
    "stream": args.stream,
    "top_k": args.top_k,
    "top_p": args.top_p,
    "stop_token_ids": stop_token_ids,
    "repetition_penalty": args.repetition_penalty,
  }
  response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
  print(response.json())

if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  # 必填
  parser.add_argument("--model-path", type=str, required=True)
  parser.add_argument("--image-path", type=str, required=True)
  parser.add_argument("--docker-ip", type=str, required=True)
  parser.add_argument("--served-port", type=str, required=True)
  parser.add_argument("--text", type=str, required=True)
  # 选填
  parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
  parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成
EOS token后继续生成token。可选
  parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的
文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
  parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越
意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
  parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启
流式推理。
  parser.add_argument("--max-tokens", type=int, default=16) # 生成序列的最大长度。必选
  parser.add_argument("--repetition-penalty", type=float, default=1.0) # 减少重复生成文本的概率。可选
  parser.add_argument("--stop-token-ids", nargs='+', type=int, default=None) # 停止tokens列表。可选
  args = parser.parse_args()
  post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip $
{docker_ip} --served-port ${port} --text 图片内容是什么
```

相关请求参数说明参照**多模态相关请求参数说明**。

多模态相关请求参数说明

表 4-22 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径

参数	是否必须	参数类型	描述
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-23 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0，1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必须	默认值	参数类型	描述
repetition_penalty	否	1.0	Float	减少重复生成文本的概率。
stop_token_ids	否	None	Int	停止tokens列表。internvl2和minicpmv需要传入，参考离线推理脚本examples/offline_inference_vision_language.py的stop_token_ids。

4.2.5.9 Chunked Prefill

什么是 Chunked Prefill

Chunked Prefill（Splitfuse）特性的目的是将长prompt request分解成更小的块，并在多个forward step中进行调度，只有最后一块的forward完成后才开始这个prompt request的生成。将短prompt request组合以精确填充step的空隙，每个step的计算量基本相等，达到所有请求平均延迟更稳定的目的。

关键行为：

1. 长prompts被分解成更小的块，并在多个迭代中进行调度，只有最后一遍迭代执行输出生成token。
2. 构建batch时，一个prefill块和其余槽位用decode填充，降低仅decode组batch的成本。

其优势主要包括：

- **提升效率**：通过合理组合长短prompt，保持模型高吞吐量运行。
- **增强一致性**：统一前向传递大小，降低延迟波动，使生成频率更稳定。
- **降低时延：**通过平衡prefill和decode的计算利用率，降低请求P90_ttft（time to first token）、P90_tpot(time per output token)时延。在短输入、短输出且高并发的场景优势明显。

约束限制

- 该特性不能和PD分离、Prefix Cache、KV Cache量化特性、multi-lora特性同时使用。
- LLama系列、Qwen系列模型支持此特性。

Chunked Prefill 参数配置

Chunked Prefill的依赖参数如下表所示。

表 4-24 依赖参数说明

配置项	取值类型	取值范围	配置说明
enable-chunked-prefill	bool	• true • false	• true：开启Chunked Prefill特性。 • false：不开启Chunked Prefill特性。
max-num-batched-tokens	int	>=256且是256的倍数	在Chunked Prefill模式下，该参数限制最大切分长度，允许比max-model-len小。推荐使用4096、8192甚至更大。

执行推理参考

1. Ascend vllm使用Chunked Prefill特性需参考**表4-24**，其它参数请参考**启动推理服务**。
2. 启动推理服务请参考**启动推理服务**。

4.2.5.10 multi-lora

什么是 multi-lora

LoRA（Low-Rank Adaptation）是一种适用于大模型的轻量化微调技术方法。原理是通过在模型层中引入低秩矩阵，将大模型的权重降维处理，来实现高效的模型适配。相比于传统的微调方法，LoRA不仅能大幅减少所需的训练参数，还降低了显存和计算成本，加快了模型微调速度。对于VLLM来说，使用LoRA进行多任务部署具有以下优势：

- **资源节省**：在大模型中引入LoRA，可以减少模型需要更新的参数量，从而节省NPU内存并提高推理速度。
- **轻量化适配**：无需改变原始模型结构，通过低秩矩阵的调整即可适配不同任务。
- **多任务并行**：支持同时加载多个LoRA模块，使得VLLM可以在不同任务间快速切换，提高多任务推理的效率。

约束限制

multi-lora特性不能和Chunked Prefill特性一起使用。

multi-lora 特性使用说明

如果需要使用multi-lora特性，需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
```

```
--max-loras=32 \
--max-cpu-loras=32
```

参数说明如下：

- --enable-lora表示开启lora挂载。
- --lora-modules后面添加挂载的lora列表，要求lora地址权重是Huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。发请求时model指定为lora1或者lora2即为LoRA推理。
- --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64，选择lora1与lora2中rank数量的较大值，比如lora1对应16rank，lora2对应32rank，挂载lora的最大rank数量为32。
- --max-loras表示支持的最大lora个数，最大32。
- --max-cpu-loras要求配置和--max-loras相同。

4.2.5.11 guided-decoding

什么是 guided-decoding

Guided Decoding是一种用于生成文本的策略，通过提供额外的上下文或约束，来引导模型生成更符合预期的结果。

比如使用openai启动服务，通过配置guided_json参数使用JSON Schema的架构来举例。

JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必须属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。

其优势主要如下：

- **上下文引导**：通过提供特定的提示或上下文信息，模型可以更好地理解生成内容的方向。
- **约束生成**：可以设定某些限制条件，如关键词、主题或风格，使生成的内容更加一致和相关。
- **提高质量**：通过引导，生成的文本通常更具逻辑性和连贯性，减少无关信息的出现。

约束限制

Guided Decoding特性不能和multi-step同时使用。

离线推理使用 Guided Decoding

离线推理，要使用guided-decoding，需要通过SamplingParams类中的GuidedDecodingParams进行配置。

下面是一种离线使用方式示例：

```
from vllm import LLM, SamplingParams
from vllm.sampling_params import GuidedDecodingParams

MODEL_NAME = ${MODEL_NAME}
llm = LLM(model=MODEL_NAME)

guided_decoding_params = GuidedDecodingParams(choice=["Positive", "Negative"])
```

```
sampling_params = SamplingParams(guided_decoding=guided_decoding_params)
outputs = llm.generate(
    prompts="Classify this sentiment: vLLM is wonderful!",
    sampling_params=sampling_params,
)
print(outputs[0].outputs[0].text)
```

MODEL_NAME表示对应模型路径。

在线推理使用 Guided Decoding

启动推理服务请参考**启动推理服务**章节。

在线推理使用Guided Decoding时，在发送的请求中包含上述guided_json架构，具体示例可参考以下代码。

```
curl -X POST http://${docker_ip}:8080/v1/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength",
    "max_tokens": 200,
    "temperature": 0,
    "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}"
}'
```

4.2.6 推理服务精度评测

本章节介绍了2种精度测评方式，分别为Lm-eval工具和MME工具。

- lm-eval工具适用于语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等，该工具为离线测评，不需要启动推理服务。
- MME工具适用于多模态模型的精度测试。目前支持模型：llava(llava-1.5系列模型)、llava-next(llava-v1.6系列模型)、minicpm、qwen-vl、internvl2、qwen2-vl、llava-onevision。

使用 Lm-eval 精度测评工具进行精度评测

使用lm-eval工具暂不支持pipeline_parallel方式，也不支持qwen-7b、qwen-14b、qwen-72b、chatglm2-6b、chatglm3-6b模型。

1. 安装精度评测工具。可以在原先的conda环境，进入到一个固定目录下，执行如下命令。

   ```
   rm -rf lm-evaluation-harness/
   git clone https://github.com/EleutherAI/lm-evaluation-harness.git
   cd lm-evaluation-harness
   git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
   pip install -e .
   ```

2. 执行如下精度测试命令，可以根据参数说明修改参数。

   ```
   lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=$
   ```

```
{gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=$
{quantization},distributed_executor_backend='ray' \
--tasks ${task}  --batch_size ${batch_size}  --log_samples  --cache_requests true  --trust_remote_code
--output_path ${output_path}
```

参数说明:

- model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
  - vllm_path是模型权重路径；
  - max_model_len是最大模型长度，默认设置为4096；
  - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
  - tensor_parallel_size是使用的卡数；
  - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
  - distributed_executor_backend是开启多进程服务方式，选择ray开启。
- model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
- tasks：评测数据集任务，比如openllm。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
- output_path：结果保存路径。

以llama3.2-1b模型的权重为例，在加载非量化场景下使用lm-eval，参考命令如下。

```
lm_eval --model vllm  --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-
Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_mo
del_len=4096,distributed_executor_backend='ray' \
--tasks openllm  --batch_size auto  --log_samples  --cache_requests true  --trust_remote_code --
output_path ./
```

以llama3.1-70b模型的权重为例，在smoothquant量化场景下使用lm-eval，参考命令如下。参数quantization="smoothquant"表示smoothquant量化，可以根据实际量化场景替换为awq等。

```
lm_eval --model vllm  --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/
llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,ma
x_model_len=4096,quantization="smoothquant",distributed_executor_backend='ray' \
--tasks openllm  --batch_size auto  --log_samples  --cache_requests true  --trust_remote_code --
output_path ./
```

lm-eval可以验证multi-lora的精度，参考命令如下：

```
lm_eval --model vllm --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=$
{tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},max_model_len=$
{max_model_len},lora_local_path=$
{lora_local_path},distributed_executor_backend='ray',enable_lora=True \
--tasks ${task}  --batch_size ${batch_size}  --log_samples  --cache_requests true  --trust_remote_code
--output_path ${output_path} --device cuda
```

参数说明：

- enable_lora=True表示开启multi-lora的精度验证。如果不开启multi-lora的精度验证，不体现enable_lora参数即可。
- lora_local_path=${lora_local_path}是挂载适配器对应路径，取值和**multi-lora特性使用说明**中的参数lora1=/path/to/lora/adapter1/保持一致。

使用 MME 精度测评工具进行精度评测

1. MME数据集获取。

   请用户自行获取**MME评估集**，将MME评估集上传至llm_tools/llm_evaluation/mme_eval/data/eval/目录中。

2. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation/mme_eval目录中，代码目录结构如下。

   ```
   mme_eval
   ├──metric.py       #MME精度测试脚本
   ├──MME.sh          #运行MME脚本
   ```

3. 启动MME精度测试脚本。

   ```
   export MODEL_PATH=/data/nfs/model/InternVL2-8B/
   export MME_PATH=/llm_tools/llm_evaluation/mme_eval/data/eval/MME
   export MODEL_TYPE=internvl2
   export OUTPUT_NAME=internvl2-8B
   export ASCEND_RT_VISIBLE_DEVICES="0:1:2:3:4:5:6:7"
   bash MME.sh
   ```

   参数说明:

   a. MODEL_PATH：模型权重路径，默认为空。

   b. MME_PATH：MME数据集路径，默认当前路径。

   c. MODEL_TYPE：模型类型。当前可选模型类型包括：llava、llava-next、minicpm、qwen-vl、internvl2、qwen2-vl、llava-onevision。

   d. OUTPUT_NAME：输出结果文件名称， 默认llava。

   e. ASCEND_RT_VISIBLE_DEVICES：表示支持多个模型服务实例，同时支持模型并行，如 0,1:2,3 默认0卡。

   f. QUANTIZATION：为量化选项，不传入默认为None即不启用量化；支持w4a16，需配套对应的权重使用。

   g. GPU_MEMORY_UTILIZATION：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。

   脚本运行完成后，测试结果输出在终端。

4.2.7 推理服务性能评测

4.2.7.1 语言模型推理性能测试

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

benchmark 代码目录

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。具体代码目录如下。

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # 评测静态性能脚本
```

```
  |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤三：上传代码包和权重文件**中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。

   ```
   cd benchmark_tools
   ```

   语言模型脚本相对路径是tools/llm_evaluation/benchmark_tools/benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

   ```
   python benchmark_parallel.py --backend openai --host ${docker_ip} --port ${port} --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```

   参数说明

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等后端。本文档使用的推理接口是openai。
   - --host：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5。
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
   - --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。
   - --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
   - --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
   - --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。
   - --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
   - --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机

token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。不输入默认为random。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。

- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-12 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```

generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

```
cd benchmark_tools
conda activate python-3.9.10
```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_serving.py --backend openai --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。

- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-13 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

单条请求性能测试

针对openai的/v1/completions以及/v1/chat/completions两个非流式接口，请求体中可以添加可选参数"return_latency"，默认为false，如果指定该参数为true，则会在相应请求的返回体中返回字段"latency"，返回内容如下：

1. prefill_latency（首token时延）：请求从到达服务开始到生成首token的耗时
2. model_prefill_latency（模型计算首token时延）：服务从开始计算首token到生成首token的耗时
3. avg_decode_latency（平均增量token时延）：服务计算增量token的平均耗时
4. time_in_queue（请求排队时间）：请求从到达服务开始到开始被调度的耗时
5. request_latency（请求总时延）：请求从到达服务开始到结束的耗时

以上指标单位均是ms，保留2位小数。

4.2.7.2 多模态模型推理性能测试

多模态模型推理的性能测试目前仅支持静态性能测试。

静态性能测试是指评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。工具相关介绍在**benchmark代码目录**。

约束限制

当前版本仅支持语言+图片多模态性能测试。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤三：上传代码包和权重文件**中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。

```
cd benchmark_tools
```

3. 多模态模型脚本相对路径是llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python modal_benchmark_parallel.py \
--host ${docker_ip} \
--port ${port} \
--tokenizer /path/to/tokenizer  \
--epochs 5 \
--parallel-num 1 4 8 16 32 \
--prompt-tokens 1024 2048 \
--output-tokens 128 256 \
--height ${height} \
--width ${width} \
--benchmark-csv benchmark_parallel.csv
```

参数说明

- --host：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --height: 图片长度（分辨率相关参数）。
- --width: 图片宽度（分辨率相关参数）。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中。

4.2.8 附录

4.2.8.1 各模型支持的最小卡数和最大序列

基于vLLM（v0.6.3）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

e5-mistral-7B和gte-Qwen2-7B-instruct模型，使用openai启动服务，发送推理请求使用的是接口curl -X POST http://localhost:port/v1/embedding。

表 4-25 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128
12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16
22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64

27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32
32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64
42	deepseek-llm-7b	1	16	1	32
43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llama-3.1-405B-AWQ	-	-	8	32
51	qwen2-57b-a14b	-	-	2	16
52	deepseek-v2-lite-16b	2	4	1	4
53	deepseek-v2-236b	-	-	8	4
54	qwen-vl	1	64	1	64
55	qwen-vl-chat	1	64	1	64

56	MiniCPM-v2	2	16	1	16
57	e5-mistral-7B	1	8	1	64
58	gte-Qwen2-7B-instruct	1	8	1	64
59	llava-1.5-7b	1	16	1	32
60	llava-1.5-13b	1	8	1	16
61	llava-v1.6-7b	1	16	1	32
62	llava-v1.6-13b	1	8	1	16
63	llava-v1.6-34b	4	32	2	64
64	internvl2-8b	1	16`	1	32
65	internvl2-26b	2	8	1	8
66	internvl2-40b	-	-	2	32
67	internVL2-Llama3-76B	-	-	4	8
68	internVL2-Llama3-76B-AWQ	2	8	1	8
69	MiniCPM-v2.6	-	-	1	8
70	qwen2-vl-2B	1	8	1	8
71	qwen2-vl-7B	1	8	1	32
72	qwen2-vl-72B	-	-	4	32
73	qwen2-vl-72B-AWQ	2	32	1	32
74	llava-onevision-qwen2-0.5b-ov-hf	2	8	1	8
75	llava-onevision-qwen2-7b-ov-hf	2	8	1	8

“-”表示不支持。

4.2.8.2 Ascend-vLLM 推理常见问题

问题 1：在推理预测过程中遇到 NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题 2：在推理预测过程中遇到 ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

问题 3：使用 llama3.1 系列模型进行推理时报错

使用llama3.1系模型进行推理时报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

解决方法：

升级transformers版本到4.43.1

```
pip install transformers --upgrade
```

问题 4：使用 SmoothQuant 进行 W8A8 进行模型量化时报错

使用SmoothQuant进行W8A8进行模型量化时报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42

```
pip install transformers==4.42 --upgrade
```

问题 5：使用 AWQ 转换 llama3.1 系列模型权重出现报错

使用AWQ转换llama3.1系列模型权重出现报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor'

解决方法：

该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

问题 6：使用 Qwen2-7B、Qwen2-72B 模型有精度问题，重复输出感叹号

检查**启动推理服务**章节中，高精度模式的环境变量是否开启。

问题 7：使用 autoAWQ 进行 qwen-7b 模型量化时报错

使用autoAWQ进行qwen-7b模型量化时报错：TypeError: 'NoneType' object is not subscriptable

解决方法：

修改qwen-7b权重路径下modeling_qwen.py第39行为SUPPORT_FP16 = True

问题 8：使用 benchmark-tools 对 GLM 系列模型进行性能测试报错

使用benchmark-tools对GLM系列模型进行性能测试报错TypeError: _pad() got an unexpected keyword argument 'padding_side'

解决方法：

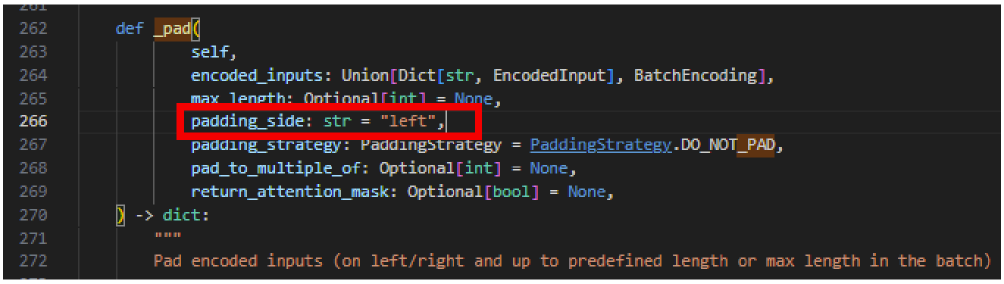
1、下载最新的tokenization_chatglm.py，替换原来权重里的tokenization_chatglm.py。

https://huggingface.co/THUDM/glm-4-9b-chat/blob/main/tokenization_chatglm.py

https://huggingface.co/THUDM/chatglm3-6b/blob/main/tokenization_chatglm.py

或者2、修改tokenization_chatglm.py，在266行增加padding_side: str = "left"，如**图4-14**所示。

图 **4-14** tokenization_chatglm.py



```
262     def _pad(
263             self,
264             encoded_inputs: Union[Dict[str, EncodedInput], BatchEncoding],
265             max_length: Optional[int] = None,
266             padding_side: str = "left",
267             padding_strategy: PaddingStrategy = PaddingStrategy.DO_NOT_PAD,
268             pad_to_multiple_of: Optional[int] = None,
269             return_attention_mask: Optional[bool] = None,
270     ) -> dict:
271         """
272         Pad encoded inputs (on left/right and up to predefined length or max length in the batch)
```

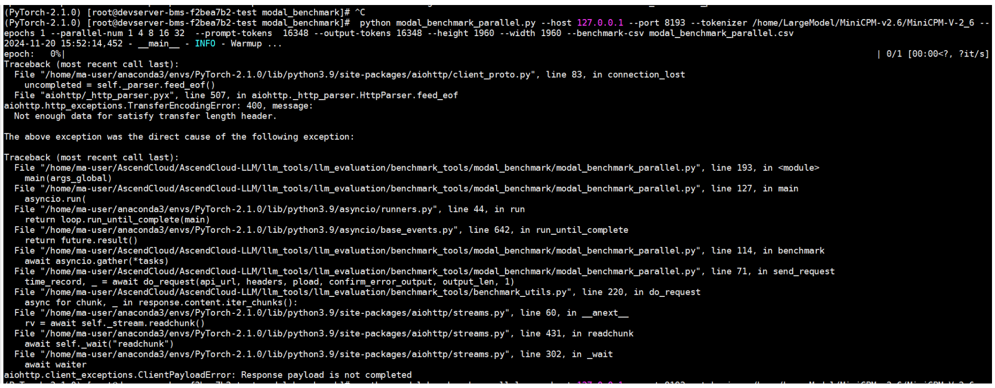
问题 9：使用 benchmark-tools 访问推理服务返回报错

使用benchmark-tools访问推理服务时，输入输出的token和大于max_model_len，服务端返回报错Response payload is not completed，见**图4-15**。

再次设置输入输出的token和小于max_model_len访问推理服务，服务端响应200，见**图4-16**。

客户端仍返回报错Response payload is not completed，见**图4-17**。

图 **4-15** 服务端返回报错 Response payload is not completed



```
(PyTorch-2.1.0) [root@devserver-bms-f2bea7b2-test modal_benchmark]# ^C
(PyTorch-2.1.0) [root@devserver-bms-f2bea7b2-test modal_benchmark]#  python modal_benchmark_parallel.py --host 127.0.0.1 --port 8193 --tokenizer /home/LargeModel/MiniCPM-v2.6/MiniCPM-V-2_6 --epochs 1 --parallel-num 1 4 8 16 32  --prompt-tokens  16348 --output-tokens 16348 --height 1960 --width 1960 --benchmark-csv modal_benchmark_parallel.csv
2024-11-20 15:52:14,452 - __main__ - INFO - Warmup ...
epoch:   0%|                                                                  | 0/1 [00:00<?, ?it/s]
Traceback (most recent call last):
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/aiohttp/client_proto.py", line 83, in connection_lost
    uncompleted = self._parser.feed_eof()
  File "aiohttp/_http_parser.pyx", line 507, in aiohttp._http_parser.HttpParser.feed_eof
aiohttp.http_exceptions.TransferEncodingError: 400, message:
  Not enough data for satisfy transfer length header.

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py", line 193, in <module>
    main(args_global)
  File "/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py", line 127, in main
    asyncio.run(
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/asyncio/runners.py", line 44, in run
    return loop.run_until_complete(main)
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/asyncio/base_events.py", line 642, in run_until_complete
    return future.result()
  File "/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py", line 114, in benchmark
    await asyncio.gather(*tasks)
  File "/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py", line 71, in send_request
    time_record, _ = await do_request(api_url, headers, pload, confirm_error_output, output_len, 1)
  File "/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools/benchmark_utils.py", line 220, in do_request
    async for chunk, _ in response.content.iter_chunks():
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/aiohttp/streams.py", line 60, in __anext__
    rv = await self._stream.readchunk()
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/aiohttp/streams.py", line 431, in readchunk
    await self._wait("readchunk")
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/aiohttp/streams.py", line 302, in _wait
    await waiter
aiohttp.client_exceptions.ClientPayloadError: Response payload is not completed
```

图 4-16 服务端响应 200

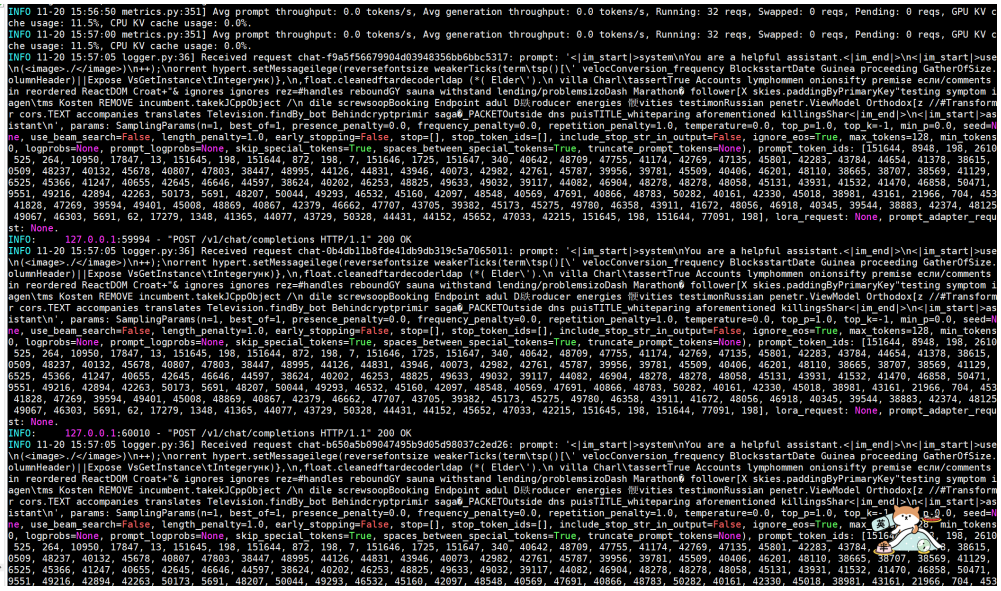
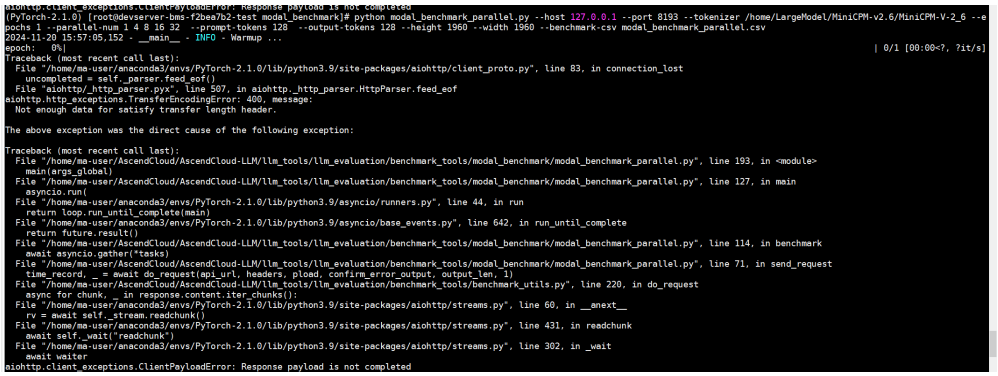


图 4-17 仍返回报错 Response payload is not completed



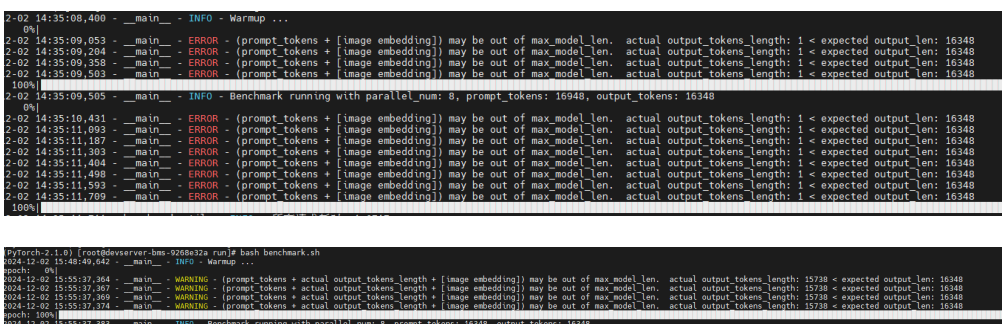
解决方法：

安装brotlipy后返回正确报错

```
pip install brotlipy
```

问题 10：使用 benchmark-tools 访问推理客户端返回报错或警告

使用benchmark-tools访问推理客户端返回报错或警告：actual output_tokens_length < expected output_len



解决方法：

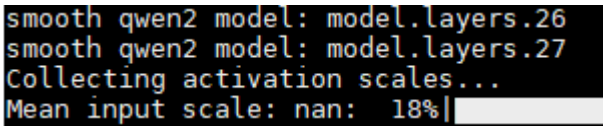
减少参数--prompt-tokens和--output-tokens的值，或者增大启动服务的参数--max-model-len的值。

问题 11：使用离线推理时，性能较差或精度异常

解决方法：将block_size大小设置为128

```
from vllm import LLM, SamplingParams
llm = LLM(model="facebook/opt-125m", block_size=128)
```

问题 12：使用 SmoothQuant 做权重转换时，scale 显示为 nan，或推理时精度异常

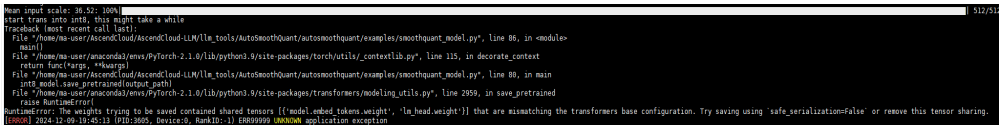


涉及模型：qwen2-1.5b、qwen2-7b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/utils/utils.py中的build_model_and_tokenizer函数，将torch_dtype类型从torch.float16改成torch.bfloat16

```
kwargs = {"torch_dtype": torch.bfloat16, "device_map": "auto"}
```

问题 13：使用 SmoothQuant 做权重转换时，有如下报错



涉及模型：qwen2-1.5b、qwen2-0.5b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples/smoothquant_model.py中的main函数，保存模型时将safe_serialization指定为False

```
int8_model.save_pretrained(output_path,safe_serialization=False)
```

4.3 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.912）

4.3.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考表4-28获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。
- Lite Server驱动版本要求23.0.6
- PyTorch版本：2.3.1
- 确保容器可以访问公网。

文档更新内容

6.3.912版本相对于6.3.911版本新增如下内容：

1. 代码结构发生变化，统一了modellink和llama-factory的启动方式。
2. 继承911版本启动方式以外增加新的启动方式：
   ```
   ascendfactory-cli train xx.yaml model-name exp_name
   ```
3. 相对于6.3.911版本仅是使用run_type来指定训练的类型，只能区分 预训练、全参微调和lora微调但实际上预训练和sft是训练的不同阶段，全参、lora是训练参数设置方式。为了更加明确的区分不同策略，以及和llama-factory对齐，6.3.912版本调整以下参数：
   - 新增 STAGE，表示训练的阶段，可以选择的参数包括: {pt, sft}.
   - 新增 FINETUNING_TYPE，表示微调的策略，可以选择的参数包括：{full, lora}
   - 删除 RUN_TYPE

   所以当前的组合情况为：

项目	full	lora
pt（预训练）	√	√
sft（指令微调）	√	√

训练支持的模型列表

本方案支持以下模型的训练，如表4-26所示。

表 4-26 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

序号	支持模型	支持模型参数量	权重文件获取地址
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLM v3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan 2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

表 4-27 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendFactory训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
训练	预训练/微调	介绍如何进行训练，包括训练数据处理、超参配置、训练任务、性能查看。

4.3.2 准备工作

4.3.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-32**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite

Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.3.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-28**所示，模型列表、对应的开源权重获取地址如**表1 支持的模型列表**所示。

表 4-28 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

获取对应模型的权重文件，获取链接参考**表4-26**。

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器hfd：**hfd** 是本站开发的huggingface专用下载工具，基于成熟工具git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了git clone repo_url的方式下载，但是不支持断点续传，并且clone会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.912中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
  |——llm_train                  # 模型训练代码包
    |——AscendFactory
      |——examples/                # config配置文件目录
      |——data.tgz                # 样例数据压缩包
      |——third-party/             # patch包
      |——src/acs_train_solution/  # 训练运行包
      |——intall.sh               # 需要的依赖包
      |——scripts_llamafactory/    # llamafactory兼容旧版本启动方式目录
      |——scripts_modellink/       # modelLink兼容旧版本启动方式目录
      |——Dockerfile
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                                # 模型训练代码包
    |——AscendFactory
      |——examples/config/                          # 配置文件
        |——deepspeed/                       # deepspeed配置json文件
        |——modellink_performance_cfgs.yaml  # ModelLink训练配置json文件
        |——.......
      |——data.tgz                 #样例数据压缩包
      |——intall.sh                # 需要的依赖包
      |——scripts_modellink/        # modelLink兼容旧版本启动方式目录
        |——llama3                 # llama3系列模型执行脚本的文件夹
        |——qwen2.5                # Qwen2.5系列模型执行脚本的文件夹
        |—— ...
        |—— dev_pipeline.sh       # 系列模型共同调用的多功能的脚本
      |——third-party/             # patch包
      |——src/acs_train_solution/   # 训练运行包
        |——ascendcloud_patch/     # patch补丁包
        |——benchmark/             #工具包，存放数据集及基线数据
          |——trainer.py          # 训练启动脚本
          |——performance.py      # benchmark训练性能比较启动脚本
          |——accuracy.py         # benchmark训练精度启动脚本
    |——model/Qwen2-7B/             # 权重词表文件目录，如Qwen2-7B
    |——training_data               # 原始数据目录
      |——alpaca_gpt4_data.json    # 微调数据
      |——train-00000-of-00001-a09b74b3ef9c3b56.parquet #预训练数据
    |——{output_dir} #{OUTPUT_SAVE_DIR}或yaml文件{output_dir}参数设置值
       # 自动生成数据目录结构
      |—— preprocessed_data
      |——converted_hf2mg_weight_TP${TP}PP${PP}
      |——checkpoint    # 训练完成生成目录Qwen2-7B，自动生成
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

```
cd /home/ma-user/ws
mkdir -p tokenizers/Llama2-70B
```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.3.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- 微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS指令微调数据**：本案例中还支持MOSS格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
  ```

```
 "meta_instruction": "",
 "num_turns": 3,
 "chat": {
  "turn_1": {
   "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
   "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
   "Commands": "<|Commands|>: None<eoc>\n",
   "Tool Responses": "<|Results|>: None<eor>\n",
   "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
  },
  "turn_2": { ... },
  "turn_3": { ... },
 "category": "Brainstorming"
}
```


- **LLama-Factory Alpaca指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
```



```
            "value": "人类指令"
        },
        {
            "from": "function_call",
            "value": "工具参数"
        },
        {
            "from": "observation",
            "value": "工具结果"
        },
        {
            "from": "gpt",
            "value": "模型回答"
        }
        ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.3.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-29 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527

表 4-30 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.3.1

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --cpus 192 \
   ```

```
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
   ```
   docker exec -it ${container_name} bash
   ```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
4. 使用ma-user用户安装依赖包。
   ```
   #进入scripts目录换
   cd /home/ma-user/ws/llm_train/AscendFactory
   #执行安装命令
   sh install.sh modellink
   ```

⚠ 注意

- 在执行 install.sh 安装命令时，需要确认机器是否已连通网络。若无法连通网络或无法git clone下载代码，用户则需要找到已连通网络的机器（本章节以Linux系统机器为例）将下载完成的源码放置**代码目录**：AscendFactory/third-party下，命令如下

```
# 三方开源源码
git clone https://gitee.com/ascend/MindSpeed.git
git clone https://github.com/huggingface/transformers.git
git clone https://github.com/NVIDIA/Megatron-LM.git
git clone https://gitee.com/ascend/ModelLink.git
```

  或找有网络机器使用**DockerFile构建镜像（可选）**构造新镜像后使用新镜像。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 脚本中的 transformers 的版本。由默认 transformers==4.46.1 修改为：transformers==4.44.2
- 为了避免因使用不同版本的 transformers 库进行训练和推理而导致冲突的问题，建议用户分别为训练和推理过程创建独立的容器环境。

5. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendFactory/third-party文件夹中。下载的源码文件结构如下：

```
AscendFactory/third-party/
   |——MindSpeed/          # MindSpeed昇腾大模型加速库
   |——Megatron-LM/        # 适配昇腾的Megatron-LM训练框架
   |——ModelLink/          # ModelLink端到端的大语言模型方案
      |——megatron/        # 注意：该文件夹从Megatron-LM中复制得到
      |——...
   |——transformers.patch
   |——llama-factory.patch
```

DockerFile 构建镜像（可选）

本章节主要介绍通过DockerFile文件构建训练镜像，将训练过程中依赖包封装使用，过程中需要连接互联网git clone，请确保环境可以访问公网，详解操作如下：

进入代码包Dockerfile文件同级目录：

```
cd /home/ma-user/ws/llm_train/AscendFactory
```

构建新镜像：

```
docker build -t <镜像名称>:<版本名称>  .
```

如无法访问公网则需配置代理，增加`--build-arg`参数指定代理地址确保访问公网。

```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx"
--network=host --build-arg install_type=modellink -t <镜像名称>:<版本名称>  .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_3_ascend:20241106
- install_type:安装类型，默认为all，可选【modellink、llmafactory、all】

⚠ 注意

构建镜像前需保证Dockerfile文件内容中镜像名与本文档**镜像**保持一致，如不同则需修改为一致。

```
# 修改以下内容：
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/xxx
```

4.3.3 执行训练任务

4.3.3.1 执行训练任务（推荐）

新的训练方式将统一管理训练日志、训练结果和训练配置，使用yaml配置文件方便用户根据自己实际需求进行修改。推荐用户使用该方式进行训练。

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

步骤二 修改训练 Yaml 配置文件

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器（默认开启）	断点续训：加载权重+优化器
• pt • sft	ckpt_load_type=0	ckpt_load_type=1 user_converted_ckpt_path=xxx，(转换MG的权重)	ckpt_load_type=2 user_converted_ckpt_path=xxx（训练输出MG权重）

修改、添加**代码目录**下modellink_performance_cfgs.yaml文件参数配置，样例yaml配置文件结构如下：

- base块：基础配置块，主要为公共配置参数
- ModelName块：该模型所需配置的参数，如qwen2.5-7b块
- exp_name：实验块，训练策略-序列长度所需参数配置

样例yaml文件仅展示常用实验配置，如需其他配置需根据样例自行添加。

图 4-18 yaml 样例


```
base: &base
  backend: modellink
  scirpts_dir: /home/ma-user/AscendFactory/script
  dataset: /path/to/alpaca_en_demo.json
  handler-name: AlpacaStyleInstructionHandler

qwen1.5-7b:
  _base: &qwen1_5-7b
    <<: *base
    model_name_or_path: /path/to/Qwen1.5-7B-Chat
    prompt-type: qwen
  full:
    <<: *qwen1_5-7b
    stage: sft
    finetuning_type: full
    tensor-model-parallel-size: 1
    pipeline-model-parallel-size: 4
    output_dir: ./saved_dir_for_output/qwen1.5-7b
```

根据以下步骤修改yaml文件：

1. 数据集选择:dataset和processed_data_dir参数二选一，详解如下：

参数	示例值	参数说明
dataset	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	训练时指定的输入数据路径。请根据实际规划修改。用户根据训练情况二选一；
processed_data_dir	/home/ma-user/ws/xxx	已处理好数据路径目录，如有处理完成数据可设置此参数

2. 权重文件、输出目录及其他重要参数设置，详解如下：

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B	【**必修改**】加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
output_dir	/home/ma-user/ws/save_dir	【**必修改**】训练任务结束生成日志及权重文件目录
scirpts_dir	/home/ma-user/ws/llm_train/AscendFactory/scripts_modellink	【**必修改**】ModelLink脚本相对或绝对路径，用于方便加载脚本

参数	示例值	参数说明
ckpt_load_type	1	【**可选**】默认为1 • 0，不加载权重 • 1，加载权重不加载优化器状态【**增量训练**】 • 2，加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
user_converted_ckpt_path	/home/ma-user/ws/xxx	【**可选**】已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。 • 增量训练：转换Megatron权重，如不指定默认为${output_dir}/converted_hf2mg_weight_TP{tp}PP{pp}目录。 • 断点续训：训练过程中保存的某个权重，可参考**断点续训和故障快恢说明**
train_auto_resume	false	【**可选**】是否开启【**故障快恢**】功能，【true、false】默认false不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。可参考**断点续训和故障快恢说明**
handler-name	• GeneralPretrainHandler • GeneralInstructionHandler • MOSSInstructionHandler • AlpacaStyleInstructionHandler • SharegptStyleInstructionHandler	示例值需要根据数据集${dataset}的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSInstructionHandler：使用微调的moss数据集 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集

3. 其他参数设置，详解如下：

参数	示例值	参数说明
stage	pt	表示训练类型。可选择值： ● pt：预训练 ● sft：指令微调
finetuning_type	full	表示训练策略。可选择值： ● full：全参微调 ● lora：lora微调
prompt-type	qwen	当handler-name为：【 AlpacaStyleInstructionHandler,SharegptStyleInstructionHandler 】需指定。
micro-batch-size	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
global-batch-size	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
tensor-model-parallel-size	8	表示张量并行。
pipeline-model-parallel-size	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
context-parallel-size	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加此值（≥ 2）。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
lr	2.5e-5	学习率设置。
min-lr	2.5e-6	最小学习率设置。
seq-length	4096	要处理的最大序列长度。
convert_mg2hf_at_last	true	是否将Megatron格式的权重转换为HuggingFace格式的权重，默认true。 true表示转换格式，false表示不转换格式。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
train-iters	10	非必填。表示训练step迭代次数，有默认值
seed	1234	随机种子数。每次数据采样时，保持一致。

参数	示例值	参数说明
save-interval	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
save-total-limit	-1	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

步骤三 启动任务

⚠ 注意

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务如产生mc2融合算子错误，可参考**mc2融合算子报错**

1. 创建test-benchmark，该目录存放训练生成的权重文件及训练日志可以多次执行，
```
# 任意目录创建
mkdir test-benchmark
```
2. 进入test-benchmark目录执行训练命令，卡数及其它配置参考**NPU卡数取值表**按自己实际情况决定。

   单机<可选>：
```
# 默认8卡
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
# 指定设备卡数，如2卡
ASCEND_RT_VISIBLE_DEVICES=0,1 ascendfactory-cli train <cfgs_yaml_file> <model_name>
<exp_name>
```
   多机<可选>多机同时执行：
```
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --
num_nodes <nodes> --rank <rank>
```
   - <cfgs_yaml_file>：ModelLink配置yaml文件地址，如**代码目录**中modellink_performance_cfgs.yaml相对或绝对路径，根据自己要求执行
   - <model_name>：训练模型名，如qwen1.5-7b，需与<cfgs_yaml_file>里面对应
   - <exp_name>：实验名称：指定本次实验的具体配置，还包括数据配置等，比如full, lora等，该名称需要和<cfgs_yaml_file>里面对应。

- --master_addr <master_addr>：主master节点IP，一般选rank0为主master。
- --num_nodes <nodes>：训练节点总个数
- --rank <rank>：节点ID，从0开始，一般选rank0为主master。

4.3.3.2 执行训练任务（历史版本）

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和**0_pl_sft_13b.sh** 。

修改模型训练脚本中的配置，参数详解可查看**训练参数说明**，其中【GBS、MBS、TP、PP】参数值可参考**模型推荐参数、NPU卡数**设置。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

⚠ 注意

同时开启故障快恢和断点续训时需满足以下条件：

- 如果用户指定${USER_CONVERTED_CKPT_PATH} 因故障快恢读取权重的优先级最高则训练过程的权重保存路径${OUTPUT_SAVE_DIR}/saved_checkpoints 必须为空，否则此参数无效断点续训失效。
- 如果就是使用最新的训练权重进行断点续训（暂停+启动场景），那么可以同时指定MA_TRAIN_AUTO_RESUME =1和 ${USER_CONVERTED_CKPT_PATH}训练过程的权重保存路径，加载路径一致。
- 故障快恢依赖训练过程的权重保存路径。所以如果开启MA_TRAIN_AUTO_RESUME=1， 则用户指定的权重加载路径${USER_CONVERTED_CKPT_PATH}不能是训练过程的权重保存路径。

步骤三 启动训练脚本

⚠ 注意

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务如产生mc2融合算子错误，可参考**mc2融合算子报错**

修改超参值后启动训练脚本，以 Llama2-70b-sft为例，各个模型NPU卡数可参考**模型推荐参数、NPU卡数**。

多机启动

多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendFactory** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **方法一：传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  多机执行命令为：sh scripts_modellink/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
  <NNODES=4> <NODE_RANK=0>
  示例:
  #第一台节点
  sh scripts_modellink/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts_modellink/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts_modellink/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts_modellink/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
  ```
- **方法二：定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts_modellink/llama2/0_pl_sft_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts_modellink/llama2/0_pl_sft_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts_modellink/llama2/0_pl_sft_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts_modellink/llama2/0_pl_sft_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendFactory** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **方法一：传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts_modellink/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost>
  <NNODES=1> <NODE_RANK=0>
  示例:
  sh scripts_modellink/llama2/0_pl_sft_13b.sh localhost 1 0
  ```
- **方法二：定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts_modellink/llama2/0_pl_sft_13b.sh
  ```

  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts_modellink/
  llama2/0_pl_sft_7b.sh
  ```

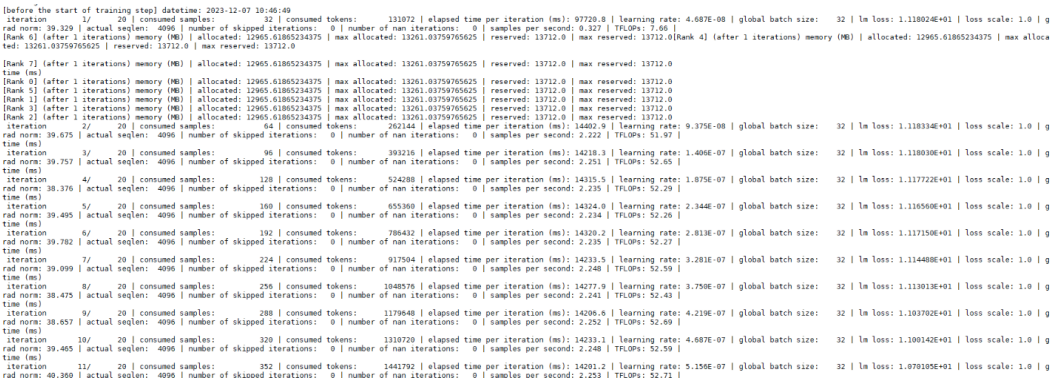
最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.3.4 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-19 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${OUTPUT_SAVE_DIR}/log路径下获取。

查看性能

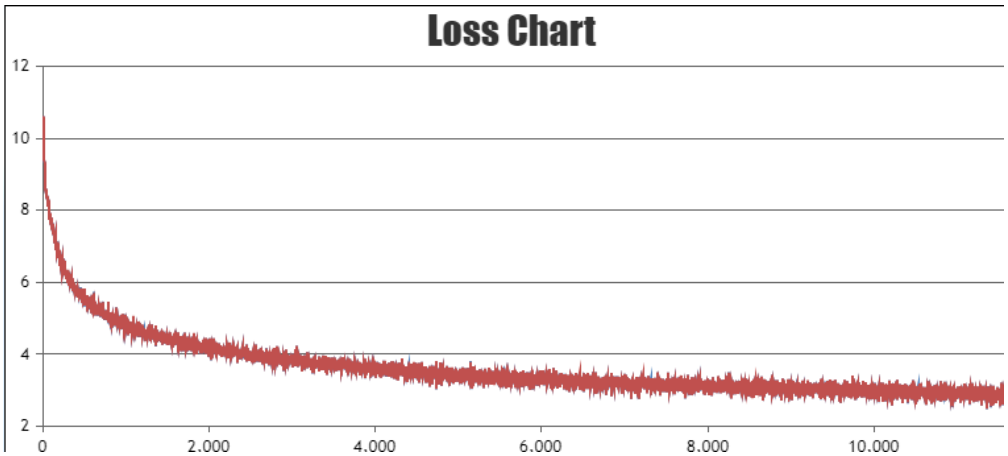
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-31**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-20**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-20 Loss 收敛情况（示意图）



4.3.5 训练脚本说明参考

4.3.5.1 训练参数配置说明【旧】

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，请根据实际模型修改。

表 4-31 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	【**必改**】训练时指定的输入原始数据路径。请根据实际规划修改。用户根据训练情况二选一；
USER_PROCESSED_DATA_DIR	/home/ma-user/ws/process_data	【**可选**】如已有预处理完成数据可指定此目录，训练过程中会优先加载此目录，跳过数据预处理过程；默认无此参数。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B	【**必改**】。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
OUTPUT_SAVE_DIR	/home/ma-user/ws/save_dir/llama2-70B_sft_lora_4096	【**必改**】。训练任务结束生成日志及权重文件目录。根据实际情况决定
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
STAGE	pt	表示当前的训练阶段。可选择值：【pt、sft】 • sft：代表监督微调； • pt：代表预训练；
FINETUNING_TYPE	full	表示训练策略。可选择值【full、lora】： • full：全参微调 • lora：lora微调

参数	示例值	参数说明
DATA_TYPE	【GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler】	【**必改**】示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSInstructionHandler：使用微调的moss数据集。 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。

参数	示例值	参数说明
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	10	非必填。表示训练step迭代次数，会进行自动计算得出。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SN	5120	指定的输入数据集中数据的总数量。更换数据集时，需要修改。
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
MA_TRAIN_AUTO_RESUME	False	**【可选】【故障快恢】**是否开启此功能，【True、False】默认False不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。详见**断点续训和故障快恢说明**

参数	示例值	参数说明
CKPT_LOAD_TYPE	1	**可选**【0、1、2】，默认为1 • 0： 不加载权重 • 1：加载权重不加载优化器状态【增量训练】 • 2：加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
USER_CONVERTED_CKPT_PATH	/home/ma-user/ws/xxx	【**可选**】已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。 • 增量训练：转换Megatron权重，如不指定默认为${output_dir}/converted_hf2mg_weight_TP{tp}PP{pp}目录。 • 断点续训：训练过程中保存的某个权重，详见**断点续训和故障快恢说明**

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-32**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-32 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点&4*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点&4*Ascend
22	mistral	mistral-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	full	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

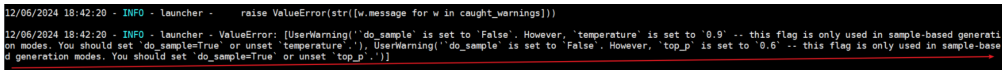
序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.3.5.2 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.46.1），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendFactory/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-21所示。

图 4-21 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-22所示。

图 4-22 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-23 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321 
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327 
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-24所示。

图 4-24 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295 
```

图 4-25 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-26所示。

图 4-26 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.3.5.3 断点续训和故障快恢说明

相同点

断点续训（Checkpointing）和故障快恢都是指训练中断后可从训练中一定间隔（${save-interval}）保存的模型（包括模型参数、优化器状态、训练迭代次数等）继续训练恢复，而不需要从头开始。

不同点

- 断点续训：可指定加载训练过程中生成的Megatron格式权重(${user_converted_ckpt_path})
- 故障快恢：默认加载${output_dir}/saved_checkpoints中最大迭代次数（iter_000xxxx）Megatron格式权重文件。

⚠ 注意

1. lora微调不支持断点续训
2. 启动前需检查latest_checkpointed_iteration.txt文件中内容是否与所需iter_000xxxx数字（表示训练后保存权重对应迭代次数）保持一致，不一致则修改latest_checkpointed_iteration.txt内容与iter_000xxxx保持一致。

```
|——${saved_checkpoints}
  |——iter_0000010
  |——iter_0000020
  |———latest_checkpointed_iteration.txt
```

示例，latest_checkpointed_iteration.txt文件内容：20

3. 同时开启故障快恢和断点续训时需满足以下条件：
   - 如果用户指定${user_converted_ckpt_path} 因故障快恢读取权重的优先级最高则训练过程的权重保存路径${output_dir}/saved_checkpoints（加载故障快恢路径） 必须为空，否则此参数无效断点续训失效。
   - 如果就是使用最新的训练权重进行断点续训（暂停+启动场景），那么可以同时指定train_auto_resume =1和 ${user_converted_ckpt_path}训练过程的权重保存路径，加载路径一致。

4.3.6 常见错误原因和解决方法

4.3.6.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size） 和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-32**进行设置。

- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

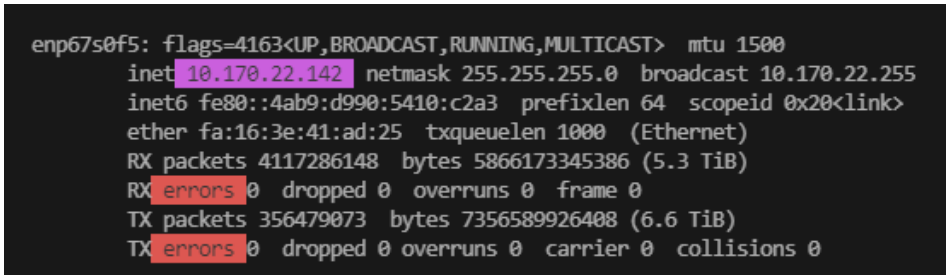
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.3.6.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-27 网卡名称错误



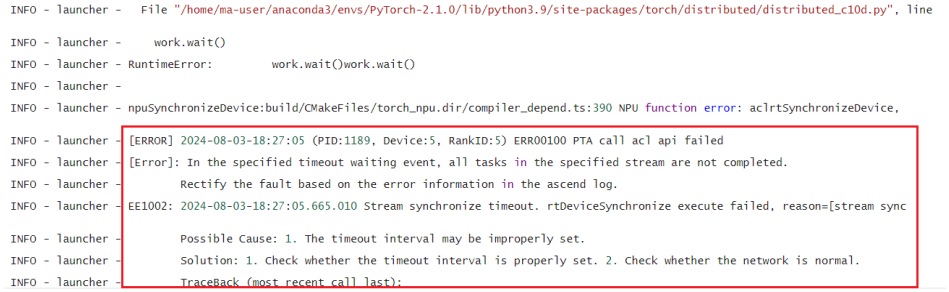

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.3.6.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-28 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.3.6.4 mc2 融合算子报错

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务时产生mc2融合算子错误。

图 4-29 mc2 融合算子错误

```
  File "/home/ma-user/AscendFactory/third-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
ird-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
    output, all_gather_grad_output = torch_npu.npu_all_gather_base_mm(
```

解决方法

修改代码文件：AscendFactory/scripts_modellink/{model_name}/3_training.sh文件，去除以下mc2融合算子--mc2

```
megatron_options="  \
    --tensor-model-parallel-size ${TP} \
    --pipeline-model-parallel-size ${PP} \
    --seed ${SEED} \
    --normalization RMSNorm \
    --position-embedding-type rope \
    --transformer-impl local \
    --sequence-parallel \
    --use-flash-attn \
    --bf16 \
    --swiglu \
    --use-fused-swiglu \
    --use-fused-rmsnorm \
    --use-distributed-optimizer \
    --use-fused-rotary-pos-emb \
    --use-rotary-position-embeddings \
    --use-mc2 \
    --no-masked-softmax-fusion \
```

4.4 主流开源大模型基于 Lite Cluster 适配 ModelLink PyTorch NPU 训练指导（6.3.912）

4.4.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Cluster上的训练方案。训练框架使用的是ModelLink。

本方案目前仅适用于企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts6.3.912版本，请参考**表4-35**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Cluster。
- 本文档中的**CCE集群版本选择v1.27~1.28**。版本使用的容器引擎为**Containerd**。
- 镜像适配的Cann版本是cann_8.0.rc3，驱动版本是23.0.6。
- 确保集群可以访问公网。

文档更新内容

6.3.912版本相对于6.3.911版本新增如下内容：

1. 代码结构发生变化，统一modellink和llama-factory的启动方式。
2. 继承911版本启动方式以外新加新的启动方式：
   ```
   ascendfactory-cli train xx.yaml model-name exp_name
   ```
3. 相对于6.3.911版本仅是使用run_type来指定训练的类型，只能区分 预训练、全参微调和lora微调但实际上预训练和sft是训练的不同阶段，全参、lora是训练参数设置方式。为了更加明确的区分不同策略，以及和llama-factory对齐，6.3.912版本调整以下参数：
   - 新增 STAGE，表示训练的阶段，可以选择的参数包括: {pt, sft}.
   - 新增 FINETUNING_TYPE，表示微调的策略，可以选择的参数包括：{full, lora}
   - 删除 RUN_TYPE

   所以当前的组合情况为：

项目	full	lora
pt（预训练）	√	√
sft（指令微调）	√	√

训练支持的模型列表

本方案支持以下模型的训练，如**表4-33**所示。

表 4-33 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLM v3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan 2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

表 4-34 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite k8s Cluster运行的，需要购买并开通k8s Cluster资源。
	准备代码	准备AscendFactory训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
训练	预训练/微调	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.4.2 准备工作

4.4.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Cluster。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-40**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**k8s Cluster资源购买**，熟悉集群资源开通流程，再开始操作购买Cluster资源。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。

- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

k8s Cluster 资源配置

若已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

其中k8s Cluster的容器中挂载存储支持OBS、SFS Turbo等方案进行挂载。例如OBS支持静态挂载和动态挂载，而SFS Turbo仅支持静态挂载，详细的挂载操作流程可阅读**通过静态存储卷使用已有极速文件存储**和**通过动态存储卷使用对象存储**。

kubectl 访问集群配置

本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的 CCE 集群控制版面中。根据**图4-30**的步骤进行操作，单击kubectl配置时，会弹出**图4-31**步骤页面。

图 4-30 配置中心

2. 根据图4-31，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-31 kubectl 访问集群配置

3. 在节点机器中，输入命令，查看Kubernetes集群信息。若显示如图图4-32的内容，则配置成功。

```
kubectl cluster-info
```

图 4-32 查看 Kubernetes 集群信息正确弹出内容

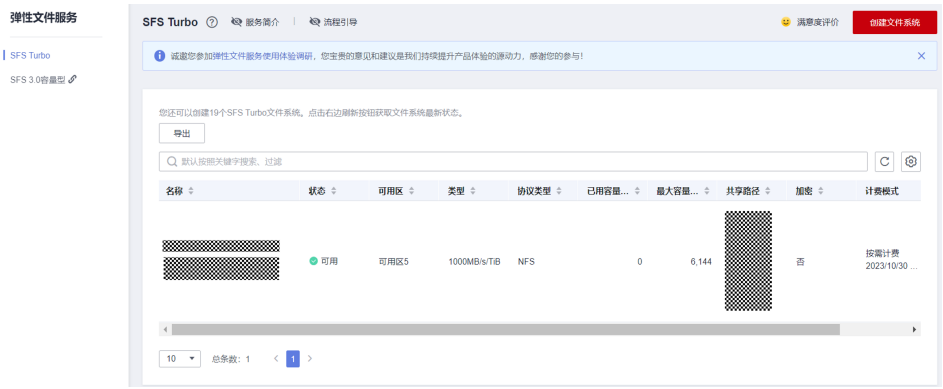
创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。

图 4-33 创建 SFS Turbo



2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

   详细操作指导请参考**创建SFS Turbo文件系统**。

   其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-34 SFS 类型和容量选择

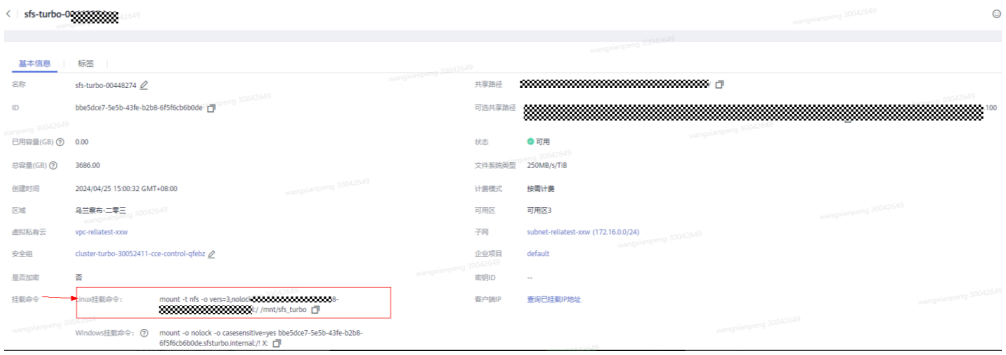
CCE 集群关联 SFS Turbo

进入已购买创建的CCE集群，选择存储，随后单击“创建存储卷声明PVC”。

- 选择“极速文件存储”，随后输入PVC名称。
- 选择“新建存储卷PV”，并单击“选择极速文件存储”。
- 进入选择页面，选择已经创建好的SFS Turbo，最后输入PV名称。

接下来需要通过访问集群节点，挂载SFS Turbo。

- 可通过ssh登录CCE集群中的某个节点（ssh使用的是eip地址）。
- 创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo
- SFS Turbo存储手动挂载到安装节点中，挂载命令如下截图：
- 挂载完成后，可通过以下步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。



4.4.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-35**所示，模型列表、对应的开源权重获取地址如**表4-33**所示。

表 4-35 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

获取对应模型的权重文件，获取链接参考**表4-33**。

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  若要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.912中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_train                  # 模型训练代码包
    ├──AscendFactory
      ├──examples/                # config配置文件目录
      ├──data.tgz                 # 样例数据压缩包
      ├──third-party/             # patch包
      ├──src/acs_train_solution/  # 训练运行包
      ├──intall.sh                # 需要的依赖包
      ├──scripts_llamafactory/    # llamafactory兼容旧版本启动方式目录
      ├──scripts_modellink/       # modelLink兼容旧版本启动方式目录
      ├──Dockerfile
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}
|──llm_train                                # 模型训练代码包
   |──AscendFactory
      |──config/                           # 配置文件
         |──deepspeed/                        # deepspeed配置json文件
         |──modellink_performance_cfgs.yaml  # ModelLink训练配置json文件
         |──.......
      |──data.tgz                  #样例数据压缩包
      |──intall.sh                 # 需要的依赖包
      |──scripts_modellink/        # modelLink兼容旧版本启动方式目录
         |──llama3                 # llama3系列模型执行脚本的文件夹
         |──qwen2.5                # Qwen2.5系列模型执行脚本的文件夹
         |── ...
         |── dev_pipeline.sh       # 系列模型共同调用的多功能的脚本
      |──third-party/              # patch包
      |──src/acs_train_solution/   # 训练运行包
         |──ascendcloud_patch/     # patch补丁包
         |──benchmark/             #工具包，存放数据集及基线数据
            |──trainer.py          # 训练启动脚本
            |──performance.py      # benchmark训练性能比较启动脚本
            |──accuracy.py         # benchmark训练精度启动脚本
   |──model/Qwen2-7B/                # 权重词表文件目录，如Qwen2-7B
   |──training_data                  # 原始数据目录
      |──alpaca_gpt4_data.json    # 微调数据
      |──train-00000-of-00001-a09b74b3ef9c3b56.parquet #预训练数据
   |──{output_dir} #{OUTPUT_SAVE_DIR}或yaml文件{output_dir}参数设置值
      # 自动生成数据目录结构
      |── preprocessed_data
      |──converted_hf2mg_weight_TP${TP}PP${PP}
      |──checkpoint    # 训练完成生成目录Qwen2-7B，自动生成
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录服务器。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如SFS Turbo的路径：/mnt/sfs_turbo目录下，以下都以/mnt/sfs_turbo为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

   ⚠ 注意

   Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务如产生mc2融合算子错误，可参考**mc2融合算子报错**

3. 上传tokenizers文件到工作目录中的/mnt/sfs_turbo/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/mnt/sfs_turbo，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /mnt/sfs_turbo
   mkdir -p models/Llama2-70B
   ```

4.4.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- 微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注他们自己
  ```

```
和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

若用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

  ```
  1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
  2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
  (随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
  3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
  (随机选择 3个数据作为测试集)
  ```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，若同时输入，则优先使用test_count，若都未输入，则返回处理失败 False。

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

  ```
  [
    {
      "instruction": "人类指令（必填）",
  ```

```
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据到指定目录

将下载的原始数据存放在/mnt/sfs_turbo/training_data目录下。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

```
mkdir training_data
```

数据存放参考目录结构如下：

```
${workdir}
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.4.2.4 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-36 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527

表 4-37 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.3.1

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v   # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择containerd作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

a. 下载并解压buildkit程序。

```
# 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

# 创建解压的目录
mkdir /usr/local/buildkit

# 解压到指定的目录
tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit

# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

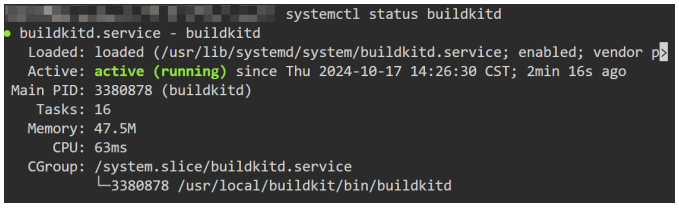
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

containerd 容器引擎有命名空间的概念。Kubernetes 下使用的 containerd 默认命名空间是 k8s.io。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。
  ```
  ctr -n k8s.io pull {image_url}
  ```
- 使用nerdctl工具拉取镜像。
  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

⚠ 注意

集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

步骤三 构建 ModelArts Lite 训练镜像

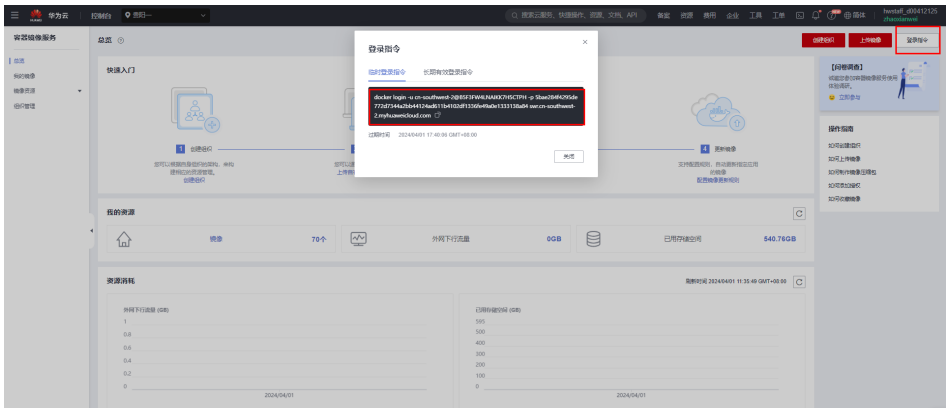
获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-35**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.912-xxx.zip，并直接进入到llm_train/AscendFactory文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendFactory
   ```
2. 编辑llm_train/AscendFactory中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。
   ```
   nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称>  .
   ```
   nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以nerdctl pull 拉取测试以下镜像是否能拉取成功。
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤四 在节点机器中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。

图 4-35 复制登录指令



由于使用的容器引擎是containerd，不再是docker，因此需要改写复制的登录指令，将docker进行替换，使用nerdctl工具。

```
# docker login 替换为：
nerdctl login
```

步骤五 修改并上传镜像

1. 在机器中输入Step4登录指令后，使用下列示例命令将镜像上传至SWR：

```
nerdctl --namespace k8s.io tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在**步骤三 构建ModelArts Lite训练镜像**中使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_3_ascend:20240606

示例：

nerdctl --namespace k8s.io tag ${dockerfile_image_name} swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_3_ascend:20240606

2. 上传镜像至镜像仓库。

```
nerdctl --namespace k8s.io push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

nerdctl --namespace k8s.io push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_3_ascend:20240606

步骤六 编写 Config.yaml 文件

k8s有两种方式来管理对象：

- 命令式，即通过Kubectl指令直接操作对象。
- 声明式，通过定义资源YAML格式的文件来操作对象。

首先给出单个节点训练的config.yaml文件模板，用于配置pod。而在训练中，需要按照参数说明修改**${}**中的参数值。该模板使用SFS Turbo挂载方案。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob              # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                     # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980     # 保持不动
data:                                    # data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                            # job名字，需要和configmap中名字保持联系
  namespace: default                     # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano                 # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}           # 镜像地址
          imagePullPolicy: IfNotPresent  # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                       # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
          env:
          - name: name
            valueFrom:
```

```
        fieldRef:
          fieldPath: metadata.name
    - name: ip
      valueFrom:
        fieldRef:
          fieldPath: status.hostIP
    - name: framework
      value: "PyTorch"
    command: ["/bin/sh", "-c"]
    args:
    - ${command}
    resources:
      requests:
        huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
        memory: ${requests_memory}                 # 容器请求的最小内存
        cpu: ${requests_cpu}                     # 容器请求的最小 CPU
      limits:
        huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
        memory: ${limits_memory}                  # 容器可使用的最大内存
        cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
    volumeMounts:                             # 容器内部映射路径
    - name: shared-memory-volume
      mountPath: /dev/shm
    - name: ascend-driver                     # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons                    # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                              # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                           # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                    # 物理机外部路径
  - name: shared-memory-volume                      # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
```

```
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
```

双个节点训练的config.yaml文件模板，用于实现双机分布式训练。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob     # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                  # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
data:                    #data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                         # job名字，需要和configmap中名字保持联系
  namespace: default                  # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano               # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                  # 采用宿主机网络模式
        containers:
        - image: ${image_name}         # 镜像地址
          imagePullPolicy: IfNotPresent      # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：
每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                     # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
```

```
    env:
    - name: name
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: ip
      valueFrom:
        fieldRef:
          fieldPath: status.hostIP
    - name: framework
      value: "PyTorch"
    command: ["/bin/sh", "-c"]
    args:
    - ${command}
    resources:
      requests:
        huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
        memory: ${requests_memory}                # 容器请求的最小内存
        cpu: ${requests_cpu}                    # 容器请求的最小 CPU
      limits:
        huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
        memory: ${limits_memory}                  # 容器可使用的最大内存
        cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
    volumeMounts:                             # 容器内部映射路径
    - name: shared-memory-volume
      mountPath: /dev/shm
    - name: ascend-driver                     # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons                    # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                              # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                           # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                    # 物理机外部路径
  - name: shared-memory-volume                        # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
```

```
        path: /usr/slog
    - name: sfs-volume
      persistentVolumeClaim:
        claimName: ${pvc_name}    #已创建的PVC名称
    restartPolicy: OnFailure
- name: work
  replicas: 1
  template:
    metadata:
      name: training
      labels:
        app: ascendspeed
        ring-controller.cce: ascend-1980  # 保持不动
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: volcano.sh/job-name
                    operator: In
                    values:
                      - vcjob
              topologyKey: kubernetes.io/hostname
      hostNetwork: true                    # 采用宿主机网络模式
      containers:
      - image: ${image_name}         # 镜像地址
        imagePullPolicy: IfNotPresent       # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
        name: ${container_name}
        securityContext:                        # 容器内 root 权限
          allowPrivilegeEscalation: false
          runAsUser: 0
        env:
        - name: name
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ip
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: framework
          value: "PyTorch"
        command: ["/bin/sh", "-c"]
        args:
        - ${command}
        resources:
          requests:
            huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
            memory: ${requests_memory}                # 容器请求的最小内存
            cpu: ${requests_cpu}                    # 容器请求的最小 CPU
          limits:
            huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
            memory: ${limits_memory}                 # 容器可使用的最大内存
            cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
        volumeMounts:                           # 容器内部映射路径
        - name: shared-memory-volume
          mountPath: /dev/shm
        - name: ascend-driver                   # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons                  # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: localtime
          mountPath: /etc/localtime
        - name: hccn                            # 驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: npu-smi                         # npu-smi
          mountPath: /usr/local/sbin/npu-smi
```

```
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                # 物理机外部路径
  - name: shared-memory-volume                    # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
      claimName: ${pvc_name}     #已创建的PVC名称
  restartPolicy: OnFailure
```

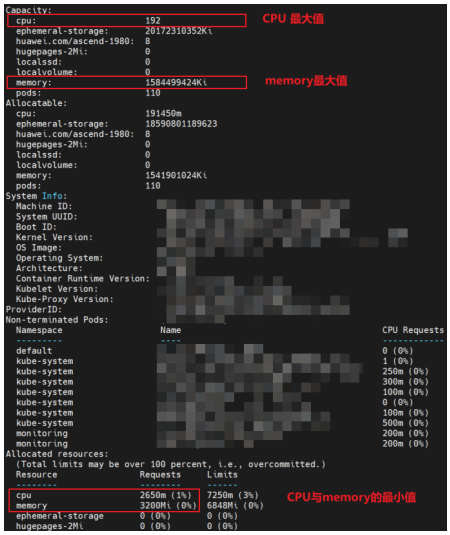
参数说明：

- ${container_name} 容器名称，此处可以自己定义一个容器名称，例如ascendspeed。
- ${image_name} 为**步骤五 修改并上传镜像**中，上传至SWR上的镜像链接。
- ${command} 使用config.yaml文件创建pod后，在容器内自动运行的命令。在进行训练任务中会给出替换命令。
- /mnt/sfs_turbo 为宿主机中默认挂载SFS Turbo的工作目录，目录下存放着训练所需代码、数据等文件。
  - 同样，/mnt/sfs_turbo 也可以映射至容器中，作为容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。为方便访问两个地址可以相同。
- ${pvc_name} 为在**CCE集群关联SFS Turbo**步骤中创建的PVC名称。
- 在设置容器中需要的CPU与内存大小时，可通过运行以下命令查看申请的节点机器中具体的CPU与内存信息。

  ```
  kubectl describe node
  ```

  - ${requests_cpu} 指在容器中请求的最小CPU核心数量，可使用Requests中的值，例如2650m。
  - ${requests_memory} 指在容器中请求的最小内存空间大小，可使用Requests中的值，例如3200Mi。
  - ${limits_cpu} 指在容器中可使用的最大CPU核心数量，例如192。

- ${limits_memory} 指在容器中可使用的最大内存空间大小，例如换算成1500Gi。



4.4.3 训练任务

4.4.3.1 执行训练任务（推荐）

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

步骤二 修改训练 Yaml 配置文件

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器（默认开启）	断点续训：加载权重+优化器
• pt • sft	ckpt_load_type=0	ckpt_load_type=1 user_converted_ckpt_path=xxx，(转换MG的权重)	ckpt_load_type=2 user_converted_ckpt_path=xxx（训练输出MG权重）

修改、添加**代码目录**下modellink_performance_cfgs.yaml文件参数配置，样例yaml配置文件结构如下：

- base块：基础配置块，主要为公共配置参数
- ModelName块：该模型所需配置的参数，如qwen2.5-7b块
- exp_name：实验块，训练策略-序列长度所需参数配置

样例yaml文件仅展示常用实验配置，如需其他配置需根据样例自行添加。

根据以下步骤修改yaml文件：

1. 数据集选择:dataset和processed_data_dir参数二选一，详解如下：

参数	示例值	参数说明
dataset	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	训练时指定的输入数据路径。请根据实际规划修改。用户根据训练情况二选一；
processed_data_dir	/home/ma-user/ws/xxx	已处理好数据路径目录，如有处理完成数据可设置此参数

2. 权重文件、输出目录及其他重要参数设置,详解如下：

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B	【**必修改**】加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
output_dir	/home/ma-user/ws/save_dir	【**必修改**】训练任务结束生成日志及权重文件目录
scirpts_dir	/home/ma-user/ws/llm_train/AscendFactory/scripts_modellink	【**必修改**】ModelLink脚本相对或绝对路径，用于方便加载脚本

参数	示例值	参数说明
ckpt_load_type	1	【**可选**】默认为1 • 0，不加载权重 • 1，加载权重不加载优化器状态【**增量训练**】 • 2，加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
user_converted_ckpt_path	/home/ma-user/ws/xxx	【**可选**】已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。 • 增量训练：转换Megatron权重，如不指定默认为${output_dir}/converted_hf2mg_weight_TP{tp}PP{pp}目录。 • 断点续训：训练过程中保存的某个权重，可详见**断点续训和故障快恢说明**
train_auto_resume	false	【**可选**】是否开启【**故障快恢**】功能，【true、false】默认false不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。详见**断点续训和故障快恢说明**
handler-name	• GeneralPretrainHandler • GeneralInstructionHandler • MOSSInstructionHandler • AlpacaStyleInstructionHandler • SharegptStyleInstructionHandler	示例值需要根据数据集${dataset}的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSInstructionHandler：使用微调的moss数据集 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集

3. 其他参数设置，详解如下：

参数	示例值	参数说明
stage	pt	表示训练类型。可选择值： ● pt：预训练 ● sft：指令微调
finetuning_type	full	表示训练策略。可选择值： ● full：全参微调 ● lora：lora微调
prompt-type	qwen	数据模板，当使用LLama-Factory模板数据时需指定此参数； handler-name为【 AlpacaStyleInstructionHandler,SharegptStyleInstructionHandler 】需指定。
micro-batch-size	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
global-batch-size	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
tensor-model-parallel-size	8	表示张量并行。
pipeline-model-parallel-size	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
context-parallel-size	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加此值（≥ 2）。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
lr	2.5e-5	学习率设置。
min-lr	2.5e-6	最小学习率设置。
seq-length	4096	要处理的最大序列长度。
convert_mg2hf_at_last	true	是否Megatron格式权重转换为HuggFace格式权重，默认true【 true或false 】
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
train-iters	10	非必填。表示训练step迭代次数，有默认值
seed	1234	随机种子数。每次数据采样时，保持一致。

参数	示例值	参数说明
save-interval	1000	用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
save-total-limit	-1	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 修改 config.yaml 中的${command}

请根据**步骤二 修改训练Yaml配置文件**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>

apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
            ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>
  - name: work1
    template:
      ...
      spec:
        ...
```

```
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendFactory;
          ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr>
--num_nodes <nodes> --rank <rank>
  - name: work2
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
            ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr>
--num_nodes <nodes> --rank <rank>
  - name: work3
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
            ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr>
--num_nodes <nodes> --rank <rank>
```

- <cfgs_yaml_file>：ModelLink配置yaml文件地址，如**代码目录**中modellink_performance_cfgs.yaml相对或绝对路径，根据自己要求执行
- <model_name>：训练模型名，如qwen1.5-7b，需与<cfgs_yaml_file>里面对应
- <exp_name>：实验名称：指定本次实验的具体配置，还包括数据配置等，比如full, lora等，该名称需要和<cfgs_yaml_file>里面对应。
- --master_addr <master_addr>：主master节点IP，一般选rank0为主master。
- --num_nodes <nodes>：训练节点总个数
- --rank <rank>：节点ID,从0开始，一般选rank0为主master。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr
<master_addr> --num_nodes <nodes> --rank <rank>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
```

- <cfgs_yaml_file>：ModelLink配置yaml文件地址，如**代码目录**中modellink_performance_cfgs.yaml相对或绝对路径，根据自己要求执行
- <model_name>：训练模型名，如qwen1.5-7b，需与<cfgs_yaml_file>里面对应
- <exp_name>：实验名称：指定本次实验的具体配置，还包括数据配置等，比如full, lora等，该名称需要和<cfgs_yaml_file>里面对应。

ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>

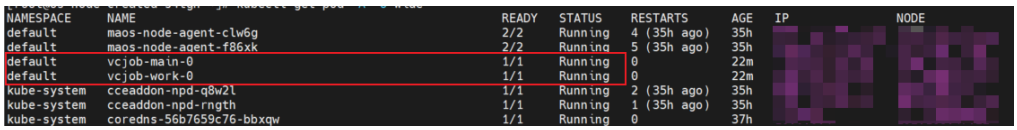
步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```



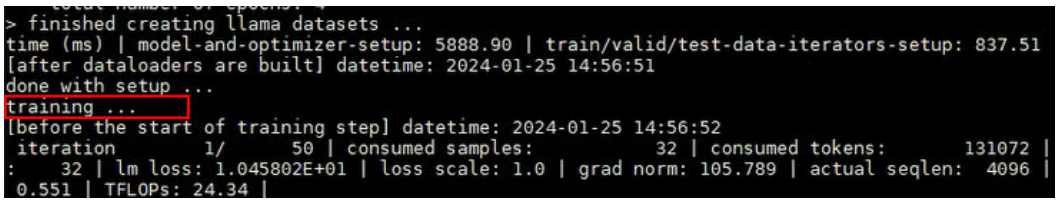
若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-36 等待模型载入



训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.4.3.2 执行训练任务（历史版本）

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

步骤二 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-38**所示。其他超参均有默认值，可以参考**表4-39**按照实际需求修改。

表 4-38 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendFactory/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendFactory/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendFactory/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/AscendFactory/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/AscendFactory/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/AscendFactory/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/AscendFactory/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/AscendFactory/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 修改 config.yaml 中的${command}

请根据**步骤二 修改训练超参配置**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts_modellink/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
<NNODES=4> <NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
  - name: work1
    template:
      ...
      spec:
```

```
      ...
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendFactory;
          sh scripts_modellink/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
- name: work2
  template:
    ...
    spec:
      ...
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendFactory;
          sh scripts_modellink/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
- name: work3
  template:
    ...
    spec:
      ...
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendFactory;
          sh scripts_modellink/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts_modellink/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost>
<NNODES=1> <NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendFactory;
            sh scripts_modellink/llama2/0_pl_pretrain_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g               2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk               2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                        1/1     Running   0             22m
default       vcjob-work-0                        1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                  1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                  1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw            1/1     Running   0             37h
```

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-37 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.4.4 查看日志和性能

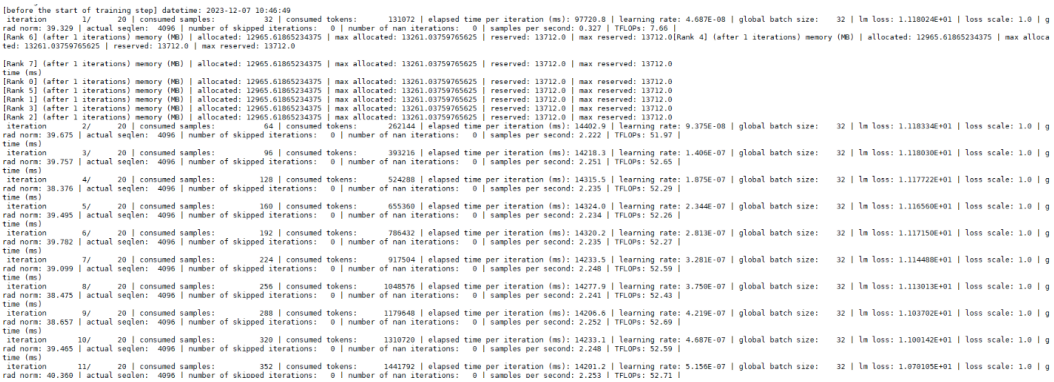
查看日志

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

训练过程中，训练日志会在最后的Rank节点打印。

图 4-38 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

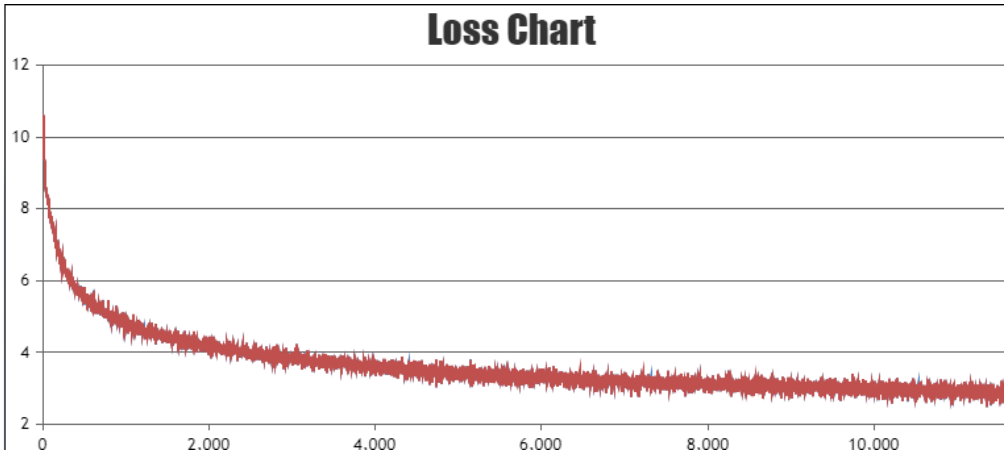
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-39**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-39**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-39 Loss 收敛情况（示意图）



4.4.5 训练脚本说明参考

4.4.5.1 训练参数配置说明【旧】

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，请根据实际模型修改。

表 4-39 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	**【必改】**训练时指定的输入原始数据路径。请根据实际规划修改。用户根据训练情况二选一；
USER_PROCESSED_DATA_DIR	/home/ma-user/ws/process_data	**【可选】**如已有预处理完成数据可指定此目录，训练过程中会优先加载此目录，跳过数据预处理过程；默认无此参数。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B	**【必改】**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
OUTPUT_SAVE_DIR	/home/ma-user/ws/save_dir/llama2-70B_sft_lora_4096	**【必改】**。训练任务结束生成日志及权重文件目录。根据实际情况决定
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
STAGE	pt	表示当前的训练阶段。可选择值：【pt、sft】 • sft：代表监督微调； • pt：代表预训练；
FINETUNING_TYPE	full	表示训练策略。可选择值【full、lora】： • full：全参微调 • lora：lora微调

参数	示例值	参数说明
DATA_TYPE	【GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler】	【**必改**】示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSInstructionHandler：使用微调的moss数据集。 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。

参数	示例值	参数说明
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	10	非必填。表示训练step迭代次数，会进行自动计算得出。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SN	5120	指定的输入数据集中数据的总数量。更换数据集时，需要修改。
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
MA_TRAIN_AUTO_RESUME	False	**【可选】【故障快恢】**是否开启此功能，【True、False】默认False不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。详见**断点续训和故障快恢说明**

参数	示例值	参数说明
CKPT_LOAD_TYPE	1	**可选**【0、1、2】，默认为1 • 0： 不加载权重 • 1：加载权重不加载优化器状态【增量训练】 • 2：加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
USER_CONVERTED_CKPT_PATH	/home/ma-user/ws/xxx	【**可选**】已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。 • 增量训练：转换Megatron权重，如不指定默认为${output_dir}/converted_hf2mg_weight_TP{tp}PP{pp}目录。 • 断点续训：训练过程中保存的某个权重，详见**断点续训和故障快恢说明**

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-40**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-40 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点&4*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点&4*Ascend
22	mistral	mistral-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	full	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
28		qwen2.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

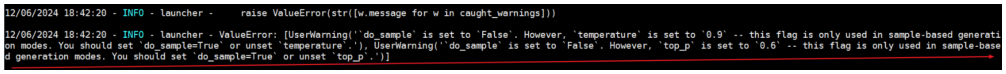
序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.4.5.2 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.46.1），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendFactory/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-40所示。

图 4-40 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-41所示。

图 4-41 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-42 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321 
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327 
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-43所示。

图 4-43 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295 
```

图 4-44 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-45所示。

图 4-45 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.4.5.3 断点续训和故障快恢说明

相同点

断点续训（Checkpointing）和故障快恢都是指训练中断后可从训练中一定间隔（${save-interval}）保存的模型（包括模型参数、优化器状态、训练迭代次数等）继续训练恢复，而不需要从头开始。

不同点

- 断点续训：可指定加载训练过程中生成的Megatron格式权重(${user_converted_ckpt_path})
- 故障快恢：默认加载${output_dir}/saved_checkpoints中最大迭代次数（iter_000xxxx）Megatron格式权重文件。

⚠ 注意

1. lora微调不支持断点续训
2. 启动前需检查latest_checkpointed_iteration.txt文件中内容是否与所需iter_000xxxx数字（表示训练后保存权重对应迭代次数）保持一致，不一致则修改latest_checkpointed_iteration.txt内容与iter_000xxxx保持一致。

```
|——${saved_checkpoints}
  |——iter_0000010
  |——iter_0000020
  |———latest_checkpointed_iteration.txt
```

示例，latest_checkpointed_iteration.txt文件内容：20

3. 同时开启故障快恢和断点续训时需满足以下条件：
   - 如果用户指定${user_converted_ckpt_path} 因故障快恢读取权重的优先级最高则训练过程的权重保存路径${output_dir}/saved_checkpoints（加载故障快恢路径） 必须为空，否则此参数无效断点续训失效。
   - 如果就是使用最新的训练权重进行断点续训（暂停+启动场景），那么可以同时指定train_auto_resume =1和 ${user_converted_ckpt_path}训练过程的权重保存路径，加载路径一致。

4.4.6 常见错误原因和解决方法

4.4.6.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size） 和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-40**进行设置。

- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

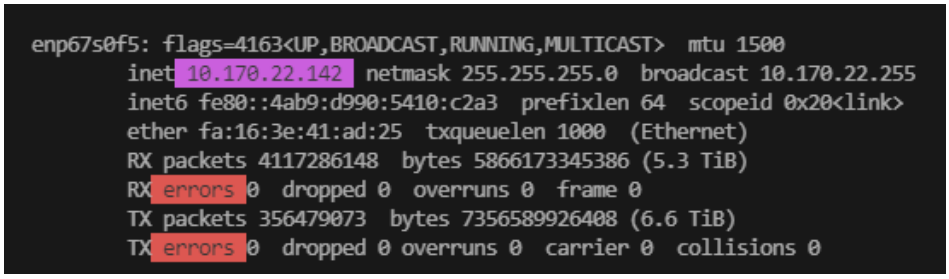
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.4.6.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-46 网卡名称错误




```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.4.6.3 工作负载 Pod 异常

Pod 状态为 Pending

当Pod状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。具体参考链接为**工作负载状态异常定位方法**。

通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。

```
Events:
  Type     Reason            Age   From     Message
  ----     ------            ----  ----     -------
  Warning  FailedScheduling  26s   volcano  0/2 nodes are unavailable: 2 GPU topology evaluator provider davinci assignment gives no assignment.
```

解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```
3. 若重启后，还是会Pending，建议多重复重启几次。

```
kube-system   volcano-admission-7c8f9fb8f5-6k8k4          1/1   Running   0             35h   <none>   <none>
kube-system   volcano-admission-7c8f9fb8f5-xvt8d          1/1   Running   3 (35h ago)   35h   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-rc8c4         1/1   Running   0             35h   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-vmz75         1/1   Running   1 (35h ago)   35h   <none>   <none>
kube-system   volcano-scheduler-c48c5c87-6kp48            1/1   Running   0             154m  <none>   <none>
kube-system   volcano-scheduler-c48c5c87-pqsh7            1/1   Running   0             175m  <none>   <none>
monitoring    kube-state-metrics-5b5498fccb-tqpzt         1/1   Running   3 (35h ago)   37h   <none>   <none>
monitoring    log-agent-fluent-bit-ls6fp                  2/2   Running   0             35h   <none>   <none>
monitoring    log-agent-fluent-bit-pb7tv                  2/2   Running   0             35h   <none>   <none>
```

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

如何删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.4.6.4 mc2 融合算子报错

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务时产生mc2融合算子错误。

图 4-47 mc2 融合算子错误

```
  File "/home/ma-user/AscendFactory/third-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
ird-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
    output, all_gather_grad_output = torch_npu.npu_all_gather_base_mm(
```

解决方法

修改代码文件：AscendFactory/scripts_modellink/{model_name}/3_training.sh文件，去除以下mc2融合算子--mc2

```
megatron_options="  \
    --tensor-model-parallel-size ${TP} \
    --pipeline-model-parallel-size ${PP} \
    --seed ${SEED} \
    --normalization RMSNorm \
    --position-embedding-type rope \
    --transformer-impl local \
    --sequence-parallel \
    --use-flash-attn \
    --bf16 \
    --swiglu \
    --use-fused-swiglu \
    --use-fused-rmsnorm \
    --use-distributed-optimizer \
    --use-fused-rotary-pos-emb \
    --use-rotary-position-embeddings \
    --use-mc2 \
    --no-masked-softmax-fusion \
```

4.5 主流开源大模型基于 Standard+OBS 适配 ModelLink PyTorch NPU 训练指导（6.3.912）

4.5.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.912版本相对于6.3.911版本新增如下内容：

1. 代码结构发生变化，统一了modellink和llama-factory的启动方式。
2. 继承911版本启动方式以外增加新的启动方式：
   ```
   ascendfactory-cli train xx.yaml model-name exp_name
   ```
3. 相对于6.3.911版本仅是使用run_type来指定训练的类型，只能区分 预训练、全参微调和lora微调但实际上预训练和sft是训练的不同阶段，全参、lora是训练参数设

置方式。为了更加明确的区分不同策略，以及和llama-factory对齐，6.3.912版本调整以下参数：

- 新增 STAGE，表示训练的阶段，可以选择的参数包括: {pt, sft}.
- 新增 FINETUNING_TYPE，表示微调的策略，可以选择的参数包括：{full, lora}
- 删除 RUN_TYPE

所以当前的组合情况为：

项目	full	lora
pt（预训练）	√	√
sft（指令微调）	√	√

4. 支持多种权重加载方式：
   - 不加载权重
   - 增量训练：加载权重，不加载优化器
   - 断点续训：加载权重+优化器，可自由指定训练输出目录下批次的权重
   - 故障快恢：加载权重+优化器，默认加载训练输出目录下最新的权重

支持的模型列表

本方案支持以下模型的训练，如表4-41所示。

表 4-41 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

表 4-42 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendFactory训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook（可选）	如果用户有自定义开发的需要，比如查看和编辑代码、数据预处理、权重转换等操作，可通过Notebook环境进行，并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。
训练	预训练/微调	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.5.2 准备工作

4.5.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-47**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.5.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- 微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

使用标准的.json或jsonl格式的数据，数据集中的每个样本包含的标签保持一致，且必须包含text标签。实际训练过程中只会读取该字段。

```
[
{
   'text': 'April is the fourth month...'
}
]
```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

```
{
  "conversation_id": 1,
  "meta_instruction": "",
  "num_turns": 3,
  "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。

- value：具体的对话内容。
- system：系统提示词，用来为整个对话设定场景或提供指导原则。
- tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据集至 OBS

1. 准备数据集，例如下载样例数据集或者在本地按照固定格式处理好自己的数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.5.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-41**。

   权重文件下载有如下几种方式，但不仅限于以下方式：

   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

     ```
     huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
     ```

     如果要下载指定版本的模型文件，则命令如下：

     ```
     huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
     ```

   - **方法三**：使用专用多线程下载器 hfd：**hfd** 是 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。

- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 利用**OBS-Browser+**工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.5.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-43**所示。

表 4-43 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

- AscendCloud-6.3.912代码包中AscendCloud-LLM代码包结构如下：

```
├──AscendCloud-LLM
 ├──llm_train                  # 模型训练代码包
  ├──AscendFactory
   ├──examples/config          # config配置文件目录
```

```
    |——modellink_performance_cfgs.yaml # modellink配置最优参数yaml文件
  |——data.tgz                # 样例数据压缩包
  |——third-party/            # 三方依赖的代码仓，和补丁patch包
  |——src/acs_train_solution/  # 训练运行包
  |——intall.sh               # 安装脚本
  |——dependences.yaml         # 需要的三方依赖包的版本和下载地址
  |——scripts_llamafactory/    # llamafactory兼容旧版本启动方式目录
  |——scripts_modellink/       # modelLink兼容旧版本启动方式目录
  |——Dockerfile
```

- AscendFactory/examples/config配置文件目录：

```
|——AscendFactory/examples/config/  # config配置文件
  |——modellink_performance_cfgs.yaml # modellink配置最优参数yaml文件
  |——performance_cfgs.yaml        # 微调性能配置yaml文件
  |——llama_factory_performance_cfgs_VL.yaml  # qwen2vl微调yaml配置文件
  |——accuracy_cfgs.yaml           # 训练精度配置yaml文件
  |——llama_factory_cfgs_posttrain.yaml  # RM、PPO、DPO训练阶段样例yaml文件
  |——llama_factory_performance_baseline.yaml # 性能基线配置
  |——llama_factory_accuracy_baseline.yaml    # 精度基线配置
```

  该目录下主要放置性能、精度任务的yaml配置文件，包含性能基线、精度基线、训练最佳实践参数等，以上配置文件仅供参考。

代码上传至 OBS

本地完成代码包AscendCloud-LLM-xxx.zip的解压，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|——llm_train                    # 模型训练代码包
   |——AscendFactory
|——model/Qwen2-7B/                # 权重词表文件目录，如Qwen2-7B
|——training_data                 # 原始数据目录
# 训练输出目录路径：根据{OUTPUT_SAVE_DIR}或yaml文件{output_dir}参数设置
|——{output_dir}                  # 输出目录，以下目录在训练过程中自动生成
   |——converted_hf2mg_weight_TP${TP}PP${PP} # 训练过程Megatron格式权重
   |——converted_mg2hf_weight   # 训练完成转换为HF格式权重目录，
   |——logs                  # 训练过程日志
   |——preprocessed_data       # 训练过程预处理后数据集目录
   |——saved_checkpoints       # 训练生成权重文件
```

⚠ 注意

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务如产生mc2融合算子错误，可参考**mc2融合算子报错**

4.5.2.5 准备镜像

4.5.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-44 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	CANN：cann_8.0.rc3 PyTorch：2.3.1

镜像可选用两种方式：基础镜像、ECS中DockerFIle构建新镜像（二选一），详解如下：

- **基础镜像**：用户可在训练作业中直接选择基础镜像作为运行环境，但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行install.sh文件，来安装依赖以及下载完整代码。
- **ECS中DockerFIle构建新镜像**：在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会尝试自动下载三方依赖源码并安装依赖的pip包，并将以上源码打包至镜像环境中；

  训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

  ⚠ 注意

  在华为公有云平台申请的资源一般默认连通网络，如未连通网络或无法git clone下载代码时用户则需要找到已连通网络的机器（本章节以Linux系统机器为例）将下载完成的源码放置**代码目录**：AscendFactory/third-party下，命令如下：

  ```
  # 三方开源源码
  git clone https://gitee.com/ascend/MindSpeed.git
  git clone https://github.com/huggingface/transformers.git
  git clone https://github.com/NVIDIA/Megatron-LM.git
  git clone https://gitee.com/ascend/ModelLink.git
  ```

  以上任务完成后重新**上传代码至OBS**。

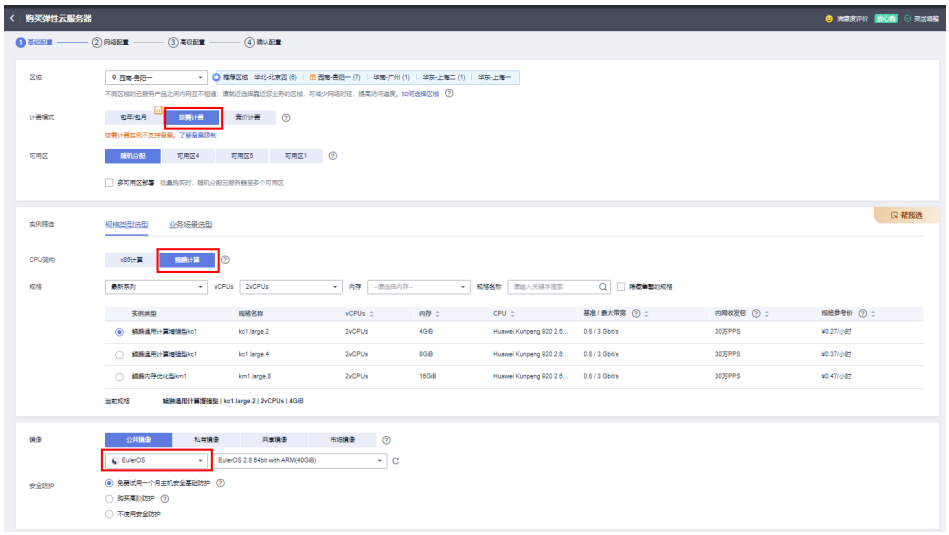
4.5.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

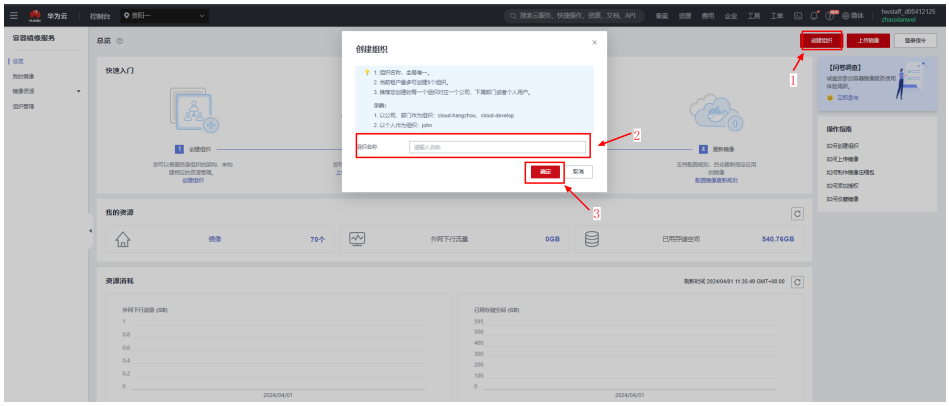
图 4-48 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-49 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

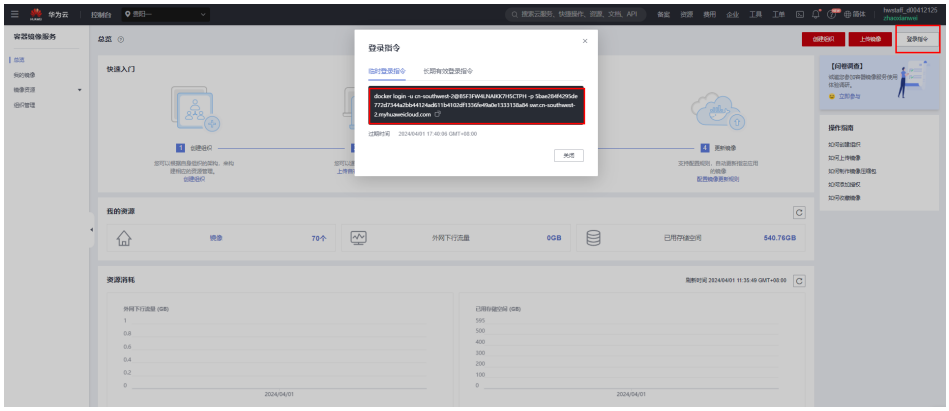
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-44**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-50 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_3_ascend:20241212

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_3_ascend:20241212

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_3_ascend:20241225

4.5.2.5.3 ECS 中构建新镜像（可选）

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

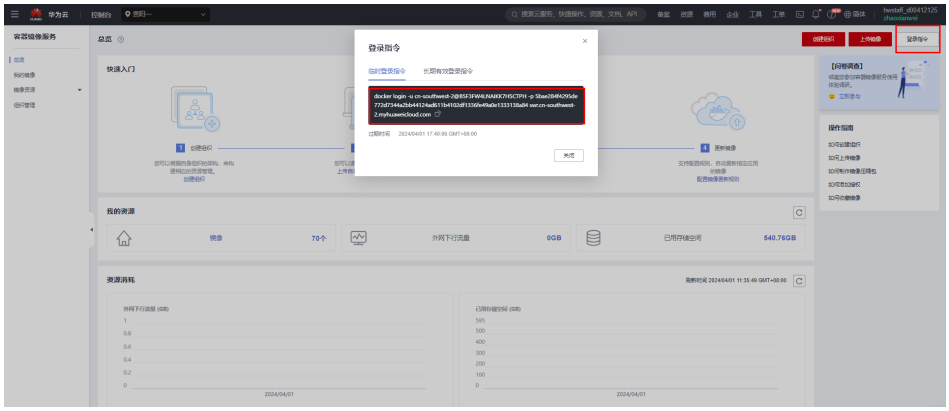
获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-43**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.912-xxx.zip，并直接进入到llm_train/AscendFactory文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendFactoryry中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host --build-arg install_type=modellink -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-51 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_3_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_3_ascend:20240606

2. 上传镜像至镜像仓库。

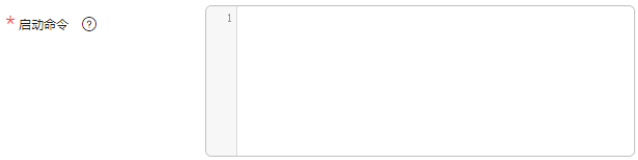
```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_3_ascend:20240606

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-52 训练作业启动命令



4.5.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中，如果用户有自定义开发的需要，比如查看和编辑代码、数据预处理、权重转换等操作，可通过Notebook环境进行，。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-53 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，如

果需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-54 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

注：修改完成之后，需要再使用mox.file.copy_parallel函数将数据copy回OBS中。

4.5.3 执行训练任务

4.5.3.1 执行训练任务（推荐）

新的训练方式将统一管理训练日志、训练结果和训练配置，使用yaml配置文件方便用户根据自己实际需求进行修改。推荐用户使用该方式进行训练。

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器（默认开启）	断点续训：加载权重+优化器
• pt • sft	ckpt_load_type=0	ckpt_load_type=1 user_converted_ckpt_path=xxx，	ckpt_load_type=2 user_converted_ckpt_path=xxx

前提条件

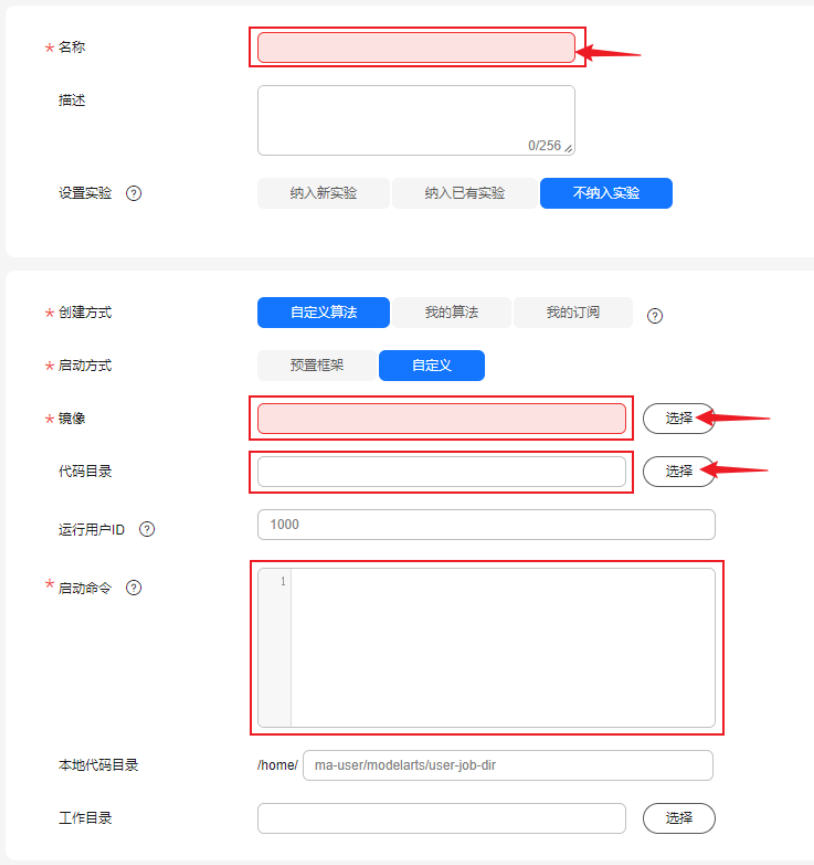
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径obs://<bucket_name>llm_train/AscendFactory代码目录。

图 4-55 创建训练作业



新的训练方式将统一管理训练日志、训练结果和训练配置，使用yaml配置文件方便用户根据自己实际需求进行修改。推荐用户使用该方式进行训练。

使用**基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendFactory;
sh install.sh modellink;
source /usr/local/Ascend/ascend-toolkit/set_env.sh;
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
```

使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendFactory;
source /usr/local/Ascend/ascend-toolkit/set_env.sh;
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
```

命令详解如下：

- <cfgs_yaml_file>：性能配置的yaml文件地址，如**代码目录**modellink_performance_cfgs.yaml相对或绝对路径。
- <model_name>：训练模型名，如qwen2-7b

- <exp_name>：实验名称，具体可以设置的值参考<cfgs_yaml_file>

样例配置截图如下：

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

输入指定的目录在训练开始时，平台会自动将指定的OBS路径下的文件copy到容器内

输出指定的目录在训练过程中，平台会自动将容器内的文件copy到指定的OBS路径下

1. 在“输入”框内设置超参配置：dataset、processed_data_dir、user_converted_ckpt_path、model_name_or_path根据实际要求选择，示例如下。

   输入**数据集**参数：是否使用已处理好数据集；

   - 是，设置以下超参
     - **processed_data_dir**:已处理好数据路径目录
   - 否，使用原始数据集，设置以下超参
     - **dataset**：训练时指定的输入原始数据集路径。

   输入**权重词表**超参：是否使用已转换Megatron格式权重或训练输出结果权重目录；

   - 是，设置以下超参

- **user_converted_ckpt_path**：已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。详见**断点续训和故障快恢说明**
- **model_name_or_path**：加载tokenizer与Hugging Face权重对应存放目录地址。

– 否，设置以下超参

- **model_name_or_path**：加载tokenizer与Hugging Face权重对应的存放地址

2. 在“输出”的输入框内设置超参：output_dir、hf_save_dir，根据实际要求选择，示例如下；
   – **output_dir**：训练完成后指定的输出模型路径。
   – **hf_save_dir**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保convert_mg2hf_at_last设置为True，默认为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。超参：dataset中则直接选中数据集文件，超参：processed_data_dir则需选中存放已处理好数据集的目录文件夹。
4. “输入”和“输出”中的获取方式全部选择为：超参。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置超参

单击“增加超参”，在增加的超参填写框中，按照**表4-45**表格中的配置进行填写。

图 4-56 超参

表 4-45 需要填写的超参

超参	示例值	参数说明
mount_type	OBS	【必填】。表示代码根据OBS存储方式运行。

超参	示例值	参数说明
scirpts_dir	/home/ma-user/modelarts/user-job-dirAscendFactory/scripts_modellink	【**必填**】模型转换数据、转换权重、训练脚本路径，scripts_modellink目录路径
train_auto_resume	False	是否开启【**故障快恢**】功能，【True、False】默认False不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。详见**断点续训和故障快恢说明**
stage	sft	【**必填**】表示当前的训练阶段。可选择值：【pt,sft】,默认sft • sft：代表监督微调； • pt：代表预训练；
ckpt_load_type	1	可选【0、1、2】，默认为1 • 0： 不加载权重 • 1：加载权重不加载优化器状态【**增量训练**】 • 2：加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
handler-name	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一，**默认**AlpacaStyleInstructionHandler。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
micro-batch-size	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
global-batch-size	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。

超参	示例值	参数说明
tensor-model-parallel-size	8	表示张量并行。
pipeline-model-parallel-size	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
context_parallel_size	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
lr	2.5e-5	学习率设置。
min-lr	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
train-iters	100	表示训练step迭代次数，根据实际需要修改。
save-interval	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
save-total-limit	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
seed	1234	随机种子数。每次数据采样时，保持一致。
convert_mg2hf_at_last	true	表示训练完成的权重文件会自动转换为Hugging Face格式权重。【true、false】默认为true

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

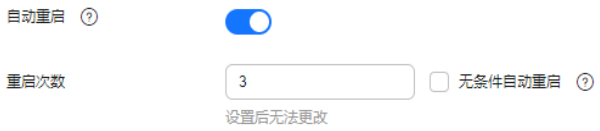
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-57 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格，设置超参train_auto_resume为True的前提下，默认为Fasle。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-47**进行配置。

图 4-58 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.5.3.2 执行训练任务（历史版本）

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器	断点续训：加载权重+优化器
• pt • sft	CKPT_LOAD_TYPE=0	CKPT_LOAD_TYPE=1 USER_CONVERTED_CKPT_PATH=xxx	CKPT_LOAD_TYPE=2 USER_CONVERTED_CKPT_PATH=xxx

前提条件

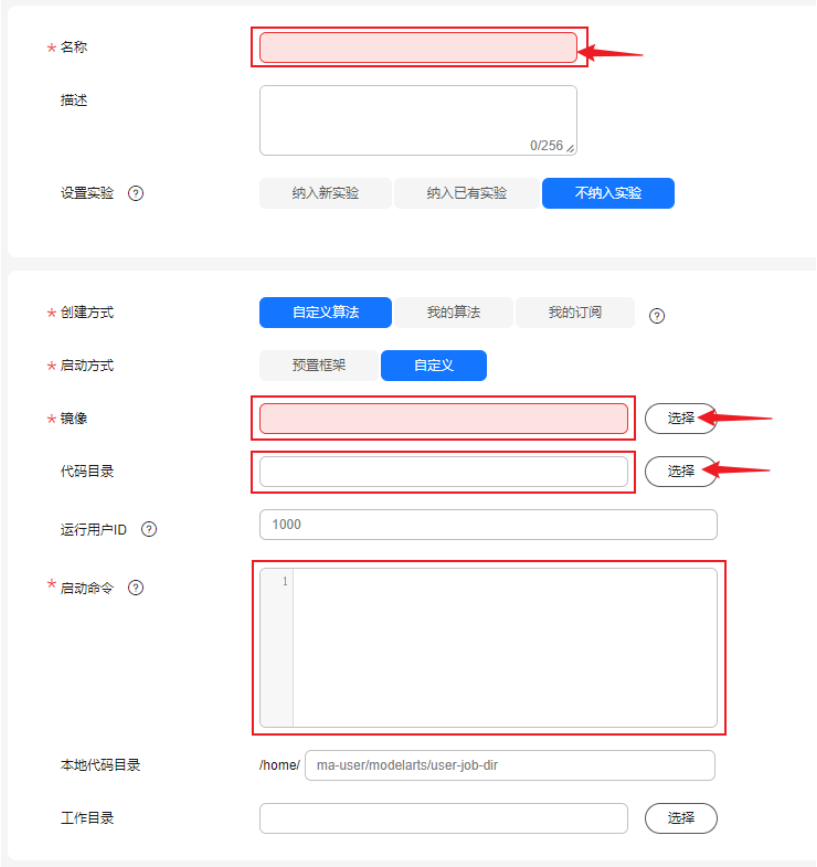
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendFactory代码目录。

图 4-59 创建训练作业



使用**基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendFactory;
sh install.sh modellink;
sh ./scripts_modellink/dev_pipeline.sh
```

使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendFactory;
source /usr/local/Ascend/ascend-toolkit/set_env.sh;
sh ./scripts_modellink/dev_pipeline.sh
```

命令详解如下：

- <cfgs_yaml_file>：性能测试配置的yaml文件地址，如**代码目录**中performance_cfgs.yaml相对或绝对路径。
- <model_name>：训练模型名，如qwen2-7b
- <exp_name>：实验名称，具体可以设置的值参考<cfgs_yaml_file>

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

输入指定的目录在训练开始时，平台会自动将指定的OBS路径下的文件copy到容器内

输出指定的目录在训练过程中，平台会自动将容器内的文件copy到指定的OBS路径下

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT、USER_PROCESSED_DATA_DIR、USER_CONVERTED_CKPT_PATH根据实际要求选择，示例如下。

   输入**数据集**变量：是否使用已处理好数据集；

   - 是，设置以下变量
     - **USER_PROCESSED_DATA_DIR**:已处理好数据路径目录
   - 否，使用原始数据集，设置以下变量
     - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入原始数据集路径。

   输入**权重词表**变量：是否使用已转换Megatron格式权重；

   - 是，设置以下变量
     - **USER_CONVERTED_CKPT_PATH**：已转换Megatron格式权重目录变量。
     - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重对应的存放目录地址。
   - 否，设置以下变量
     - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重对应的存放地址
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR，根据实际要求选择，示例如下；
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **MG_TO_HF_PATH**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件，USER_CONVERTED_CKPT_PATH则需选中存放已处理好数据集的目录文件夹。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-46表格中的配置进行填写。

图 4-60 环境变量

表 4-46 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
FINETUNING_TYPE	lora	表示训练策略。可选择值： • full：全参微调 • lora：lora微调
CODE_DIR	/home/ma-user/AscendFactory	【**可选**】工作目录,默认/home/ma-user/AscendFactory
SHELL_FOLDER	$CODE_DIR/scripts_modellink/${MODEL_NAME%-*}	【**可选**】模型转换数据、转换权重、训练脚本路径
MA_TRAIN_AUTO_RESUME	False	【**可选**】【**故障快恢**】是否开启此功能，【True、False】默认False不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。
STAGE	sft	表示当前的训练阶段。可选择值：【pt,sft】,默认sft • sft：代表监督微调； • pt：代表预训练；
CKPT_LOAD_TYPE	1	**可选**【0、1、2】，默认为1 • 0： 不加载权重 • 1：加载权重不加载优化器状态【增量训练】 • 2：加载权重且加载优化器状态【**断点续训**】

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。

环境变量	示例值	参数说明
SAVE_INTERVAL	1000	【**可选**】用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	【**可选**】表示训练完成的权重文件会自动转换为Hugging Face格式权重。【True、False】默认为True

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

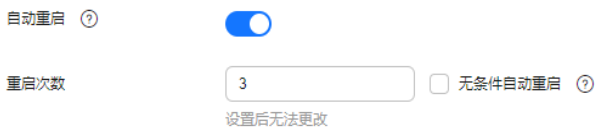
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-61 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格，设置变量MA_TRAIN_AUTO_RESUME为True的前提下，默认为Fasle。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-47**进行配置。

图 4-62 选择资源池规格

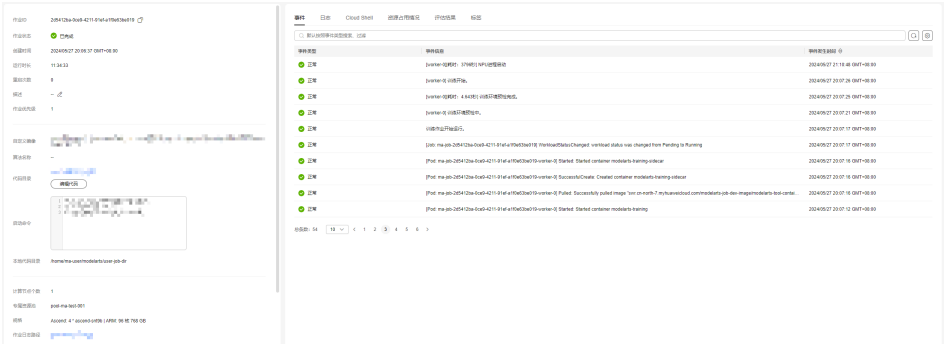
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.5.4 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

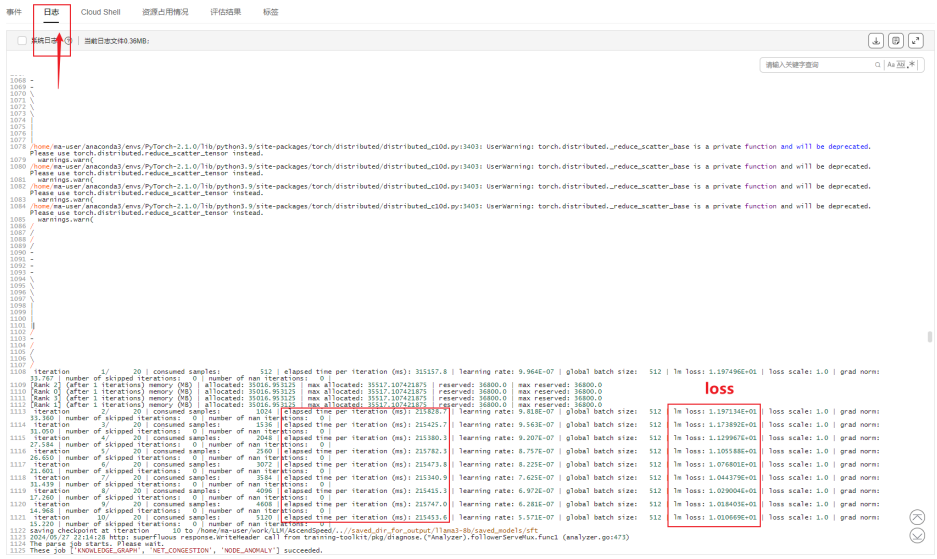
图 4-63 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-64 查看日志和性能



训练结束之后，在保存路径下生成了如下几个文件：

```
   |——converted_hf2mg_weight_TP${TP}PP${PP} # 训练过程Megatron格式权重
   |——converted_mg2hf_weight               # 训练完成转换为HF格式权重目录，只有配置了自动转换才有内容，否则为空
   |——logs                                 # 训练过程日志
   |——preprocessed_data                    # 训练过程预处理后数据集目录
   |——saved_checkpoints                    # 训练生成权重文件
```

4.5.5 训练脚本说明

4.5.5.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 ……）的训练脚本（在scripts_modellink下）和配置（在examples/config下），**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
```

```
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-47所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-47 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
2		llama2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
6	Qwen	qwen-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
7		qwen-14b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
11		qwen1.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
12		qwen1.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
17	Qwen2	qwen2-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
22	mistral	mistral-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	full	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5 -0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
27		qwen2.5-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
28		qwen2.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

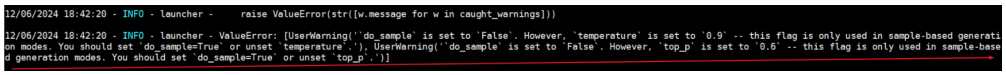
序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
32		llama3.2-3b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.5.5.2 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（ transformers==4.46.1 ），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendFactory/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-65所示。

图 4-65 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-66所示。

图 4-66 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-67 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-68所示。

图 4-68 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-69 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-70所示。

图 4-70 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41
42
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.5.5.3 断点续训和故障快恢说明

相同点

断点续训（Checkpointing）和故障快恢都是指训练中断后可从训练中一定间隔（${save-interval}）保存的模型（包括模型参数、优化器状态、训练迭代次数等）继续训练恢复，而不需要从头开始。

不同点

- 断点续训：可指定加载训练过程中生成的Megatron格式权重(${user_converted_ckpt_path})
- 故障快恢：默认加载${output_dir}/saved_checkpoints中最大迭代次数（iter_000xxxx）Megatron格式权重文件。

⚠ 注意

1. lora微调不支持断点续训
2. 启动前需检查latest_checkpointed_iteration.txt文件中内容是否与所需iter_000xxxx数字（表示训练后保存权重对应迭代次数）保持一致，不一致则修改latest_checkpointed_iteration.txt内容与iter_000xxxx保持一致。

```
|——${saved_checkpoints}
  |——iter_0000010
  |——iter_0000020
  |——latest_checkpointed_iteration.txt
```

示例，latest_checkpointed_iteration.txt文件内容：20

3. 同时开启故障快恢和断点续训时需满足以下条件：
   - 如果用户指定${user_converted_ckpt_path} 因故障快恢读取权重的优先级最高则训练过程的权重保存路径${output_dir}/saved_checkpoints（加载故障快恢路径） 必须为空，否则此参数无效断点续训失效。
   - 如果就是使用最新的训练权重进行断点续训（暂停+启动场景），那么可以同时指定train_auto_resume =1和 ${user_converted_ckpt_path}训练过程的权重保存路径，加载路径一致。

4.5.6 常见错误原因和解决方法

4.5.6.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表4-47进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

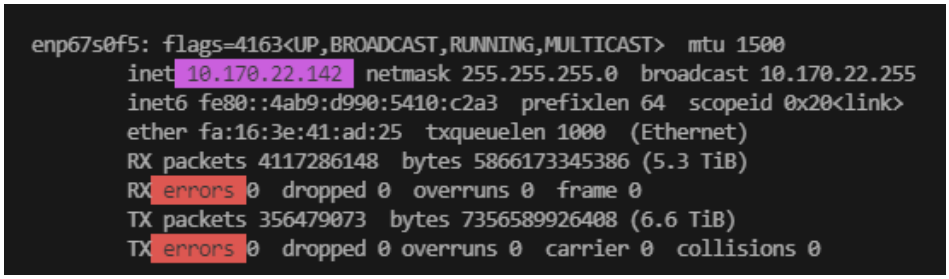
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.5.6.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-71 网卡名称错误




```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.5.6.3 mc2 融合算子报错

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务时产生mc2融合算子错误。

图 4-72 mc2 融合算子错误

```
File "/home/ma-user/AscendFactory/third-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
ird-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
    output, all_gather_grad_output = torch_npu.npu_all_gather_base_mm(
```

解决方法

修改代码文件：AscendFactory/scripts_modellink/{model_name}/3_training.sh文件，去除以下mc2融合算子--mc2

```
megatron_options="  \
    --tensor-model-parallel-size ${TP} \
    --pipeline-model-parallel-size ${PP} \
    --seed ${SEED} \
    --normalization RMSNorm \
    --position-embedding-type rope \
    --transformer-impl local \
    --sequence-parallel \
    --use-flash-attn \
    --bf16 \
    --swiglu \
    --use-fused-swiglu \
    --use-fused-rmsnorm \
    --use-distributed-optimizer \
    --use-fused-rotary-pos-emb \
    --use-rotary-position-embeddings \
    --use-mc2 \
    --no-masked-softmax-fusion \
```

4.6 主流开源大模型基于 Standard+OBS+SFS 适配 ModelLink PyTorch NPU 训练指导（6.3.912）

4.6.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.912版本相对于6.3.911版本新增如下内容：

1. 代码结构发生变化，统一了modellink和llama-factory的启动方式。
2. 继承911版本启动方式以外增加新的启动方式：
   ```
   ascendfactory-cli train xx.yaml model-name exp_name
   ```
3. 相对于6.3.911版本仅是使用run_type来指定训练的类型，只能区分预训练、全参微调和lora微调但实际上预训练和sft是训练的不同阶段，全参、lora是训练参数设置方式。为了更加明确的区分不同策略，以及和llama-factory对齐，6.3.912版本调整以下参数：
   - 新增STAGE，表示训练的阶段，可以选择的参数包括: {pt，sft}。
   - 新增FINETUNING_TYPE，表示微调的策略，可以选择的参数包括：{full，lora}
   - 删除RUN_TYPE

   所以当前的组合情况为：

项目	full	lora
pt（预训练）	√	√
sft（指令微调）	√	√

支持的模型列表

本方案支持以下模型的训练，如表4-48所示。

表 4-48 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce

序号	支持模型	支持模型参数量	权重文件获取地址
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-73 操作原理图

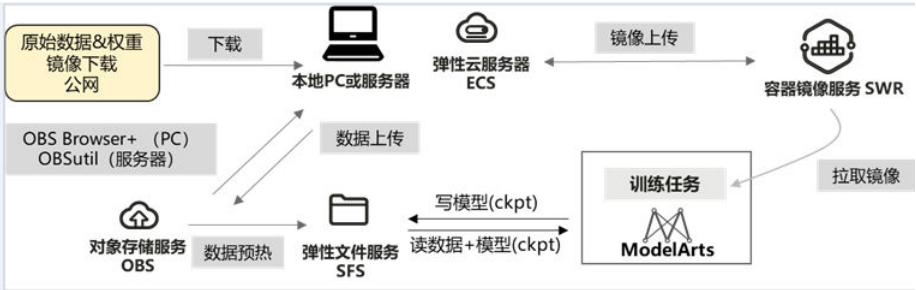


表 4-49 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendFactory训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
训练	预训练/微调	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.6.2 准备工作

4.6.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-54**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。具体过程请参考**创建OBS桶**，例如桶名：standard-llama2-13b。

由于ModelArts创建训练作业时，需要将作业日志输出至OBS桶中，因此创建OBS桶为必选项。用户可通过**OBS Browser+**、**obsutil**等工具访问和管理OBS桶，将代码、模型文件、数据集等数据上传或下载进行备份。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

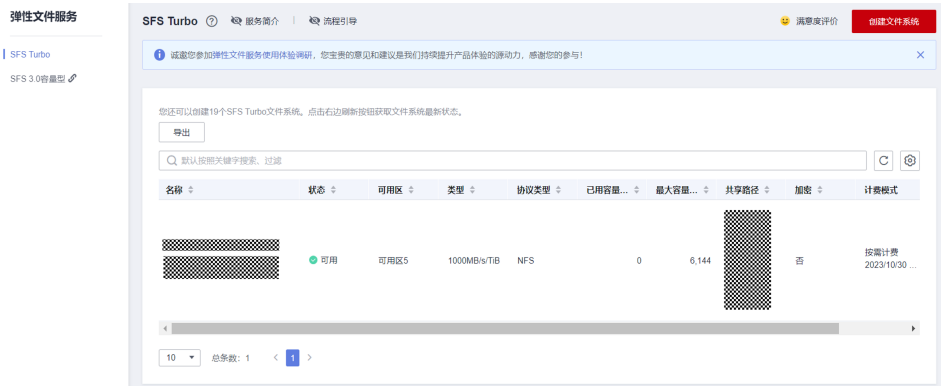
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-74 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

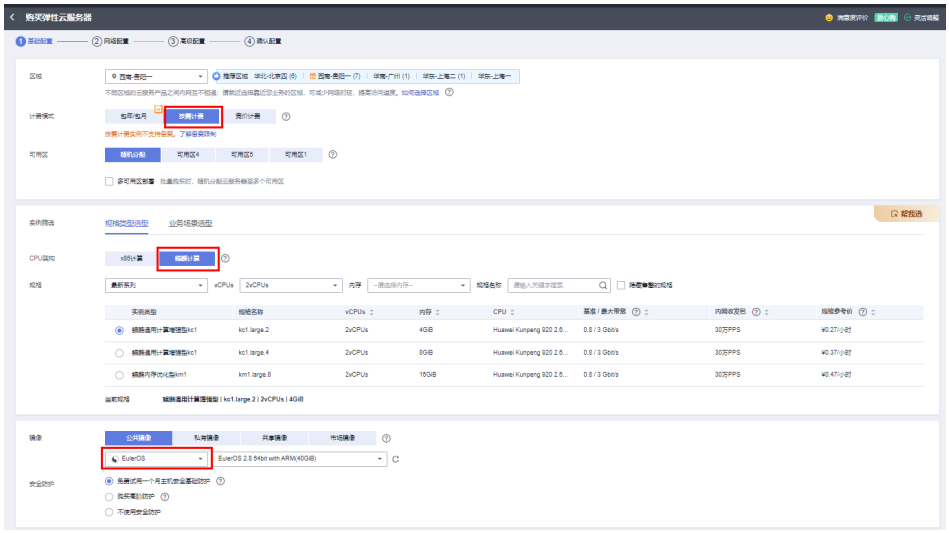
图 4-75 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 4-76 购买 ECS

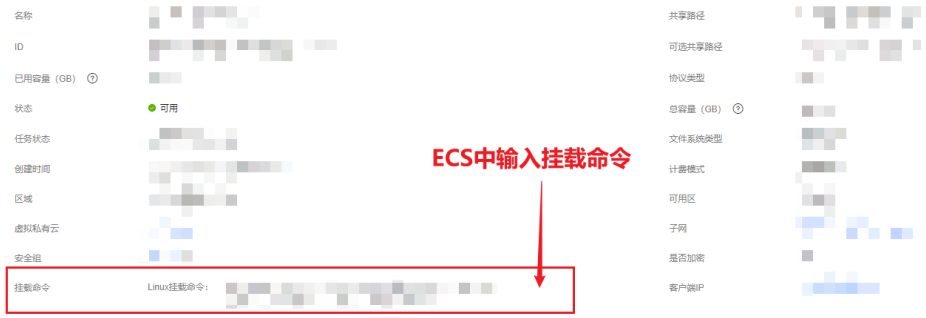


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看基本信息**图4-77**，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 4-77 SFS Turbo 基本信息



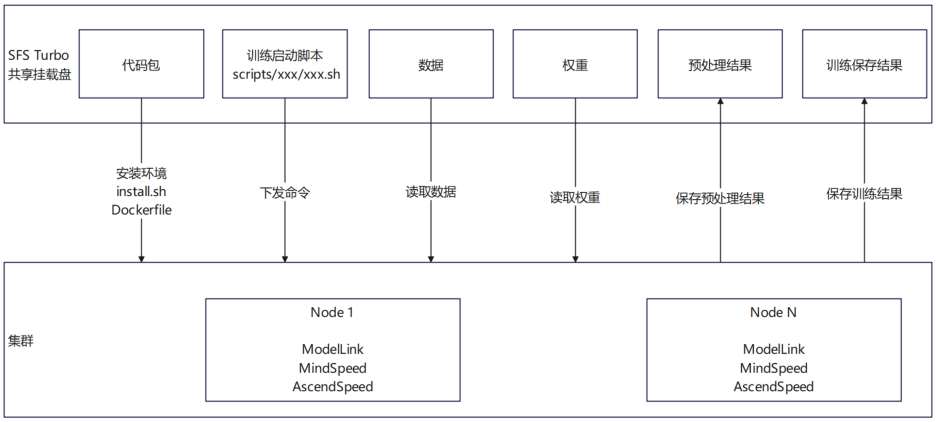
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-78 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表4-51**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中llm_train/AscendFactory/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendFactory路径访问。
4. 在ModelArts中创建训练作业如：**执行训练任务【新】**，执行命令：ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>开始训练。
5. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.6.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- 微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca指令微调数据**：如上述提供的alpaca_gpt4_data.json数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction对应的内容会与input对应的内容拼接后作为指令，即指令为instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持MOSS格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他
  ```

```
人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
```

```
        ],
        "system": "系统提示词（选填）",
        "tools": "工具描述（选填）"
    }
]
```

上传数据集至 SFS Turbo

准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。可通过两种方式，将数据集上传至SFS Turbo中。

方式一：将下载的原始数据通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/training_data目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具。数据存放参考目录：

```
/mnt/sfs_turbo/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
2. 利用**OBS Browser+工具**将下载的数据集上传至创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.6.2.3 准备权重

获取对应模型的权重文件，获取链接参考**表4-48**。权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。文件会直接下载用户本地，需要再上传至SFS Turbo中。
- **方法二：huggingface-cli**：**huggingface-cli**是Hugging Face官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

随后可通过以下两种方式，将下载到本地的模型文件上传至SFS Turbo中。

本地上传权重文件至 SFS Turbo

通过以下两种方式将下载到本地的模型文件上传至SFS Turbo中。方式一操作简单，但是数据传输速度比较慢，费时间。方式二操作相对方式一复杂一些，但是数据传输速度较快。

方式一：将已下载的模型文件通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/model目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放模型，例如在桶standard-llama2-13b中创建文件夹model/llama-2-13b-hf。
2. 利用**OBS Browser+工具**将下载的模型文件上传至创建的文件夹目录下。
3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.6.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-50**所示。

表 4-50 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.912代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts_modellink文件夹中：

```
|——AscendCloud-LLM
  |——llm_train                  # 模型训练代码包
    |——AscendFactory
      |——examples/               # config配置文件目录
      |——data.tgz                # 样例数据压缩包
      |——third-party/            # patch包
      |——src/acs_train_solution/  # 训练运行包
      |——intall.sh               # 需要的依赖包
      |——scripts_llamafactory/    # llamafactory兼容旧版本启动方式目录
      |——scripts_modellink/       # modelLink兼容旧版本启动方式目录
      |——Dockerfile
```

代码上传至 SFS Turbo

将AscendFactory代码包AscendCloud-LLM-xxx.zip直接上传至ECS服务器中的SFS Turbo中，例如存放在/mnt/sfs_turbo/AscendCloud-LLM-xxx.zip目录下并解压缩。

```
unzip AscendCloud-*.zip
unzip AscendCloud-LLM-*.zip
```

⚠ 注意

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务如产生mc2融合算子错误，可参考**mc2融合算子报错**

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至SFS Turbo后，目录结构如下。

```
/mnt/sfs_turbo/
|——llm_train                              # 模型训练代码包
   |——AscendFactory
      |——config/                          # 配置文件
         |——deepspeed/                       # deepspeed配置json文件
         |——modellink_performance_cfgs.yaml  # ModelLink训练配置json文件
         |——.......
      |——data.tgz                 #样例数据压缩包
      |——intall.sh                # 需要的依赖包
      |——scripts_modellink/         # modelLink兼容旧版本启动方式目录
         |——llama3                 # llama3系列模型执行脚本的文件夹
         |——qwen2.5                 # Qwen2.5系列模型执行脚本的文件夹
         |—— ...
         |—— dev_pipeline.sh       # 系列模型共同调用的多功能的脚本
      |——third-party/              # patch包
      |——src/acs_train_solution/   # 训练运行包
         |——ascendcloud_patch/     # patch补丁包
         |——benchmark/             #工具包，存放数据集及基线数据
            |——trainer.py          # 训练启动脚本
            |——performance.py      # benchmark训练性能比较启动脚本
            |——accuracy.py         # benchmark训练精度启动脚本
   |——model/Qwen2-7B/               # 权重词表文件目录，如Qwen2-7B
   |——training_data                 # 原始数据目录
      |——alpaca_gpt4_data.json    # 微调数据
      |——train-00000-of-00001-a09b74b3ef9c3b56.parquet #预训练数据
   |——{output_dir} #{OUTPUT_SAVE_DIR}或yaml文件{output_dir}参数设置值
      # 自动生成数据目录结构
      |—— preprocessed_data
      |——converted_hf2mg_weight_TP${TP}PP${PP}
      |——checkpoint    # 训练完成生成目录Qwen2-7B，自动生成
```

4.6.2.5 准备镜像

4.6.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-51 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	CANN：cann_8.0.rc3 PyTorch：2.3.1

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**可二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

使用以上方案时，都会下载Megatron-LM、MindSpeed、ModelLink源码至

```
AscendFactory/third-party/
  |——MindSpeed/          # MindSpeed昇腾大模型加速库
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——ModelLink/          # ModelLink端到端的大语言模型方案
     |——megatron/        # 注意：该文件夹从Megatron-LM中复制得到
     |——...
  |——transformers.patch
  |——llama-factory.patch
```

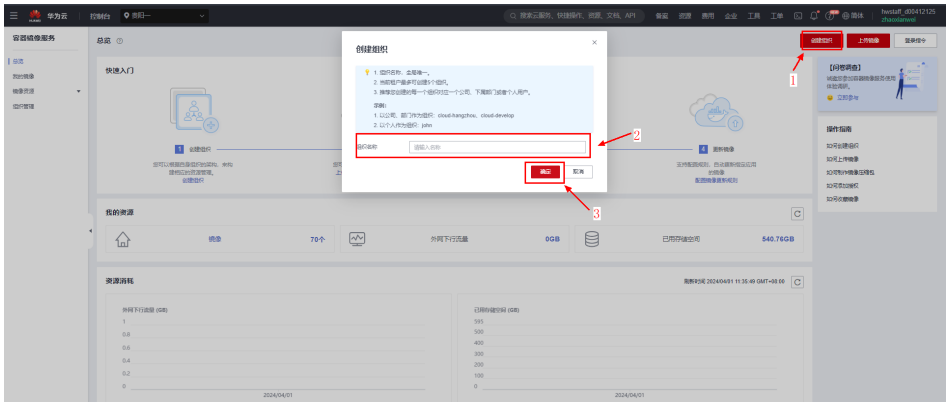
训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

4.6.2.5.2 ECS 获取和上传基础镜像

Step1 创建镜像组织

在SWR服务页面创建镜像组织。

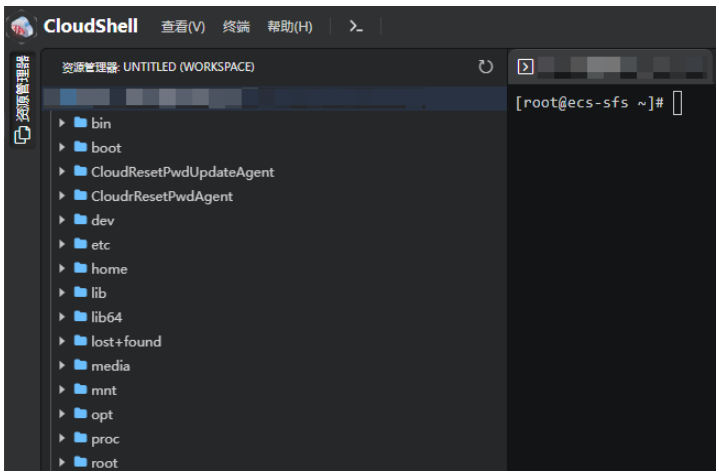
图 4-79 创建镜像组织



Step2 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录如图所示。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

图 4-80 CloudShell 远程登录界面



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step4 获取训练镜像

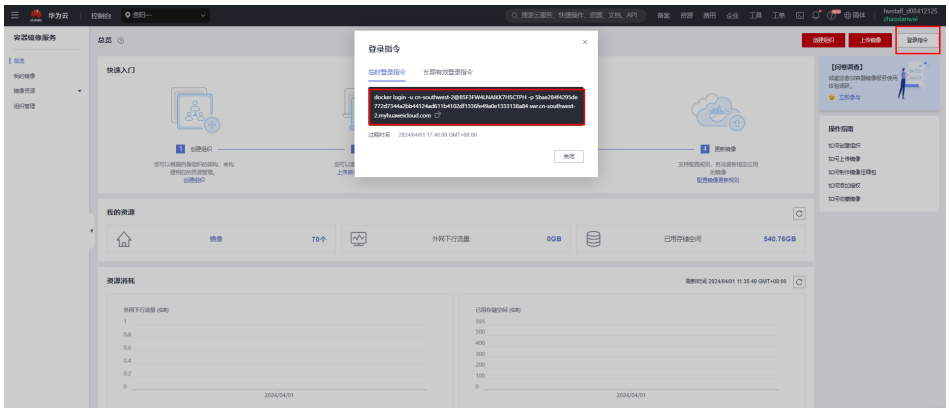
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-51**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-81 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_3_ascend:20241212

示例：

```
docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/
pytorch_2_3_ascend:20241212
```

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

```
docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/
pytorch_2_3_ascend:20241212
```

4.6.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-82**中都需要执行 install.sh文件，来安装依赖以及下载完整代码。

⚠ 注意

- 使用基础镜像的方法，需要确认训练作业的资源池是否联通公网，否则执行 install.sh 文件时下载代码会失败。因此可以选择配置网络或使用**ECS中构建新镜像**的方法。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 中的 transformers 的版本。

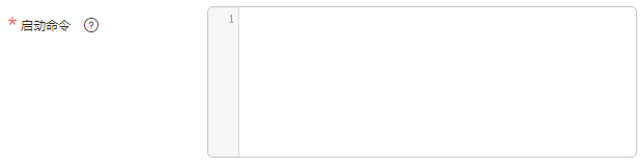
  由默认 transformers==4.47.0 修改为：transformers==4.44.2

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendFactory;
sh ./scripts_modellink/install.sh;
sh ./scripts_modellink/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-82 训练作业启动命令



4.6.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-50**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-xxx.zip，并直接进入到llm_train/AscendFactory文件夹下面
   ```
   cd ./llm_train/AscendFactory
   ```
2. 编辑llm_train/AscendFactory中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```

⚠ 注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

由默认 transformers==4.47.0 修改为：transformers==4.44.2

3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网

```
docker build -t <镜像名称>:<版本名称> .
```

如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

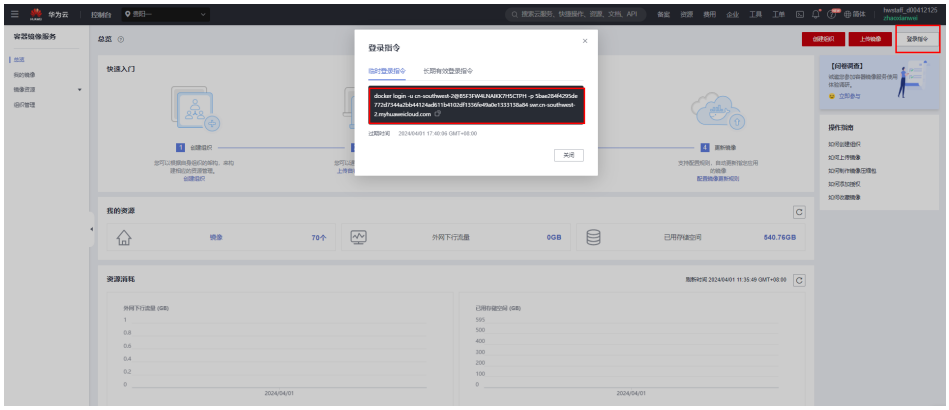
```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host --build-arg install_type=modellink -t <镜像名称>:<版本名称> .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称，后续使用 ${dockerfile_image_name} 进行表示。
- install_type:安装类型，默认为all，可选【modellink、llmafactory、all】默认为all

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-83 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20241212

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20241212

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20241212

4.6.3 执行训练任务

4.6.3.1 执行训练任务【新】

新的训练方式将统一管理训练日志、训练结果和训练配置，使用yaml配置文件方便用户根据自己实际需求进行修改。推荐用户使用该方式进行训练。

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器（默认开启）	断点续训：加载权重+优化器
• pt • sft	ckpt_load_type=0	ckpt_load_type=1 user_converted_ckpt_path=xxx，	ckpt_load_type=2 user_converted_ckpt_path=xxx

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

步骤一 修改训练 Yaml 配置文件

修改或添加**代码目录**下modellink_performance_cfgs.yaml文件参数内容，参数详解可查看**表4-52**。

样例yaml配置文件结构分为：

- base块：基础配置块，主要为公共配置参数
- ModelName块：该模型所需配置的参数，如qwen2.5-7b块
  - exp_name：实验块；训练策略-序列长度所需参数配置

样例yaml文件仅展示qwen1.5-7b-4096-sft-full配置，如需其他配置需根据样例自行添加。

图 4-84 yaml 文件样例


```
base: &base
  backend: modellink
  scirpts_dir: /home/ma-user/AscendFactory/script
  dataset: /path/to/alpaca_en_demo.json
  handler-name: AlpacaStyleInstructionHandler

qwen1.5-7b:
  _base: &qwen1_5-7b
    <<: *base
    model_name_or_path: /path/to/Qwen1.5-7B-Chat
    prompt-type: qwen
  full:
    <<: *qwen1_5-7b
    stage: sft
    finetuning_type: full
    tensor-model-parallel-size: 1
    pipeline-model-parallel-size: 4
    output_dir: ./saved_dir_for_output/qwen1.5-7b
```

表 4-52 模型训练参数

参数	示例值	参数说明
dataset	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	**【必修改】**训练时指定的输入数据路径。请根据实际规划修改。用户根据训练情况二选一；
processed_data_dir	/home/ma-user/ws/xxx	已处理好数据路径目录，如有处理完成数据可设置此参数
scirpts_dir	/home/ma-user/ws/llm_train/AscendFactory/scripts_modellink	**【必修改】**ModelLink脚本相对或绝对路径，用于方便加载脚本
model_name_or_path	/home/ma-user/work/llm_train/AscendFactory/model/llama2-70B	**【必修改】**加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
output_dir	/home/ma-user/work/save_dir	**【必修改】**训练任务结束生成日志及权重文件目录

参数	示例值	参数说明
ckpt_load_type	1	【**可选**】默认为1 • 0，不加载权重 • 1，加载权重不加载优化器状态【**增量训练**】 • 2，加载权重且加载优化器状态【**断点续训**】详见**断点续训和故障快恢说明**
user_converted_ckpt_path	/home/ma-user/ws/xxx	【**可选**】已转换Megatron格式权重目录或训练输出结果权重目录，一般搭配断点续训或增量训练。 • 增量训练：转换Megatron权重，如不指定默认为${output_dir}/converted_hf2mg_weight_TP{tp}PP{pp}目录。 • 断点续训：训练过程中保存的某个权重，可参考**断点续训和故障快恢说明**
train_auto_resume	false	【**可选**】是否开启【**故障快恢**】功能，【true、false】默认false不开启，当训练中断时重启任务会从最新生成权重文件处继续训练。可参考**断点续训和故障快恢说明**
stage	pt	表示训练类型。可选择值： • pt：预训练 • sft：指令微调
finetuning_type	full	表示训练策略。可选择值： • full：全参微调 • lora：lora微调

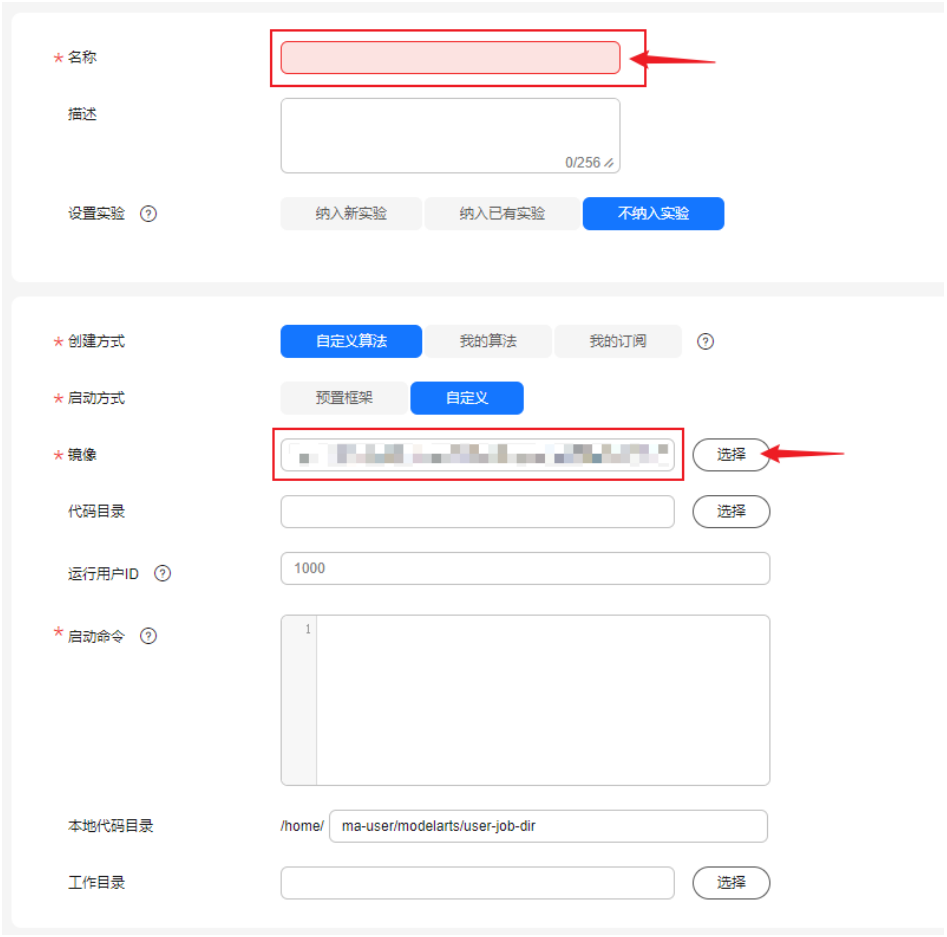
参数	示例值	参数说明
handler-name	• GeneralPretrainHandler • GeneralInstructionHandler • MOSSInstructionHandler • AlpacaStyleInstructionHandler • ShareggptStyleInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSInstructionHandler：使用微调的moss数据集。 • AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 • ShareggptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
micro-batch-size	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
global-batch-size	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
tensor-model-parallel-size	8	表示张量并行。
pipeline-model-parallel-size	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
context-parallel-size	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加此值（≥ 2）。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
lr	2.5e-5	学习率设置。
min-lr	2.5e-6	最小学习率设置。
seq-length	4096	要处理的最大序列长度。
convert_mg2hf_at_last	1	Megatron格式权重转换为HuggFace格式权重，如不

参数	示例值	参数说明
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
train-iters	10	非必填。表示训练step迭代次数。默认值为10
seed	1234	随机种子数。每次数据采样时，保持一致。
save-interval	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
save-total-limit	-1	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

步骤二 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-85 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendFactory;
sh install.sh modellink;
source /usr/local/Ascend/ascend-toolkit/set_env.sh;
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendFactory;
ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
```

--命令参数详解如下：

- <cfgs_yaml_file>：ModelLink配置yaml文件地址，如**代码目录**中modellink_performance_cfgs.yaml相对或绝对路径，根据自己要求执行
- <model_name>：训练模型名，如qwen1.5-7b，需与<cfgs_yaml_file>里面对应
- <exp_name>：实验名称：指定本次实验的具体配置，还包括数据配置等，比如full, lora等，该名称需要和<cfgs_yaml_file>里面对应。

说明

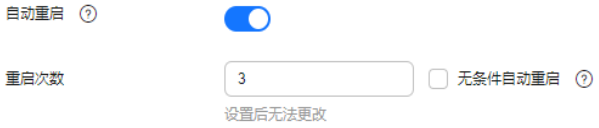
如单独修改某个参数值，也可单击“增加超参”，在增加的超参填写框中，按照表4-52表格中的配置进行填写。

图 4-86 超参

步骤三 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-87 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

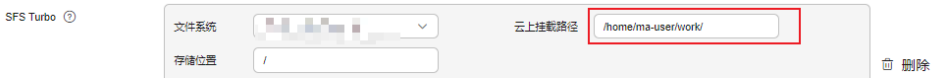
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**不同模型推荐参数、NPU卡数**进行配置。

图 4-88 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-89 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.6.3.2 执行训练任务【旧】

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b SFT全参微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

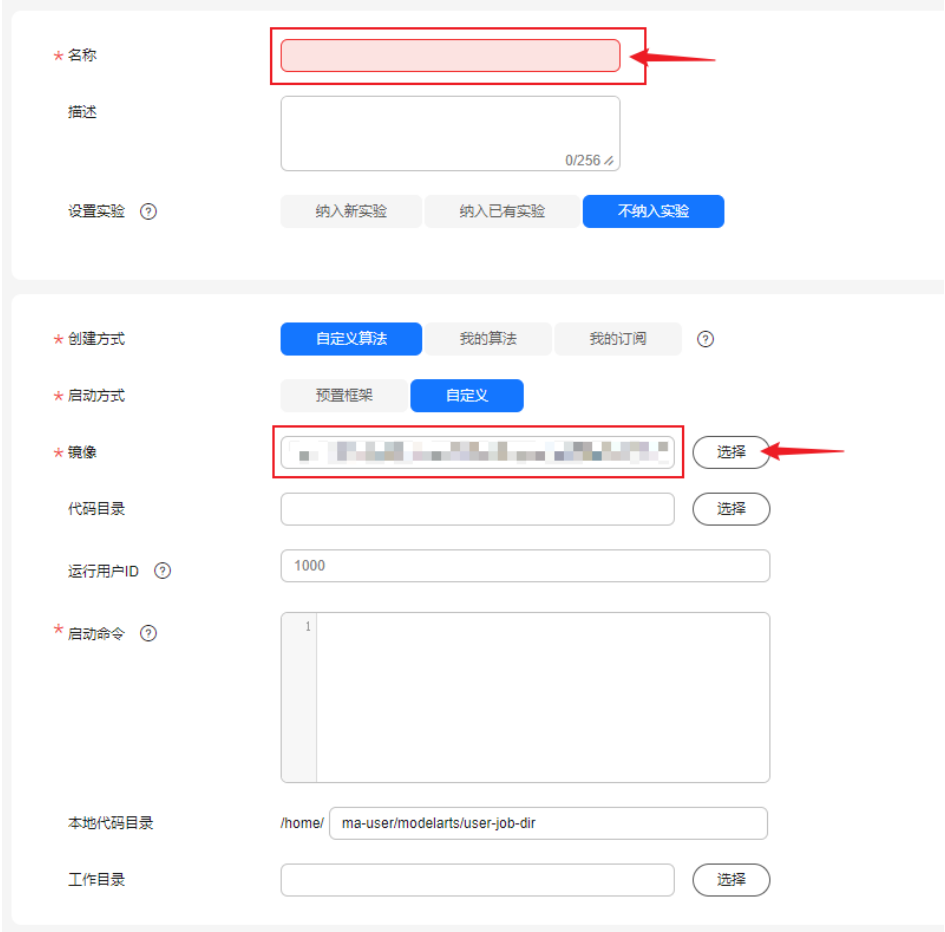
修改模型训练脚本中的配置，参数详解可查看**训练参数说明**，其中【GBS、MBS、TP、PP】参数值可参考**模型推荐参数、NPU卡数**设置。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-90 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

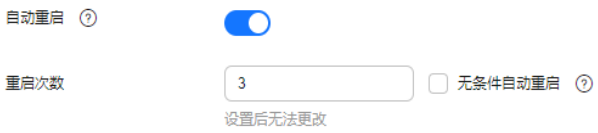
```
cd /home/ma-user/work/llm_train/AscendFactory;
sh ./scripts_modellink/install.sh;
sh ./scripts_modellink/llama2/0_pl_sft_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendFactory;
sh ./scripts_modellink/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-91 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

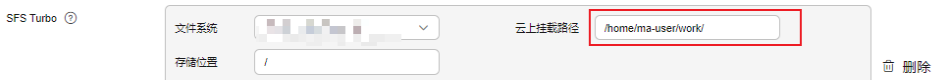
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**不同模型推荐参数、NPU卡数**进行配置。

图 4-92 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-93 选择 SFS Turbo



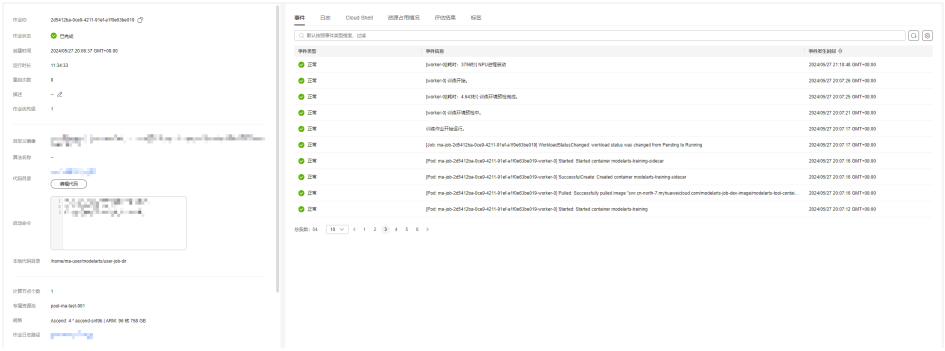
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.6.4 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

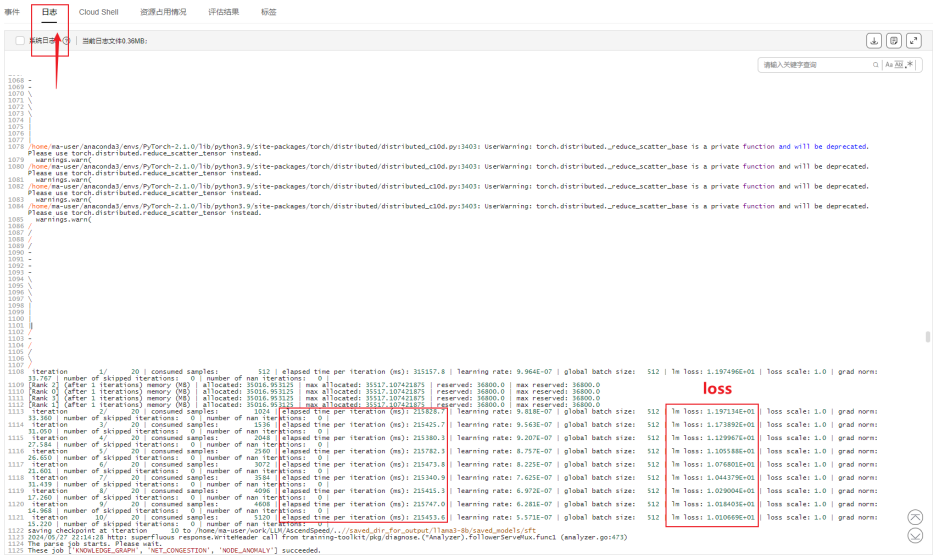
图 4-94 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-95 查看日志和性能



4.6.5 训练脚本说明

4.6.5.1 训练启动脚本说明和参数配置【旧】

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**

环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-53 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	【预训练：pt】**预训练数据集**相对或绝对地址 【微调：sft】**微调数据集**相对或绝对地址	**【必改】**。训练时指定的输入原始数据路径。请根据实际规划修改。用户根据训练情况二选一；
USER_PROCESSED_DATA_DIR	/home/ma-user/work/process_data	**【可选】**如已有预处理完成数据可指定此目录,训练过程中会优先加载此目录，跳过数据预处理过程；默认无此参数，用户自定义自行添加
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**【必改】**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
OUTPUT_SAVE_DIR	/home/ma-user/work/AscendFactory/saved_dir_for_output/	**【必改】**该路径下统一保存生成的 CKPT、PLOG、LOG 文件。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/AscendFactory/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
STAGE	pt	表示当前的训练阶段。可选择值：【pt、sft】 • sft：代表监督微调； • pt：代表预训练；
FINETUNING_TYPE	full	表示训练策略。可选择值【full、lora】： • full：全参微调 • lora：lora微调

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-54所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-54 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点&4*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点&4*Ascend
22	mistral	mistral-7b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	full	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	full	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			full	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	full	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			full	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

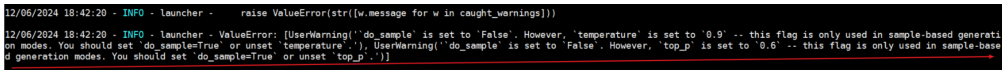
序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	full	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			full	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.6.5.2 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.46.1），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendFactory/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-96所示。

图 4-96 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-97所示。

图 4-97 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-98 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321 
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327 
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-99所示。

图 4-99 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295 
```

图 4-100 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-101所示。

图 4-101 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.6.5.3 断点续训和故障快恢说明

相同点

断点续训（Checkpointing）和故障快恢都是指训练中断后可从训练中一定间隔（${save-interval}）保存的模型（包括模型参数、优化器状态、训练迭代次数等）继续训练恢复，而不需要从头开始。

不同点

- 断点续训：可指定加载训练过程中生成的Megatron格式权重(${user_converted_ckpt_path})
- 故障快恢：默认加载${output_dir}/saved_checkpoints中最大迭代次数（iter_000xxxx）Megatron格式权重文件。

⚠ 注意

1. lora微调不支持断点续训
2. 启动前需检查latest_checkpointed_iteration.txt文件中内容是否与所需iter_000xxxx数字（表示训练后保存权重对应迭代次数）保持一致，不一致则修改latest_checkpointed_iteration.txt内容与iter_000xxxx保持一致。

```
|——${saved_checkpoints}
  |——iter_0000010
  |——iter_0000020
  |———latest_checkpointed_iteration.txt
```

示例，latest_checkpointed_iteration.txt文件内容：20

3. 同时开启故障快恢和断点续训时需满足以下条件：
   - 如果用户指定${user_converted_ckpt_path} 因故障快恢读取权重的优先级最高则训练过程的权重保存路径${output_dir}/saved_checkpoints（加载故障快恢路径） 必须为空，否则此参数无效断点续训失效。
   - 如果就是使用最新的训练权重进行断点续训（暂停+启动场景），那么可以同时指定train_auto_resume =1和 ${user_converted_ckpt_path}训练过程的权重保存路径，加载路径一致。

4.6.6 常见错误原因和解决方法

4.6.6.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size） 和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-54**进行设置。

- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

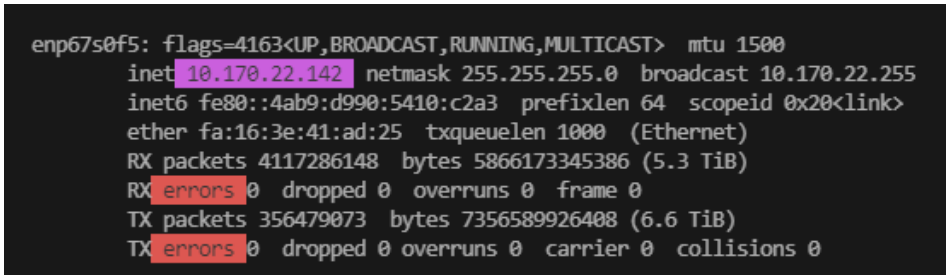
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.6.6.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-102 网卡名称错误



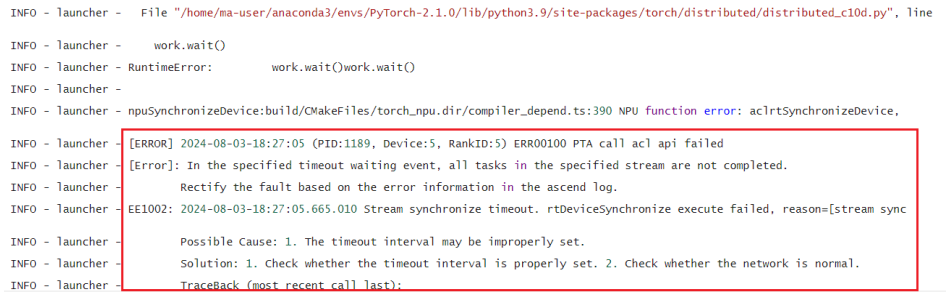

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.6.6.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-103 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.6.6.4 mc2 融合算子报错

Yi-34B、Qwen1.5系列、GLM4-9B模型执行lora微调策略任务时产生mc2融合算子错误。

图 4-104 mc2 融合算子错误

```
  File "/home/ma-user/AscendFactory/third-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
ird-party/ModelLink/modellink/core/tensor_parallel/ascend_turbo/mc2_linears_seq_parallel.py", line 42, in forward
    output, all_gather_grad_output = torch_npu.npu_all_gather_base_mm(
```

解决方法

修改代码文件：AscendFactory/scripts_modellink/{model_name}/3_training.sh文件，去除以下mc2融合算子--mc2

```
megatron_options="  \
    --tensor-model-parallel-size ${TP} \
    --pipeline-model-parallel-size ${PP} \
    --seed ${SEED} \
    --normalization RMSNorm \
    --position-embedding-type rope \
    --transformer-impl local \
    --sequence-parallel \
    --use-flash-attn \
    --bf16 \
    --swiglu \
    --use-fused-swiglu \
    --use-fused-rmsnorm \
    --use-distributed-optimizer \
    --use-fused-rotary-pos-emb \
    --use-rotary-position-embeddings \
    --use-mc2 \
    --no-masked-softmax-fusion \
```

4.7 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.912）

4.7.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的不同训练阶段方案，包括指令监督微调、DPO偏好训练、RM奖励模型训练、PPO强化训练方案。

- DPO(Direct Preference Optimization)：直接偏好优化方法，通过直接优化语言模型来实现对大模型输出的精确把控，不用进行强化学习，也可以准确判断和学习到使用者的偏好，最后，DPO算法还可以与其他优化算法相结合，进一步提高深度学习模型的性能。
- RM奖励模型(Reward Model)：是强化学习过程中一个关键的组成部分。它的主要任务是根据给定的输入和反馈来预测奖励值，从而指导学习算法的方向，帮助强化学习算法更有效地优化策略
- PPO强化学习(Proximal Policy Optimization)：是一种在强化学习中广泛使用的策略优化算法。它属于策略梯度方法的一种，旨在通过限制新策略和旧策略之间的差异来稳定训练过程。PPO通过引入一个称为“近端策略优化”的技巧来避免过大的策略更新，从而减少了训练过程中的不稳定性和样本复杂性。
- 指令监督式微调(Self-training Fine-tuning)：是一种利用有标签数据进行模型训练的方法。 它基于一个预先训练好的模型，通过调整模型的参数，使其能够更好地拟合特定任务的数据分布。 与从头开始训练模型相比，监督式微调能够充分利用预训练模型的知识和特征表示，从而加速训练过程并提高模型的性能。

训练阶段下有不同的训练策略，分为全参数训练、部分参数训练、LoRA、QLoRA，本文档主要支持全参数（Full）和LoRA、LoRA+。

- LoRA(Low-Rank Adaptation)： 这种策略主要针对如何在保持模型大部分参数固定的同时，通过引入少量可训练参数来调整模型以适应特定任务。
- LoRA+(Efficient Low Rank Adaptation of Large Models)：延续了LoRA的精髓进一步提升了在复杂任务上对大模型进行微调的效率和性能，核心在于其独特的学习率比率（loraplus_lr_ratio）机制，适用于那些需要精确控制模型微调过程的场景，当前该策略仅支持qwen1.5-7B指令监督式微调。
- 全参训练（Full）：这种策略主要对整个模型进行微调。这意味着在任务过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表4-57**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.RC3。
- Server驱动版本要求23.0.6
- PyTorch版本：2.3.1
- Python版本：3.10
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如表4-55所示。

表 4-55 支持的模型列表及权重文件地址

支持模型	支持模型参数量	权重文件获取地址
Llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
	llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
	llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-chat-hf
Llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
	llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
Llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct/tree/main
	llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct/tree/main
Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
	qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
	qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
	qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
	yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
	qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
	qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
	qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct

支持模型	支持模型参数量	权重文件获取地址
Qwen2_VL（支持多模态数据集）	qwen2_vl-2b	**https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main**
	qwen2_vl-7b	**https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main**
	qwen2_vl-72b	**https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct**
Falcon2	falcon-11B	**https://huggingface.co/tiiuae/falcon-11B**
GLM-4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本 4b556ad4d70c38924cb8c120adbf21a0012de6ce
Qwen2.5	qwen2.5-0.5b	**https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct**
	qwen2.5-7b	**https://huggingface.co/Qwen/Qwen2.5-7B-Instruct**
	qwen2.5-14b	**https://huggingface.co/Qwen/Qwen2.5-14B-Instruct**
	qwen2.5-32b	**https://huggingface.co/Qwen/Qwen2.5-32B-Instruct**
	qwen2.5-72b	**https://huggingface.co/Qwen/Qwen2.5-72B-Instruct**
llama3.2	llama3.2-1b	**https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct**
	llama3.2-3b	**https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct**

表 4-56 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendFactory训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。

阶段	任务	说明
训练	启动训练	介绍各个训练阶段：指令微调、PPO强化训练、RM奖励模型、DPO偏好训练使用全参/lora训练策略进行训练任务、性能查看。

4.7.2 准备工作

4.7.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-62**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Lite Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户如果购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.7.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-57**所示，模型列表、对应的开源权重获取地址如**表4-55**所示。

表 4-57 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendFactory是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.912中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
  |——llm_train                  # 模型训练代码包
    |——AscendFactory
      |——examples/               # config配置文件目录
      |——data.tgz                # 样例数据压缩包
      |——third-party/            # patch包
      |——src/acs_train_solution/  # 训练运行包
      |——intall.sh               # 需要的依赖包
      |——scripts_llamafactory/    # llamafactory兼容旧版本启动方式目录
      |——scripts_modellink/       # modelLink兼容旧版本启动方式目录
      |——Dockerfile
```

工作目录介绍

详细的工作目录参考如下，根据实际要求设置。

```
${workdir}（例如/home/ma-user/ws）
|——llm_train                          # 模型训练代码包
   |——AscendFactory
      |——examples/                     # config配置文件目录
        |——config/                    # 配置文件
          |——deepspeed/                # deepspeed配置json文件
          |——performance_cfgs.yaml    # 微调性能配置json文件
          |——llama_factory_performance_cfgs_VL.yaml   # qwen2vl微调json文件
          |——accuracy_cfgs.yaml       # 训练精度配置json文件
          |——.......
      |——data.tgz                    # 样例数据压缩包
      |——intall.sh                   # 需要的依赖包
      |——scripts_llamafactory/      # 兼容旧版本启动方式目录
        |——tools/                    # 针对昇腾云平台适配的功能补丁包
        |——demo.yaml                 # 样例yaml配置文件
        |——demo.sh                   # 训练启动shell脚本
        |——merge_lora_cli.sh         # lora权重合并脚本
      |——third-party/                # patch包
      |——src/acs_train_solution/     # 训练运行包
        |——ascendcloud_patch/        # patch补丁包
        |——benchmark/                #工具包，存放数据集及基线数据
          |——trainer.py              # 训练启动脚本
          |——performance.py          # 训练性能比较启动脚本
          |——accuracy.py             # 训练精度启动脚本
```

```
|──model/Qwen2-7B/            # 权重词表文件目录，如Qwen2-7B
|──saves/qwen2-7b/sft_lora/   # 训练完成生成目录Qwen2-7B，自动生成
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/model/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p model/Qwen2-72B
   ```

4.7.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-58 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527

表 4-59 模型镜像版本

模型	版本
CANN	cann_8.0.RC3
驱动	23.0.6
PyTorch	2.3.1

步骤一：检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三：启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

– ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

– --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如：  chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录
cd /home/ma-user/ws/llm_train/AscendFactory
#执行安装命令,安装依赖包及LLaMAFactory代码包
sh install.sh llamafactory
```

4.7.2.4 DockerFile 构建镜像（可选）

本章节主要介绍通过DockerFile文件构建训练镜像，将训练过程中依赖包封装使用，过程中需要连接互联网git clone，请确保环境可以访问公网，详解操作如下：

进入代码包Dockerfile文件同级目录：

```
cd /home/ma-user/ws/llm_train/AscendFactory
```

构建新镜像：

```
docker build -t <镜像名称>:<版本名称>  .
```

如无法访问公网则需配置代理，增加`--build-arg`参数指定代理地址确保访问公网。

```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx"
--network=host --build-arg install_type=llamafactory -t <镜像名称>:<版本名称>  .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_2_ascend:20241106
- install_type:安装类型，默认为all，可选【modellink、llmafactory、all】

注意

构建镜像前需保证Dockerfile文件内容中镜像名与本文档**镜像**保持一致，如不同则需修改为一致。

```
# 修改以下内容：
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/xxx
```

4.7.2.5 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前支持alpaca格式和sharegpt格式的微调数据集；使用自定义数据集时，请更新dataset_info.json文件；请务必在dataset_info.json文件中添加数据集描述；具体示例如下。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/AscendFactory/data目录下。具体步骤如下：

1. 解压data.tgz压缩包
   ```
   tar -zxvf /home/ma-user/ws/llm_train/AscendFactory/data.tgz
   ```
2. 进入到/home/ma-user/ws/llm_train/AscendFactory/data目录下。
   ```
   cd  /home/ma-user/ws/llm_train/AscendFactory/data
   ```
3. 将自定义原始数据（指令监督微调样例数据集：alpaca_gpt4_data.json.json）按照下面的数据存放目录要求放置。

   说明

   指令微调样例数据集alpaca_gpt4_data.json的下载链接：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws/llm_train ）
    |── AscendFactory/data
        |── alpaca_en_demo.json                # 代码原有数据集
        |── identity.json                    # 代码原有数据集
        ...
        |── alpaca_gpt4_data.json              # 自定义数据集
   ```
4. 更新代码目录下data/dataset_info.json文件。如使用以下示例数据集则命令如下。关于数据集文件格式及配置，更多样例格式信息请参考**README_zh.md** 的内容。
   ```
   vim dataset_info.json
   ```

   新加配置参数如下：

   ```
   "alpaca_gpt4_data": {
     "file_name": "alpaca_gpt4_data.json"
   },
   ```

   样例截图：

   ```
   {
     "identity": {
       "file_name": "identity.json"
     },
     "alpaca_en_demo": {
       "file_name": "alpaca_en_demo.json"
     },
     "alpaca_zh_demo": {
       "file_name": "alpaca_zh_demo.json"
     },
     "alpaca_gpt4_data": {
       "file_name": "alpaca_gpt4_data.json"
     },
   ```

4.7.3 执行训练任务

4.7.3.1 ascendfactory-cli 方式启动（推荐）

相对于之前**demo.sh方式启动（历史版本）**的启动方式，本章节新增了通过benchmark工具启动训练的方式。此方式训练完成后json日志或打屏日志直接打印性能结果，免于计算，方便用户验证发布模型的质量。并且新的训练方式将统一管理训练日志、训练结果和训练配置，使用yaml配置文件方便用户根据自己实际需求进行修改。

权重文件支持以下组合方式，用户根据自己实际要求选择：

训练stage	不加载权重	增量训练：加载权重，不加载优化器	断点续训：加载权重+优化器
sft、dpo	model_name_or_path=xxx train_from_scratch=true	model_name_or_path=xxx 【lora】 adapter_name_or_path=xxx	model_name_or_path=xxx 【lora】 adapter_name_or_path=xxx resume_from_checkpoint=xxx 或者： overwrite_output_dir=false
rm, ppo	model_name_or_path=xxx train_from_scratch=true	model_name_or_path=xxx 【lora】 adapter_name_or_path=xxx	xxx

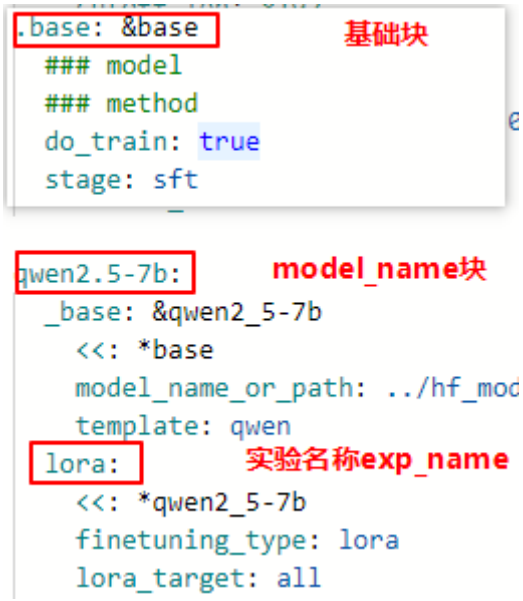
步骤一：准备工作

1. 完成**准备工作**内容，生成ascendfactory-cli工具。
2. 创建test-benchmark目录，该目录存放训练生成的权重文件及训练日志。
```
# 任意目录创建
mkdir test-benchmark
```
3. 修改config目录下yaml文件中model_name_or_path、dataset_dir和dataset或eval_dataset参数或其它内容，根据实际情况选择yaml文件；参数详解可参考**表4-61**。

   config目录结构，请根据实际要求选择、修改文件：
```
|——AscendFactory/examples/config/  # config配置文件
  |——performance_cfgs.yaml        # 微调性能配置yaml文件
  |——llama_factory_performance_cfgs_VL.yaml  # qwen2vl微调yaml配置文件
  |——accuracy_cfgs.yaml           # 训练精度配置yaml文件
  |——llama_factory_cfgs_posttrain.yaml  # RM、PPO、DPO训练阶段样例yaml文件
  |——llama_factory_performance_baseline.yaml # 性能基线配置
  |——llama_factory_accuracy_baseline.yaml    # 精度基线配置
```
   修改样例如下，根据自己实际要求修改相应yaml文件：
```
# 默认参数；根据自己实际要求修改
dataset_dir: /xxxx/benchmark/data/dataset
dataset: gsm8k_train_alpaca
model_name_or_path: /data/wulan1/model/qwen2.5-7b
## accuracy_cfgs.yaml
eval_dataset: gsm8k_test
```
   样例yaml配置文件结构分为

- base块：基础配置块。
- ModelName块：该模型所需配置的参数，如qwen2.5-7b块。
- exp_name：实验块，训练策略-序列长度所需参数配置。

样例yaml文件仅展示常用实验配置，如需其他配置需根据样例自行添加，样例截图如下：



步骤二：执行训练任务

1. 进入test-benchmark目录执行训练命令，可以多次执行，卡数及其它配置参考**NPU卡数取值表**按自己实际情况决定

   单机<可选>：

   ```
   # 默认8卡
   ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
   # 指定设备卡数，如2卡
   ASCEND_RT_VISIBLE_DEVICES=0,1 ascendfactory-cli train <cfgs_yaml_file> <model_name>
   <exp_name>
   # 指定修改yaml中某个参数内容，如save_steps等值，使用超参命令传递形式：
   ASCEND_RT_VISIBLE_DEVICES=0,1 ascendfactory-cli train <cfgs_yaml_file> <model_name>
   <exp_name> --save_steps=5 --max_steps 100
   ```

   多机<可选>多机同时执行：

   ```
   # 默认yaml值
   ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --
   num_nodes <nodes> --rank <rank>
   # 指定修改yaml中某个参数内容，如save_steps等值，使用超参命令传递形式：
   ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --
   num_nodes <nodes> --rank <rank>  --save_steps=5  --max_steps 100
   ```

   - <cfgs_yaml_file>：性能或精度测试配置的yaml文件地址，如**代码目录**中performance_cfgs.yaml、accuracy_cfgs.yaml相对或绝对路径，根据自己要求执行
   - <model_name>：训练模型名，如qwen2-7b
   - <exp_name>：实验名称：具体可以设置的值参考<cfgs_yaml_file>
   - --master_addr <master_addr>：主master节点IP，一般选rank0为主master。
   - --num_nodes <nodes>：训练节点总个数

- --rank <rank>：节点ID，从0开始，一般选rank0为主master。
- --超参<key>：参数key可参考**Yaml配置文件参数配置说明**

2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志，如qwen2.5-7b日志
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt，打印吞吐值及训练参数

```
10/28/2024 16:54:54 - INFO - trainer - pkill -9 npu-smi
10/28/2024 16:54:58 - INFO - trainer - +++++++++ 性能: 3855.0588235294117 tokens/s/p +++++++++

10/28/2024 16:47:46 - INFO - trainer - {'do_train': True, 'stage': 'sft', 'lora_target':
'/data/wulan1/user/lmz/third-party-large-mode-labraries/LLM/LLaMAFactory/benchmark/data/
'overwrite_cache': True, 'preprocessing_num_workers': 16, 'logging_steps': 1, 'save_steps
'include_num_input_tokens_seen': True, 'per_device_train_batch_size': 1, 'gradient_accum
0.1, 'bf16': True, 'ddp_timeout': 180000000, 'flash_attn': 'sdpa', 'model_name_or_path':
'./saves/qwen2.5-7b/sft_lora', 'disable gradient checkpointing': True}
```

   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt，打印训练过程中AICORE利用率

NpuID(Idx)	ChipId(Idx)	Pwr(W)	Temp(C)	AI Core(%)	AI Cpu(%)	Ctrl Cpu(%)	Memory(%)	Memory BW(%)
0	0	93.3	52	0	0	1	5	0
1	0	99.8	50	0	0	2	5	0
2	0	92.8	49	0	0	2	5	0
3	0	93.8	54	0	0	1	5	0
4	0	95.5	51	0	0	0	5	0

说明

- 本章节主要介绍训练性能训练任务流程，如需执行训练精度任务可参考**训练精度测试**
- PPO强化训练时必须关闭共享内存，启动任务命令需设置PYTORCH_NPU_ALLOC_CONF值为False，具体命令如下：
  PYTORCH_NPU_ALLOC_CONF = expandable_segments:False ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>

4.7.3.2 demo.sh 方式启动（历史版本）

本章节介绍历史版本的训练任务启动方式。6.3.912版本同时兼容历史版本的训练任务启动方式。

步骤一：上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

步骤二：修改训练 yaml 文件配置

LlamaFactroy配置文件为Yaml文件，启动训练前需修改Yaml配置文件，Yaml配置文件在代码目录下的{work_dir}/llm_train/AscendFactory/scripts_llamafactory/demo.yaml。修改详细步骤如下所示。

1. 选择训练阶段类型。
   - 指令监督微调，复制**tune_yaml样例模板**内容覆盖demo.yaml文件内容。
   - DPO偏好训练，复制**dpo_yaml样例模板**内容覆盖demo.yaml文件内容。
   - PPO强化训练，先进行**RM奖励训练**任务后，复制**ppo_yaml样例模板**内容覆盖demo.yaml内容。

- RM奖励训练，复制**rm_yaml样例模板**内容覆盖demo.yaml文件内容。

说明

1、DPO偏好训练、Reward奖励模型训练、PPO强化学习目前仅限制支持于llama3系列
2、PPO训练暂不支持ZeRO-3存在通信问题，如llama3-70B使用ZeRO-3暂不支持

2. 训练策略类型
   - 全参full，配置如下：
     ```
     finetuning_type: full
     ```
   - lora，如dpo仅支持此策略；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     ```
   - lora+，目前仅支持qwen1.5-7B**指令监督微调**；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     loraplus_lr_ratio: 16.0
     ```
3. 修改**yaml文件**(demo.yaml)的参数如**表4-60**所示。

表 4-60 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-62**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	**可修改**。指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配。取值可参考**表4-62**中**梯度累积值**列。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配

参数	示例值	参数说明
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/llm_train/AscendFactory/third-party/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

4. 是否选择加速深度学习训练框架Deepspeed，可参考**表4-62**选择不同的框架。
   - 是，选用ZeRO (Zero Redundancy Optimizer)优化器。
     - ZeRO-0，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z0_config.json
       ```
     - ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/AscendFactory/LLaMA-Factory/examples/deepspeed
       ```
       deepspeed: examples/deepspeed/ds_z1_config.json
       ```
     - ZeRO-2，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z2_config.json
       ```
     - ZeRO-2-Offload，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z2_offload_config.json
       ```
     - ZeRO-3，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z3_config.json
       ```
     - ZeRO-3-Offload，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z3_offload_config.json
       ```
   - 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。
5. 是否开启NPU FlashAttention融合算子，具体约束详见**NPU_Flash_Attn融合算子约束**
   - 是，配置以下参数。
     ```
     flash_attn: sdpa
     ```
   - 否，配置以下参数关闭。
     ```
     flash_attn: disabled
     ```
6. 是否使用固定句长。
   - 是，配置以下参数
     ```
     packing: true
     ```
   - 否，默认使用动态句长，**注释**掉packing参数。
7. 选用数据精度格式bf16或fp16二者选一，两者区别可查看**BF16和FP16说明**。
   - bf16，配置以下参数。
     ```
     bf16: true
     ```

- fp16，相比bf16还需配置loss scale参数，配置如下。
  - 设置fp16为True。
    ```
    fp16: true
    ```
  - 修改deepspeed的"loss_scale"参数，配置如下。
    - 修改**ZeRO优化器**配置文件，如ZeRO2命令如下。
      ```
      cd /home/ma-user/AscendFactory/third-party/LLaMA-Factory/examples/deepspeed
      vim ds_z2_config.json
      ```
    - 使用fp16容易出现数值溢出，因此配置loss scale建议配置4096或4096以上：
      ```
      "loss_scale": 4096,
      ```

8. 是否使用自定义数据集。
   - 是，参考**准备数据（可选）**，以指令监督微调数据集为例，配置以下参数：参考**修改重要参数**dataset_dir和dataset参数说明；如alpaca_gpt4_data.json数据集前缀则为alpaca_gpt4_data。
     ```
     dataset: alpaca_gpt4_data
     dataset_dir: /home/ma-user/ws/llm_train/AscendFactory/data
     ```
   - 否，使用代码包自带数据集，注释掉dataset_dir参数，配置参数如下。
     - 指令监督微调/PPO数据集
       ```
       dataset: identity,alpaca_en_demo
       ```
     - 多模态数据集，如qwen2_vl系列模型
       ```
       dataset: mllm_demo,identity
       ```
     - RM/DPO，目前仅支持llama3系列模型
       ```
       dataset: dpo_en_demo
       ```
9. 是否使用falcon-11b、qwen2_vl系列、glm4-9b模型。
   - 是，更新配置或命令。
     - falcon-11b，参考**falcon-11B模型**替换文件。
     - glm4-9b，参考**glm4-9b模型**修改文件内容。
     - qwen2_vl系列，数据集为**多模态数据集**，如果前面步骤已配置请忽略。具体配置如下：

       数据集dataset配置：
       ```
       dataset: mllm_demo,identity
       ```
   - 否，忽略此步骤，执行下一步。
10. 如需其他配置参数，可参考**表4-61**按照实际需求修改。

步骤三：启动训练脚本

修改完yaml配置文件后，启动训练脚本。模型不同最少NPU卡数不同，NPU卡数建议值可参考**表4-62**。

1. **修改启动脚本demo.sh**

   进入代码目录{work_dir}/llm_train/AscendFactory/scripts_llamafactory下**修改启动脚本**，其中{work_dir}为容器挂载路径

   ①是否为PPO强化训练；
   - 是，demo.sh添加变量；
     ```
     export PYTORCH_NPU_ALLOC_CONF = expandable_segments:False
     ```

- 否，demo.sh添加变量，开启虚拟显存；
  ```
  export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
  ```

②修改路径：修改demo.sh最后两行代码，将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/AscendFactory/scripts_llamafactory/demo.yaml，例如将以下命令

- 修改前

```
cd $LLaMAFactory_PATH
FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/AscendFactory/demo.yaml
```

- 修改后：

```
cd $ACS_INSTALL_DIR/third-party/LLaMA-Factory
FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/AscendFactory/scripts_llamafactory/demo.yaml
```

2. **执行多机启动命令**（可选）

   多台机器执行训练启动命令如下。

   ```
   多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
   示例:
   #第一台节点
   sh demo.sh  xx.xx.xx.xx  4  0
   # 第二台节点
   sh demo.sh  xx.xx.xx.xx  4  1
   # 第三台节点
   sh demo.sh  xx.xx.xx.xx  4  2
   # 第四台节点
   sh demo.sh  xx.xx.xx.xx  4  3
   ```

   以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

3. **执行单机启动命令**（可选）

   一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

   进入代码目录**/home/ma-user/ws/llm_train/AscendFactory/scripts_llamafactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

   ```
   # 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
   sh demo.sh localhost 1 0
   ```

   单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
   ```

   其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务。

训练成功标志

“******* train metrics *******”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：训练常见问题**解决。

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：训练常见问题**问题3小节。

3、大模型参数如（qwen2-72B、llama2-70B）等sft训练完成后多线程退出时报“torch.distributed.DistStoreError: Socket Timeout”时请参考**问题4：Error waiting on exit barrier错误**

4、需要开启profiling功能进行性能数据采集和解析请参考**录制Profiling**

5、训练过程中报"ModuleNotFoundError: No module named 'tyro'"可参考**依赖包tyro错误:"ModuleNotFoundError..."**

4.7.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-105 打印训练日志

```
  0%|          | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|▏         | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▎         | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍         | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌         | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋         | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊         | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉         | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█         | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏        | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▍        | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▍        | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▌        | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▋        | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▊        | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|██        | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**修改重要参数**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

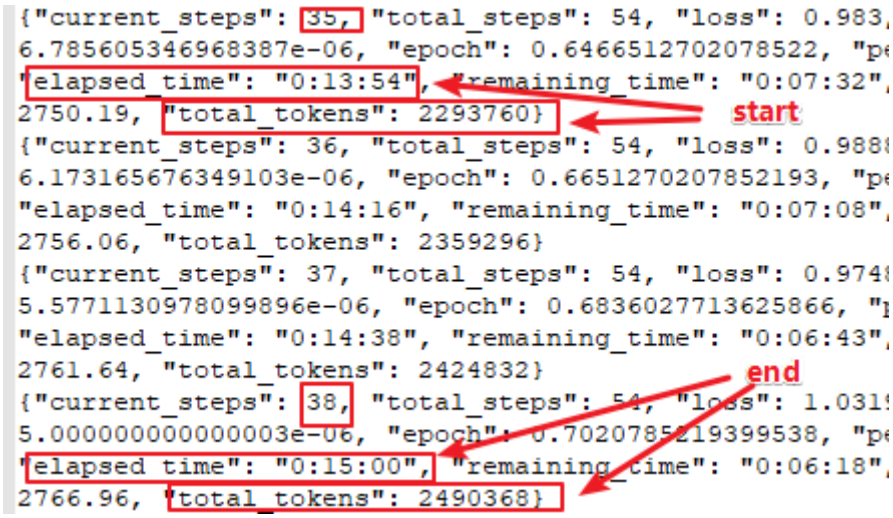
- 吞吐量（tokens/s/p）：可通过**修改重要参数**表格中**output_dir**参数值路径下的**trainer_log.jsonl**计算性能。取中间过程多steps平均值吞吐计算公式为：

  delta_tokens = end_total_tokens-start_ total_tokens

  delta_time = end_elapsed_time - start_elapsed_time

  吞吐值(tps) = delta_tokens / delta_time / 训练卡数

  如图所示：

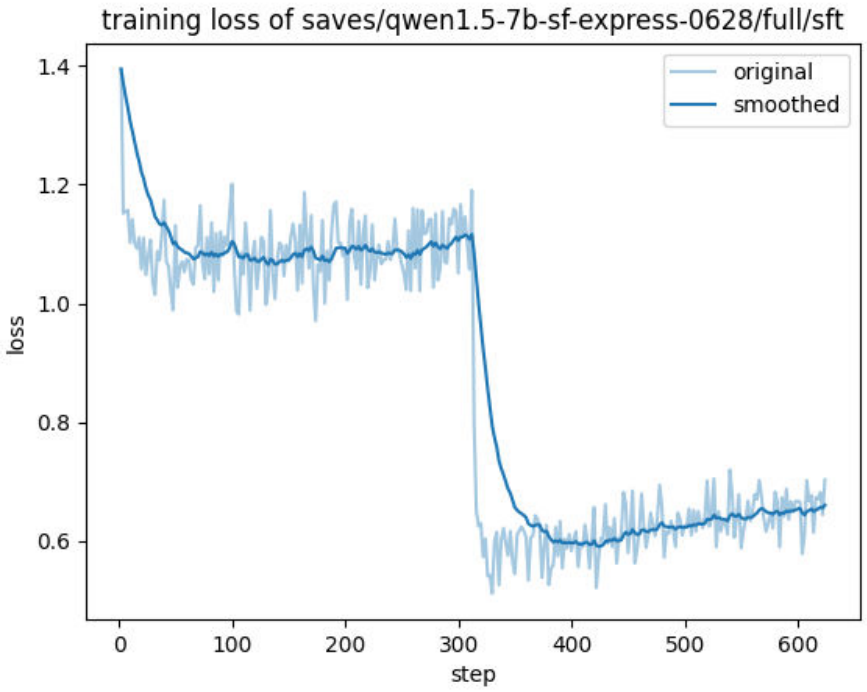
  

- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-60**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，将**trainer_log.jsonl**文件长传至可视化工具页面，如**图4-106**所示。

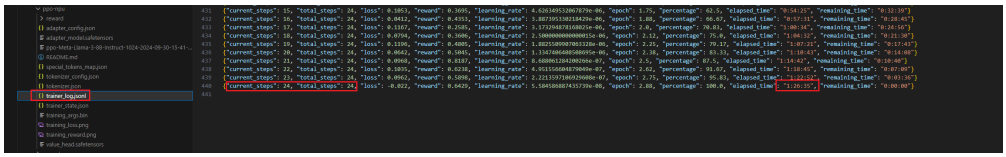
  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

图 4-106 Loss 收敛情况（示意图）



ppo训练结束不会打印性能。建议根据保存路径下的trainer_log.jsonl文件的最后一行总的训练steps和时间来判断性能。



4.7.5 训练 benchmark 工具

4.7.5.1 工具介绍及准备工作

本章节主要介绍针对LLaMAFactory开发的测试工具benchmark，支持训练、性能对比、下游任务评测、loss和下游任务对比能力。对比结果以excel文件呈现。方便用户验证发布模型的质量。所有配置都通过yaml文件设置，用户查看默认yaml文件即可知道最优性能的配置。

说明

目前仅支持SFT指令监督微调训练阶段。

准备工作

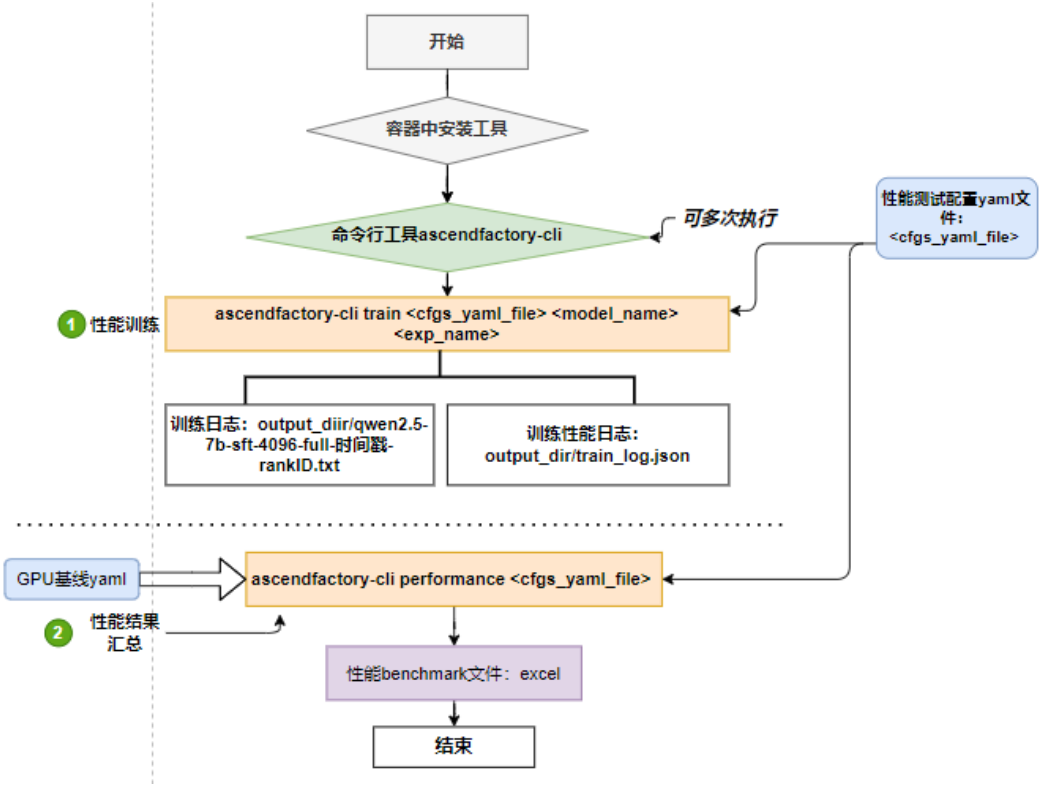
参考**benchmark-准备工作**，开始训练测试，具体步骤参考**训练性能测试**或**训练精度测试**，根据实际情况决定。

4.7.5.2 训练性能测试

流程图

训练性能测试流程图如下图所示：

图 4-107 训练性能测试流程



执行性能比较脚本

1. 完成**benchmark启动**任务。
2. 进入**test-benchmark目录**执行命令。

   ```
   ascendfactory-cli performance <cfgs_yaml_file> --baseline <baseline> --o <output_dir>
   ```

   - <cfgs_yaml_file>：性能测试配置的yaml文件地址，指**代码目录**中performance_cfgs.yaml相对或绝对路径，此配置文件为训练最优配置参数。
   - --baseline <baseline>：<可选>GP-Ant8机器性能基线yaml文件路径，用户可自行修改，不填则使用工具自带基线配置，默认基线配置样例如下：

     ```
     # 性能依次是 A800, ModelLink 313T, ModelLink 400T
     qwen2.5-7b:
         full:    [2380, -1, -1]
         lora:    [3254, -1, -1]
         full-8k: [2394, -1, -1]
         lora-8k: [3199, -1, -1]
     ```

   - --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径。

查看性能结果

任务完成之后会在test-benchmark目录下生成excel表格：

性能结果LLaMAFactory_train_performance_benchmark_<版本号>_<时间戳>.xlsx

表格样例如下：

分类	训练框架	训练类型	序列长度	训练策略	lora_targ	加速框架	训练卡数	GBS	MBS	zero并行	FA	CORE	利用存	利用吐	值(tok	性能值(t	ory VS	都基线(to	vs GP-Ant8
qwen2.5-7	LLaMAFact	SFT	4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	43	51	743.9821	-	-	-	-	
qwen2-7b	LLaMAFact	SFT	4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	38.3	50	508.0854	-	-	-	2500	0.203234

4.7.5.3 训练精度测试

约束限制

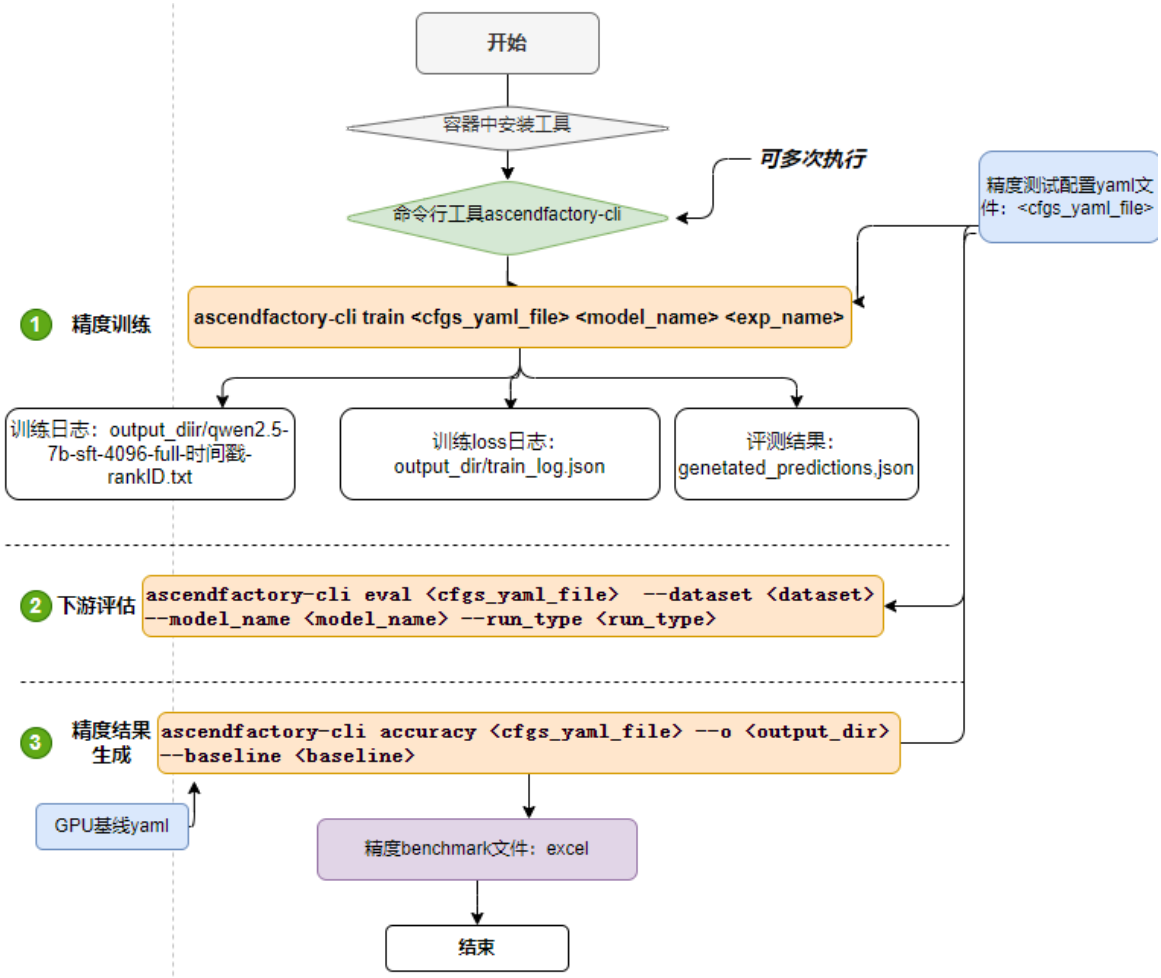
目前仅支持以下模型：

- qwen2.5-7b
- qwen2-7b
- qwen1.5-7b
- llama3.2-3b
- llama3.1-8b
- llama3-8b
- llama2-7b
- yi-6b

流程图

训练精度测试流程图如下图所示。

图 4-108 训练精度测试流程图



执行训练任务

1. 进入**test-benchmark目录**执行训练命令，可以多次执行，按自己实际情况。
   ```
   ascendfactory-cli train <cfgs_yaml_file> <model_name> <exp_name>
   ```
   - <cfgs_yaml_file>：精度测试配置的yaml文件地址，指**代码目录**中accuracy_cfgs.yaml相对或绝对路径
   - <model_name>：训练模型名，如qwen2.5-7b
   - <exp_name>：实验名称：包含训练策略类型及数据序列长度：【lora：4096-lora、full：4096-full】
2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志及权重文件，如qwen2.5-7b日志：
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt

执行下游评估

为增加精度评测的稳定性及进一步确保训练精度，使用多个数据集【MMLU、CEVAL】评测，执行过程如下：

1. 获取到训练权重后使用ascendfactory-cli、eval接口用mmlu、ceval数据集对训练后的结果进行评测

test-benchmark目录目录下执行命令

```
ascendfactory-cli eval <cfgs_yaml_file>  --dataset <dataset>
--model_name <model_name> --run_type <run_type>
```

- <cfgs_yaml_file>：精度评估配置的yaml文件地址，如**代码目录**中accuracy_cfgs.yaml相对或绝对路径
- --dataset <dataset>：评估数据集；可选值：all、mmlu、ceval，默认值为all，用户只需选择参数即可，数据集路径eval接口已指定好。
- --model_name <model_name>：训练模型名
- --run_type <run_type>：训练类型：【full or lora】

2. 执行任务完成后test-benchmark目录下会生成ceval_validation、mmlu_test评估目录，根据自己选择数据集决定包含json及log文件，结果如下：
   - ceval_validation或mmlu_test
     - results.log
     - results.json

⚠ 注意

目前只支持以上超参，客户修改其他参数则需手动修改yaml文件内容

执行精度比较脚本

进入**test-benchmark目录**目录执行命令。

```
ascendfactory-cli accuracy <cfgs_yaml_file> --o <output_dir> --baseline <baseline>
```

- <cfgs_yaml_file>：精度测试配置的yaml文件地址，如**代码目录**中accuracy_cfgs.yaml相对或绝对路径
- --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径
- --baseline <baseline>：<可选>GP-Ant8机器精度基线Yaml文件路径，不填则使用工具自带基线配置，包含loss、score、mmlu_score、ceval_score基线值；默认基线配置样例如下：

```
qwen2-7b:
  full:
    loss: [0.4339,0.1433,0.2185,0.172,0.2376,0.
    score: 0.6815769522365428
    mmlu_score: 69.16
    ceval_score: 81.20
  lora:
    loss: [0.4339,0.3791,0.3035,0.1714,0.1674,0
    score: 0.7490523123578469
    mmlu_score: 68.38
    ceval_score: 80.01
```

说明

客户使用工具自带精度基线Yaml则需使用accuracy_cfgs.yaml文件中默认配置，权重使用**表1 模型权重**中指定的Huggingface地址，数据指定data.tgz里面提供的gsm8k和mmlu、ceval数据。

查看精度结果

任务完成之后会在test-benchmark目录下生成excel表格：

精度结果 LLaMAFactory_train_accuracy_benchmark_<版本号>_<时间戳>.xlsx

样例截图：


benchmark示例：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	warmup学习率衰减策略	数据集	MBS	GBS	平均相对误差	平均绝对误差	loss绝对误差	推理评分差	NPU推理评分	GP-A推理评分
qwen2-7b	lora	233	adamw	0.0002	0	0 cosine	gsm8k_trs	1	32	0.418063	0.016463	0.0162	0.025018	0.728582	0.7536

具体精度benchmark包含如下字段：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	学习率warmup ratio	学习率衰减策略	数据集	MBS	GBS	loss平均相对误差	loss平均绝对误差	step1 loss绝对误差	推理评分差	NPU推理评分	GP-A推理评分

4.7.6 训练脚本说明

4.7.6.1 Yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-61 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/model/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改
adapter_name_or_path	/home/ma-user/ws/xxx/sft_lora/	基于lora训练完成后生成的lora产物未合并时的权重文件。基于lora微调后模型进行**增量训练时，需要**传入此权重文件。
train_from_scratch	false	用于指示模型是否从头开始训练，如果**true**模型将从一个全新的初始状态开始训练则**不加载权重**。【 true or false 】，默认false
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能

参数	示例值	参数说明
stage	sft	表示当前的训练阶段。可选择值：【sft、rm、ppo、dpo】 • sft代表监督微调； • rm代表奖励模型训练； • ppo代表PPO训练； • dpo代表DPO训练。
finetuning_type	full	用于指定微调策略类型，可选择值full、lora。 如果设置为full，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
lora_target	all	采取lora策略方法的目标模块，默认为all
dataset	• 指令微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo，identity	**【可选】** 注册在dataset_info.json文件数据集名称。如选用**自定义数据**则需配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下，详情可参考**自定义数据**。
dataset_dir	/home/ma-user/AscendFactory/data	**【可选】** 代码自带数据：如identity等地址为AscendFactory/third-party/LLaMA-Factory/data目录下； **自定义数据**：地址为代码包AscendFactory/data目录下
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表4-62**中的**template列**
max_samples	50000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。

参数	示例值	参数说明
per_device_train_batch_size	1	指定每个设备的训练批次大小。
gradient_accumulation_steps	8	**必须修改**，指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可参考**表4-62**
output_dir	/home/ma-user/ws/saves/qwen2-7b/sft_lora/	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
max_steps	5000	非必填。表示训练step迭代次数。会自动计算得出。
save_steps	5000	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练。 • 当参数值>=max_steps时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<max_steps时，生成模型会每经过save_steps次，保存一次模型版本。 模型版本保存次数=max_steps//save_steps+1
save_total_limit	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=max_steps//save_steps+1 • 当参数值>1时，保存模型版本次数与save_total_limit的值一致。
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时清空输出目录。“false”加载中断时最新保存训练权重继续训练，开启【**故障快恢**】。如需指定可设置resume_from_checkpoint参数。默认为true
resume_from_checkpoint	{output_dir}/checkpoint-xx	【**断点续训**】加载训练中断时某个checkpoint-xx权重目录，目的方便用户指定权重基于此权重继续训练。此参数优先于overwrite_output_dir参数。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。

参数	示例值	参数说明
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	2.0e-5	指定学习率
disable_gradient_checkpointing	true	关闭重计算，用于禁用梯度检查点，**默认开启**梯度检查点;在深度学习模型训练中用于保存模型的状态，以便在需要时恢复。这种技术可以帮助减少内存使用，特别是在训练大型模型时，但同时影响性能。True表示关闭重计算功能。
include_tokens_per_second include_num_input_tokens_seen	true	用于在训练过程中包含每秒处理的tokens和已经看到的输入tokens，方便计算性能。
reward_mode	/home/ma-user/ws/saves/tune/Qwen2-72B/sft	**PPO强化必修改**；指定Reward奖励任务完成时output_dir目录，PPO强化训练前提为完成Reward奖励学习；请根据实际规划修改。
loraplus_lr_ratio	16.0	lora+策略算法独有参数；设置Lora+算法的lambda值为16.0

tune_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/model/Qwen2-72B
### method
stage: sft
do_train: true

# 全参
finetuning_type: full

# lora
# finetuning_type: lora
# lora_target: all

deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/AscendFactory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/tune/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
```

```
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

dpo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/model/Qwen2-72B
### method
stage: dpo
do_train: true

# lora
finetuning_type: lora
lora_target: all

pref_beta: 0.1
pref_loss: sigmoid
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: dpo_en_demo
dataset_dir: /home/ma-user/ws/llm_train/AscendFactory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/dpo/llama3-8b/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 5.0e-6
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ppo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/model/llama3-8b
reward_model: /home/ma-user/ws/saves/rm/llama3-8b/lora
### method
stage: ppo
do_train: true

# 全参
# finetuning_type: full
# reward_model_type: full

# lora
```

```
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: identity,alpaca_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/ppo/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000
flash_attn: sdpa
include_tokens_per_second: true
include_num_input_tokens_seen: true
### generate
max_new_tokens: 512
top_k: 0
top_p: 0.9
```

rm_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/model/llama3-8b
### method
stage: rm
do_train: true

# 全参
# finetuning_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: dpo_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/rm/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
```

```
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
  }
}
```

4.7.6.2 模型 NPU 卡数、梯度累积值取值表

不同模型推荐的训练参数和计算规格要求如表4-62所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推。

表 4-62 NPU 卡数、加速框架、梯度配置取值表

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
llama2	llama2	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		13B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3	llama3	70B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full		gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
llama3.1	llama3	8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	4096	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3.2	llama3	1B	lora	4096/8192	gradient_accumulation_steps: 32	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 16	ZeRO-1	1*节点 & 1*Ascend
		3B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
Qwen2	qwen	72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		0.5B	lora	*4096*/8192	gradient_accumulation_steps: 32	ZeRO-1	1*节点 & 1*Ascend
			full	*8192*	gradient_accumulation_steps: 32	ZeRO-1	1*节点 & 1*Ascend
Qwen2_vl	qwen2_vl	2B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 2*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		72B	lora	1024	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	1024	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
Qwen1.5	qwen	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
			full	8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full	8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
Qwen2.5	qwen	0.5B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
falcon2	falcon	11B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
GLM4	glm4	9B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
Yi	yi	6B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
		34B	full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
			full	8192	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

说明

以上参数为开启NPU FlashAttention融合算子，上述参数值仅供参考，请根据自己实际要求合理配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器、NPU节点数及其他配置。

具体优化工具使用说明可参考**如何选择最佳性能的zero-stage和-offloads**。

4.7.6.3 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入{work_dir}/...../ascendcloud_patch/models/falcon2目录下：

```
cd /home/ma-user/ws/llm_train/AscendFactory/src/acs_train_solution/ascendcloud_patch/models/falcon2
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/model/falcon-11B/
```

glm4-9b 模型

在训练开始前，需要修改glm4-9b模型中的tokenizer文件modeling_chatglm.py内容，具体步骤如下：

- 进入到tokenizer目录下{work_dir}/model/glm4-9B/，命令如下：

```
cd /home/ma-user/ws/model/glm4-9B
```

- 修改modeling_chatglm.py文件内容：

```
vim modeling_chatglm.py
# 注释掉以下两行内容
# if attention_mask is not None
    # attention_mask = ~attention_mask
```

样例图：

```
268 class SdpaAttention(CoreAttention):
269     def forward(self, query_layer, key_layer, value_layer, atte
270         if attention_mask is None and query_layer.shape[2] == k
271             context_layer = torch.nn.functional.scaled_dot_prod
272
273
274         else:
275             # if attention_mask is not None:
276                 # attention_mask = ~attention_mask
```

4.7.6.4 NPU_Flash_Attn 融合算子约束

1. query、key、value都需要梯度。默认开启重计算，则前向时qkv没有梯度，如果需要关闭重计算，可以在yaml配置 `disable_gradient_checkpointing: true` 关闭，但显存占用会直线上升。
2. attn_mask只支持布尔（bool）数据类型，或者为None。
3. query的shape仅支持 [B, N1, S1, D]，其中N1≤ 2048，D≤ 512并且dim== 4。
4. 对于GQA，key的shape是 [B, N2, S2, D]，其中N2 ≤ 2048，并且N1是N2的正整数倍。

不满足以上场景，则不能实现NPU_Flash_Attn功能。

4.7.6.5 BF16 和 FP16 说明

在大模型训练中，BF16（Brain Floating Point）和FP16（Float16）都是使用的半精度浮点数格式，但它们在结构和适用性上有一些重要的区别。

BF16：具有8个指数位和7个小数位。在处理大模型时有优势，能够避免在训练过程中数值的上溢或下溢，从而提供更好的稳定性和可靠性，在大模型训练和推理以及权重存储方面更受欢迎。

FP16：用于深度学习训练和推理过程中，可以加速计算并减少内存的占用，对模型准确性的影响在大多数情况下较小。与BF16相比在处理非常大或非常小的数值时遇到困难，导致数值的精度损失。

综上所述，BF16因其与FP32相似的数值范围和稳定性，在大模型训练中提供了优势。而FP16则在计算效率和内存使用方面有其独特的优点，但可能在数值范围和稳定性方面略逊一筹。因此，选择哪种格式取决于具体的应用场景和训练需求。

4.7.6.6 录制 Profiling

Ascend PyTorch Profiler是针对PyTorch框架开发的性能数据采集和解析工具，通过在PyTorch训练脚本中插入Ascend PyTorch Profiler接口，执行训练的同时采集性能数据，完成训练后直接输出可视化的性能数据文件，提升了性能分析效率。

Ascend PyTorch Profiler接口可全面采集PyTorch训练场景下的性能数据，主要包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等，可以全方位分析PyTorch训练时的性能状态。

录制命令如下：

在启动训练脚本基础：**启动训练脚本**新加DO_PROFILER=1和PROF_SAVE_PATH=/save_path参数，单机启动举例说明：

```
DO_PROFILER=1 PROF_SAVE_PATH=/save_path sh demo.sh localhost 1 0
```

- PROF_SAVE_PATH：Profiling录制结果存放路径
- DO_PROFILER：是否开启Profiling录制功能

4.7.7 附录：训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

1. 容器内执行以下命令，指定NPU内存分配策略的环境变量，开启动态内存分配，即在需要时动态分配内存，可以提高内存利用率，减少OOM错误的发生。
   ```
   export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True
   ```
2. 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。
3. 替换深度学习训练加速的工具或增加zero等级，可参考**模型NPU卡数、梯度累积值取值表**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。
   a. - ZeRO-0 数据分布到不同的NPU
   b. - ZeRO-1 Optimizer States分布到不同的NPU
   c. - ZeRO-2 Optimizer States、Gradient分布到不同的NPU

d. - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU

4. 增加卡数重新训练，未解决找相关人员定位。

问题 2：访问容器目录时提示 Permission denied

解决方法：

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: XXX not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

问题 4：Error waiting on exit barrier 错误

错误截图：

```
[ERROR] Error waiting on exit barrier. Elapsed: 300.106763912200g seconds
[ERROR] Traceback (most recent call last):
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py

[ERROR]     store_util.barrier(
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     synchronize(store, data, rank, world_size, key_prefix, barrier_timeout)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li

[ERROR]     agent_data = get_all(store, rank, key_prefix, world_size)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     store_get(f"{prefix}{node_rank}_FIN")
[ERROR] torch.distributed.DistStoreError: Socket Timeout
```

报错原因：多线程退出各个节点间超时时间默认为300s，时间设置过短。

解决措施：

修改容器内torch/distributed/elastic/agent/server/api.py文件参数：

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/
agent/server/api.py
```

修改def _exit_barrier(self)方法中的barrier_timeout参数，修改后如**图4-109**所示。

```
#修改前
barrier_timeout=self._exit_barrier_timeout
#修改后
barrier_timeout=3000
```


图 4-109 修改后的 barrier_timeout 参数

```
913     def _exit_barrier(self):
914         """
915         Define a barrier that keeps the agent process alive until all workers finish.
916
917         Wait for ``exit_barrier_timeout`` seconds for all agents to finish
918         executing their local workers (either successfully or not). This
919         acts as a safety guard against user scripts that terminate at different
920         times.
921         """
922         log.info(
923             "Local worker group finished (%s). "
924             "Waiting %s seconds for other agents to finish",
925             self._worker_group.state, self._exit_barrier_timeout
926         )
927         start = time.time()
928         try:
929             store_util.barrier(
930                 self._store,
931                 self._worker_group.group_rank,
932                 self._worker_group.group_world_size,
933                 key_prefix= TERMINAL_STATE_SYNC_ID,
934                 barrier_timeout=3000,
935             )
936             log.info(
937                 "Done waiting for other agents. Elapsed: %s seconds", time.time() - start
938             )
939         except SignalException as e:
940             log.warning("Got termination signal: %s", e.sigval)
941             raise
942         except Exception:
943             log.exception(
944                 "Error waiting on exit barrier. Elapsed: %s seconds",
945                 time.time() - start
946             )
```

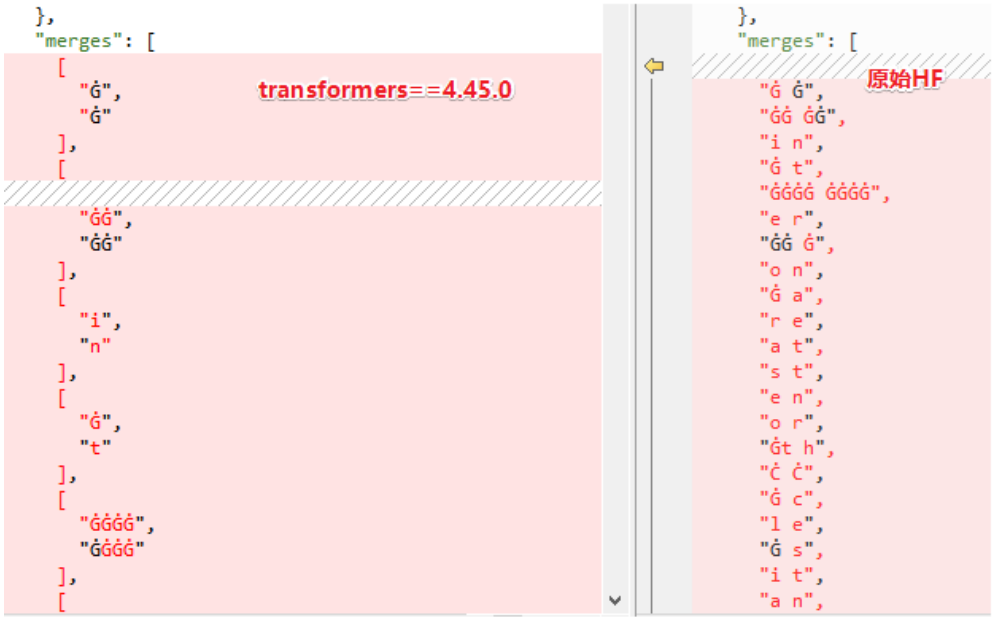
问题 5：训练完成使用 vllm0.6.0 框架推理失败：

错误截图：

```
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/tokenization_u
tils_fast.py", line 115, in __init__
    fast_tokenizer = TokenizerFast.from_file(fast_tokenizer_file)
Exception: data did not match any variant of untagged enum ModelWrapper at line 757272 column 1
```

报错原因：

训练时transformers版本要求为4.45.0，训练完成后保存的tokenizer.json文件中的“merges”时保存的是拆开的列表不是字符串，导致推理异常



解决措施，以下两种方法任选其一：

1. 更新transformes和tokenizers版本
   - GLM4-9B模型，容器内执行以下步骤：
     ```
     pip install transformers==4.43.2
     ```
   - 其它模型，容器内执行以下步骤：
     ```
     pip install transformers==4.45.0
     pip install tokenizers==0.20.0
     ```
2. 使用原始hf权重的tokenizer.json覆盖保存的tokenizer.json即可，如llama3-8b_lora具体过程如下：
   ```
   # 进入模型tokenizer目录
   cd /home/ma-user/ws/tokenizers/llama3-8b/
   # 替换tokenizer.json文件
   cp -f tokenizer.json /home/ma-user/ws/saves/rm/llama3-8b/lora/tokenizer.json
   ```

问题 6：训练过程中报依赖包 tyro 错误:"ModuleNotFoundError: No module named 'tyro'"

错误截图：

```
ModuleNotFoundError: No module named 'tyro'
```

报错原因：未指定tyro依赖包版本，导致安装依赖为最新0.9.0版本导致与其他依赖冲突

解决措施：任务前容器内更新'tyro'版本为0.8.14或以下版本

```
pip install tyro==0.8.14
```

问题 7：训练过程中报“an exception occurred : ('copy_d2d:build/xxx NPU function error”

错误截图：

```
|:    While copying the parameter named "base_model.model.model.layers.63.mlp.up_proj.lora_B.default.weight", whose di
3, 8]), an exception occurred : ('copy_d2d:build/CMakeFiles/torch_npu.dir/compiler_depend.ts:263 NPU function error:
ROR] 2025-01-11-21:04:30 (PID:1461, Device:1, RankID:1) ERR00100 PTA call acl api failed\n[Error]: The vector core exe
 Inner Error!\n          rtStreamSynchronizeWithTimeout execute failed, reason=[vector core exception][FUNC:FuncError
33 synchronize stream failed, runtime result = 507035[FUNC:ReportCallError][FILE:log_inner.cpp][LINE:161]\n
```

报错原因：开启虚拟内存导致，虚拟内存不兼容某些训练场景如PPO、基于lora微调增量训练等

解决措施：关闭虚拟内存

使用历史版本demo.sh启动训练时，任务前容器中执行以下命令：

```
# 历史版本demo.sh启动,:
export PYTORCH_NPU_ALLOC_CONF = expandable_segments:False
```

使用ascendfactory-cli方式启动训练时，命令行参数新加以下环境变量：

```
PYTORCH_NPU_ALLOC_CONF = expandable_segments:False ascendfactory-cli train <cfgs_yaml_file>
<model_name> <exp_name> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>
```

4.8 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.911）

4.8.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.3版本。
- 支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc3。
- DevServer驱动版本要求23.0.6。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的DevServer。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用DevServer资源，请参考**DevServer资源开通**，购买DevServer资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-63 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-64**所示。

表 4-64 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.3版本。不同vLLM版本支持的模型列表有差异，具体如**表4-65**所示。

表 4-65 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**
3	llama-65b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-65b**
4	llama2-7b	√	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
5	llama2-13b	√	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-34b-hf/tree/main
57	internvl2-8B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-8B/tree/main
58	internvl2-26B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-26B/tree/main
59	internvl2-40B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-40B/tree/main
60	internVL2-Llama3-76B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B/tree/main
61	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
62	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
63	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
64	qwen2-vl-2B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
65	qwen2-vl-7B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
66	qwen2-vl-72B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct/tree/main
67	qwen-vl	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL
68	qwen-vl-chat	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL-Chat

69	MiniCPM-v2	√	x	x	x	x	https://huggingface.co/HwwwH/MiniCPM-V-2
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation, antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.3-py3-none-any.whl   # 推理安装包
       ├── build.sh               # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile              # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
   ├──AutoSmoothQuant  # W8A8量化工具
       ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
       ├── autosmoothquant_ascend          # 量化代码
       ├── build.sh                        # 安装量化模块的脚本
   ├──AutoAWQ             # W4A16量化工具
       ├──convert_awq_to_npu.py   # awq权重转换脚本
       ├──quantize.py   # 昇腾适配的量化转换脚本
       ├──build.sh      # 安装量化模块的脚本
   ├──llm_evaluation    # 推理评测代码包
     ├──benchmark_tools    #性能评测
       ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
       ├── benchmark_parallel.py   # 评测静态性能脚本
       ├── benchmark_serving.py    # 评测动态性能脚本
       ├── benchmark_utils.py      # 抽离的工具集
       ├── generate_datasets.py    # 生成自定义数据集的脚本
       ├── requirements.txt        # 第三方依赖
     ├──benchmark_eval #精度评测
       ├──opencompass.sh       #运行opencompass脚本
       ├──install.sh            #安装opencompass脚本
       ├──vllm_api.py          #启动vllm api服务器
       ├──vllm.py             #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型（PyTorch）基于DevServer训练指导**。

4.8.2 部署推理服务

4.8.2.1 非分离部署推理服务

本章节介绍如何使用vLLM 0.6.3框架部署并启动推理服务。

什么是非分离部署

全量推理和增量推理在同一节点上进行。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。

- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-63**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.911-xxx.zip和算子包AscendCloud-OPP-6.3.911-xxx.zip到主机中，包获取路径请参见**表4-64**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-65**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下。
```
df -h
```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.911-xxx.zip和算子包AscendCloud-OPP-6.3.911-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。
```
unzip AscendCloud-*.zip -d ./AscendCloud && cd ./AscendCloud && unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-2.1.0*.zip -d ./AscendCloud-OPP && cd .. && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

步骤五 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

步骤六 启动推理服务

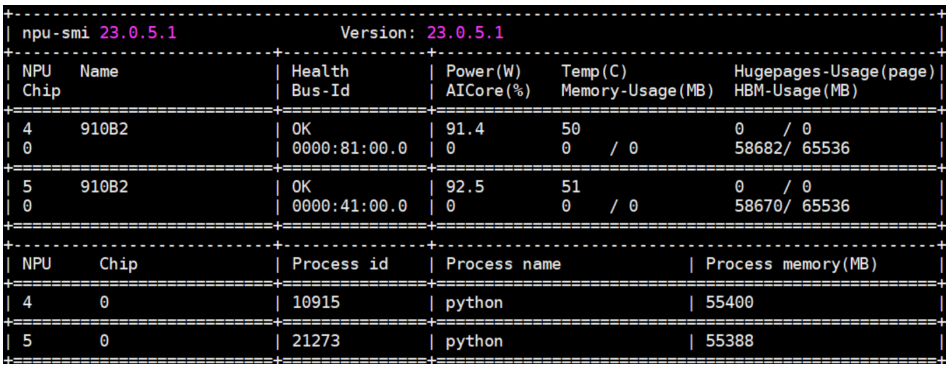
1. 进入容器。
```
docker exec -it -u ma-user ${container-name} /bin/bash
```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
```
npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-110 查询结果



4. 配置环境变量。
```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```
5. 如果要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量unset INFER_MODE
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

在MoE模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

6. 如果要使用eagle投机，配置环境变量，使eagle投机对齐论文版本实现。目前默认开启此模式，如果不开启，目前vllm0.6.3版本与实验室版本权重无法对齐，会导致小模型精度问题。

   ```
   export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现，默认开启
   export ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
   ```

   如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE 文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

   ```
   python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
   ```

   具体可参考章节eagle投机小模型训练中的**步骤五：训练生成权重转换成可以支持vLLM推理的格式**。

7. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理

8. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**【推荐，在vllm-0.6.0之后的版本性能更好】

     在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     （1）非多模态

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code \
--enforce-eager
```

（2）多模态

当前支持Llava、InternVL2、MiniCPM、qwen2-vl模型，具体模型和权重地址参见**表4-65**，推荐显卡数量参见**表4-71**。

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：

  （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
```

```
--trust-remote-code \
--enforce-eager
```

- **方式三：多机部署vLLM服务API接口启动服务**（可选）

  当单机显存无法放下模型权重时，可选用该种方式部署；该种部署方式，需要机器在同一个集群，NPU卡之间IP能够ping通方可，具体步骤如下：

  i. 查看卡IP。
  ```
  for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
  ```

  ii. 检查卡之间的网络是否通。
  ```
  # 在另一个节点上执行，29.81.3.172是上一步输出的ipaddr的值
  hccn_tool -i 0 -ping -g address 29.81.3.172
  ```

  iii. 启动Ray集群。
  ```
  # 指定通信网卡，使用ifconfig查看，找到和主机IP一致的网卡名
  export GLOO_SOCKET_IFNAME=enp67s0f5
  export TP_SOCKET_IFNAME=enp67s0f5
  export RAY_EXPERIMENTAL_NOSET_ASCEND_RT_VISIBLE_DEVICES=1
  # 指定可使用的卡
  export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
  # 将其中一个节点设为头节点
  ray start --head --num-gpus=8
  # 在其他节点执行
  ray start --address='10.170.22.18:6379' --num-gpus=8
  ```

  - --num-gpus：要跟ASCEND_RT_VISIBLE_DEVICES指定的可用卡数一致。
  - --address：头节点IP+端口号，头节点创建成功后，会有打印。

  iv. 正常启服务即可。

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启投机推理后无需配置该参数，否则会导致投机推理启动报错。

- --multi-step-stream-outputs: 设置false后，mult-step会关闭流式输出提升性能，一次将返回num-scheduler-steps个token。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。
- --enforce-eager：未设置INFER_MODE环境变量时，部分模型会默认使用CANNGraph图模式启动来提升性能，设置该参数后将关闭图模式。CANNGraph图模式目前支持llama和qwen2系列大语言模型单卡场景，包含该系列AWQ量化模型，其他场景(如Multi-lora)暂未支持。小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
- --disable-async-output-proc：关闭异步后处理特性，关闭后性能会下降。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。
  ```
  export USE_PREFIX_HIGH_PRECISION_MODE=1
  ```
- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。
  ```
  --enable-lora \
  --lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
  --max-lora-rank=16 \
  --max-loras=32 \
  --max-cpu-loras=32
  ```
  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --enable-chunked-prefill：chunked-prefill特性参数，不传入默认为None即不启用；在不定长、短输入、短输出、大并发场景下推荐使用，需配合--max-num-batched-tokens使用，建议max-num-batched-tokens设置为2048或更大。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，

但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。

- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:
Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on
http://0.0.0.0:8080 (Press CTRL+C to quit)
```

步骤七 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-66**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
   "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
```

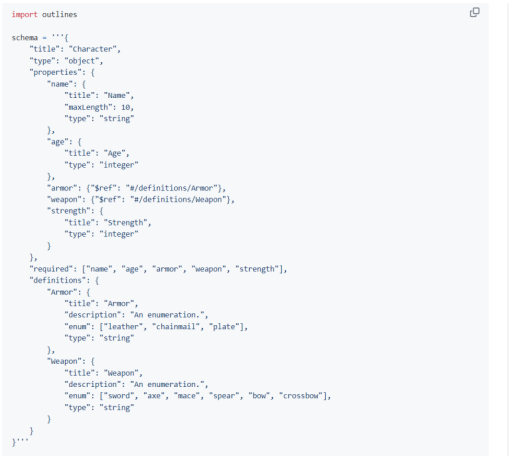
```
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-66 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-111** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
import json
from typing import List
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

def get_stop_token_ids(model_path):
    with open(f"{model_path}/config.json") as file:
        data = json.load(file)
        if data.get('architectures')[0] == "InternVLChatModel":
            return [0, 92543, 92542]
    return None

def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    stop_token_ids = args.stop_token_ids if args.stop_token_ids is not None else
get_stop_token_ids(args.model_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
```

```
        "max_tokens": args.max_tokens,
        "temperature": args.temperature,
        "ignore_eos": args.ignore_eos,
        "stream": args.stream,
        "top_k": args.top_k,
        "top_p": args.top_p,
        "stop_token_ids": stop_token_ids,
        "repetition_penalty": args.repetition_penalty,
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成
EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的
文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越
意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启
流式推理。
    parser.add_argument("--max-tokens", type=int, default=16) # 生成序列的最大长度。必选
    parser.add_argument("--repetition-penalty", type=float, default=1.0) # 减少重复生成文本的概率。可选
    parser.add_argument("--stop-token-ids", nargs='+', type=int, default=None) # 停止tokens列表。可选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip $
{docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-67 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-68 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。
repetition_penalty	否	1.0	Float	减少重复生成文本的概率。
stop_token_ids	否	None	Int	停止tokens列表。internvl2和minicpmv需要传入，参考离线推理脚本examples/offline_inference_vision_language.py的stop_token_ids。

4.8.2.2 分离部署推理服务

本章节介绍如何使用vLLM 0.6.3框架部署并启动推理服务。

什么是分离部署

大模型推理是自回归的过程，有以下两阶段：

- Prefill阶段（全量推理）

  将用户请求的prompt传入大模型，进行计算，中间结果写入KVCache并推出第1个token，属于计算密集型。

- Decode阶段（增量推理）

  将请求的前1个token传入大模型，从显存读取前文产生的KVCache再进行计算，属于访存密集型。

分离部署场景下，全量推理和增量推理在不同的容器上进行，用于提高资源利用效率。

分离部署的实例类型启动分为以下三个阶段：

1. **步骤六 启动全量推理实例**：必须为NPU实例，用于启动全量推理服务，负责输入的全量推理。全量推理占用至少1个容器。
2. **步骤七 启动增量推理实例**：必须为NPU实例，用于启动增量推理服务，负责输入的增量推理。增量推理占用至少1个容器。
3. **步骤八 启动scheduler实例**：可为CPU实例，用于启动api-server服务，负责接收推理请求，向全量或增量推理实例分发请求，收集推理结果并向客户端返回推理结果。服务调度实例不占用显卡资源，建议增加1个容器，也可以在全量推理或增量推理的容器上启动。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-63**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.911-xxx.zip和算子包AscendCloud-OPP-6.3.911-xxx.zip到主机中，包获取路径请参见**表4-64**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-65**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

   3.权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：

   ```
   df -h
   ```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.911-xxx.zip和算子包AscendCloud-OPP-6.3.911-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && cd ./AscendCloud && unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-2.1.0*.zip -d ./AscendCloud-OPP && cd .. && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

如果推理需要使用npu加速图片预处理，需要安装torchvision_npu，可放到镜像制作脚本里面。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

步骤五 生成 ranktable

介绍如何生成ranktable，以1p1d-tp2分离部署模式为例。当前1p1d分离部署模式，全量节点和增量节点分别占用2张卡，一共使用4张卡。

- 配置tools工具根目录环境变量

  使用AscendCloud-LLM发布版本进行推理，基于AscendCloud-LLM包的解压路径配置tool工具根目录环境变量：

  ```
  export LLM_TOOLS_PATH=${root_path_of_AscendCloud-LLM}/llm_tools
  ```

其中，`${root_path_of_AscendCloud-LLM}`为AscendCloud-LLM包解压后的根路径。

当使用昇腾云的官方指导文档制作推理镜像时，可直接基于该固定路径配置环境变量：

```
export LLM_TOOLS_PATH=/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools
```

- 获取每台机器的rank_table

  在每个机器生成global rank_table信息与local rank_table信息。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 4,5 --decode-server-list 6,7 --api-server --save-dir ./save_dir
  ```

  执行后，会生成一个global_ranktable.json文件和使用实例个数的local_ranktable.json文件；如果指定了`--api-server`，还会生成一个local_ranktable_host.json文件用于确定服务入口实例。

  ./save_dir 生成ranktable文件如下（假设本地主机ip为10.**.**.18）。

  ```
  global_ranktable_10.**.**.18.json      # global rank_table
  local_ranktable_10.**.**.18_45.json    # 全量节点local rank_table
  local_ranktable_10.**.**.18_67.json    # 增量节点local rank_table
  local_ranktable_10.**.**.18_host.json  # api-server
  ```

  如果要启动多P多D服务，则需要修改--prefill-server-list和--decode-server-list参数，每个实例之间用空格隔开，例如2p2d-tp2：

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 0,1 2,3 --decode-server-list 4,5 6,7 --api-server --save-dir ./save_dir
  ```

- 合并不同机器的global rank_table(可选)

  如果分离部署在多台机器，获取每台机器的rank_table后，合并各个机器的global rank_table得到完整的global rank_table。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode merge --global-ranktable-list ./ranktable/global_ranktable_0.0,0,0.json ./ranktable/global_ranktable_1.1.1.1.json --save-dir ./save_dir
  ```

  pd_ranktable_tools.py的入参说明如下。

  - --mode：脚本的处理模式，可选值为gen或者merge。gen模式表示生成rank_table文件，merge模式表示合并global rank_table文件。
  - --save-dir：保存生成的rank_table文件的根目录，默认为当前目录。
  - --api-server：仅在`gen`模式有效，可选输入，当存在该输入时，表示分离部署的服务入口在该机器。注意，在多台机器启动分离部署时，只能有一台机器存在服务入口。当存在该输入时，会生成local_ranktable_xx_host.json文件，用于在启动推理服务时确定服务入口实例。
  - --prefill-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm全量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应全量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定全量实例。
  - --decode-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm增量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应增量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定增量实例。
  - --global-ranktable-list：仅在`merge`模式有效，必选输入，后续入参表示需要合并的global rank_table，使用空格分隔开。

  执行后，会生成完成合并的global_ranktable_merge.json文件。

- global_rank_table.json格式说明

server_group_list的长度必须为3，第一个元素(group_id="0")代表Scheduler实例的ip信息，只能有一个实例。

第二个元素(group_id="1")代表全量实例信息，长度即为全量实例个数。其中需要配置每个全量实例的ip信息以及使用的device信息。rank_id为逻辑卡号，必然从0开始计算，device_id为物理卡号，device_ip则通过上面的hccn_tool获取。

第三个元素(group_id="2")代表增量实例信息，长度即为增量实例个数。其余信息和全量类似。

global_rank_table.json具体示例如下：

```
{
   "version": "1.0",
   "status": "completed",
   "server_group_list": [
      {
         "group_id": "0",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost"
            }
         ]
      },
      {
         "group_id": "1",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "4",
                     "device_ip": "10.**.**.22",
                     "rank_id": "0"
                  },
                  {
                     "device_id": "5",
                     "device_ip": "10.**.**.23",
                     "rank_id": "1"
                  }
               ]
            }
         ]
      },
      {
         "group_id": "2",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "6",
                     "device_ip": "29.**.**.56",
                     "rank_id": "0"
                  },
                  {
                     "device_id": "7",
                     "device_ip": "29.**.**.72",
                     "rank_id": "1"
                  }
               ]
            }
         ]
      }
```

```
  ]
}
```

- local_rank_table.json格式说明

  每个全量/增量实例都需要local_rank_table.json。下面以某一个增量实例为例，需要和global_rank_table.json中的增量信息完全对应，group_id为0。

```
{
   "version": "1.0",
   "status": "completed",
   "group_id": "0",
   "server_count": "1",
   "server_list": [
      {
         "server_id": "localhost",
         "server_ip": "localhost",
         "device": [
            {
               "device_id": "6",
               "device_ip": "29.**.**.56",
               "rank_id": "0"
            },
            {
               "device_id": "7",
               "device_ip": "29.**.**.72",
               "rank_id": "1"
            }
         ]
      }
   ]
}
```

步骤六 启动全量推理实例

以下介绍如何启动全量推理实例。

1. 启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci4、davinci5。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- 如果需要多个全量实例，每个全量都需要启动一个容器，只挂载对应的NPU

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。
```
docker exec -it -u ma-user ${container-name} /bin/bash
```
3. 启动全量推理实例，命令如下。
```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_45.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8088 \
   --served-model-name ${served-model-name}
```
其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量或增量推理实例启动的--port参数相关。--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(`--port`)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量

- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

参数定义和使用方式与vLLM0.6.3版本一致，此处介绍关键参数。详细参数解释请参见**https://github.com/vllm-project/vllm/blob/main/vllm/engine/arg_utils.py**。

步骤七 启动增量推理实例

1. 启动增量推理容器

   启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci6、davinci7。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

     说明

     - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
     - driver及npu-smi需同时挂载至容器。
     - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
     - 如果需要多个增量实例，每个增量都需要启动一个容器，只挂载对应的NPU
   - --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动增量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_67.json

export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=8089 \
    --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用,（英文逗号）分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP地址
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

步骤八 启动 scheduler 实例

建议在PD服务（即全量推理和增量推理服务）启动后，再启动scheduler服务。

1. 启动scheduler容器。启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了0张卡。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动scheduler实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1
export no_proxy=localhost,127.0.0.1,10.**.**.18

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=9000 \
   --served-model-name ${served-model-name}
```

```
# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。当前端口9000是对外服务端口，而8088、8089则为scheduler调度推理服务端口。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。
- no_proxy：可选，避免scheduler实例和P、D实例之间访问时走不必要的网关。

其中常见的参数如下，

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号。分离部署对外服务使用的是scheduler实例端口，在后续推理性能测试和精度测试时，服务端口需要和scheduler实例端口保持一致。
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量USE_OPENAI=1时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

说明

- 全量和增量节点的local rank table必须一一对应。
- 全量和增量节点不能使用同一个端口。
- scheduler实例中NODE_PORTS=8088,8089；端口设置顺序必须与global rank table文件中各全量和增量节点顺序一致，否则会报错。
- 确保scheduler实例和P、D实例之间网络通畅，检查代理设置例如no_proxy环境变量，避免scheduler访问P、D实例时走不必要的网关。

步骤九 开启动态配比调整功能（可选）

动态配比调整功能允许服务在运行时根据负载调整全量和增量的数量配比。例如启动时设置全量个数为2，增量个数为2。开启此功能后，服务能够根据负载的特性自动调整为1：3或3：1的全量增量比。

全量和增量的启动方法无需变化，scheduler实例启动时需要额外配置一些参数，示例命令如下：

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=9000 \
    --served-model-name ${served-model-name} \
    --dynamic \
    --replan-interval=6000 \
    --workload-results="workload_dir" \
    --max-files=2 \
    --max-workload-nums=100 \
    --profile-input-len="64,256,2048" \
    --profile-out-len="4,16,64,128" \
    --profile-results="training_data_8b-063.json" \
    --ratio-model="ProducerConsumer"

# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

基本参数请参考**步骤八 启动scheduler实例**，额外的参数如下：

- --dynamic：是否开启自适应角色转换，如果不开启，则下方参数无效
- --replan-interval：重新计算pd配比转换的间隔时间，默认6000，单位为秒
- --profile-input-len：profiling的输入长度，默认"64,256,2048"，服务会在启动时测算这些输入长度数据的执行时间，该数量会影响服务启动时间，建议不超过10个
- --profile-out-len：profiling的输出长度，默认"4,16,64,128"，服务会在启动时测算这些输出长度数据的执行时间，该数量会影响服务启动时间，建议不超过5个
- --profile-results：profiling结果的输出地址，用于估算配比的重要数据，可重复使用及用于离线计算配比
- --ratio-model：配比计算方法，当前仅支持"ProducerConsumer"
- --workload-results：定期将处理过的请求输出到该目录下，务必是一个空文件夹
- --max-files：记录请求的最大文件数量
- --max-workload-num: 每个文件记录的最大请求数量，当请求数超过该值时才会触发配比调整

除了在线配比调整的功能之外，还提供了额外离线估算最优配比的工具，用户可以在服务启动之前用此工具获得一个较优的配比，使用示例如下：

```
python ${LLM_TOOLS_PATH}/PD_separate/ratio_offline.py \
    --model=${model} \
    --tensor-parallel-size=1 \
    --max-model-len=8192 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=8192 \
    --ratio-model=ProducerConsumer \
    --block-num=476 \
    --instance-num=6 \
    --input-len=1024 \
    --output-len=48 \
```

```
    --request-rate=10 \
    --num-prompts=1000 \
    --arrival-pattern=poisson \
    --profile-results=training_data_8b-063.json
```

其中常见的参数如下：

- --model：HuggingFace下载的模型文件，此工具不会加载模型权重，也无需NPU
- --max-num-seqs：同时处理的最大句子数量
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --ratio-model：配比计算方法，当前仅支持"ProducerConsumer"
- --block-num: 全量或增量实例可用的free block数，可通过**步骤六 启动全量推理实例**中的启动日志获得
- --instance-num: 全量+增量的实例总个数
- --input-len: 模拟数据的输入长度
- --output-len: 模拟数据的输出长度
- --request-rate: 平均每秒发送的请求个数（ request per seconds ）
- --num-prompts: 模拟的请求总数
- --arrival-pattern: 请求发送的pattern，当前仅支持poisson
- --profile-results: profiling的结果文件，可通过在scheduler实例开启动态功能获得，较大影响配比计算的准确率，建议先获取此文件后使用该工具

步骤十 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-69**。

通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

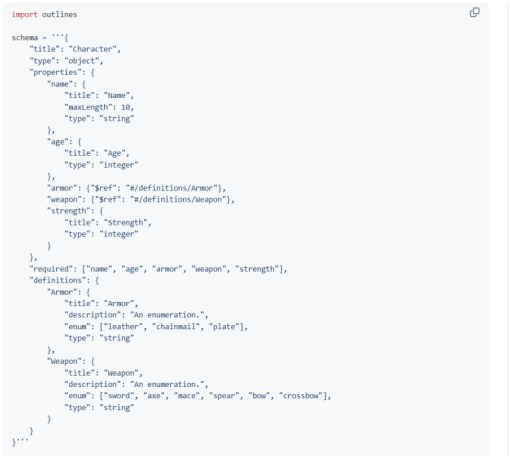
```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
   "stream": false
}'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-69 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** 分离部署暂不支持n>1 n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0 **说明** 分离部署暂不支持use_beam_search=True
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-112** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.8.3 推理性能测试

4.8.3.1 语言模型推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
    |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
    |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。

   ```
   cd benchmark_tools
   ```

语言模型脚本相对路径是tools/llm_evaluation/benchmark_tools/benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_parallel.py --backend openai --host ${docker_ip} --port ${port} --tokenizer /path/to/tokenizer  --epochs 5 \
--parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等后端。本文档使用的推理接口是openai。
- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
- --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。
- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。不输入默认为random。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。
- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-113 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。

- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

```
cd benchmark_tools
conda activate python-3.9.10
```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_serving.py --backend openai --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-114 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

单条请求性能测试

针对openai的/v1/completions以及/v1/chat/completions两个非流式接口，请求体中可以添加可选参数"return_latency"，默认为false，如果指定该参数为true，则会在相应请求的返回体中返回字段"latency"，返回内容如下：

1. prefill_latency（首token时延）：请求从到达服务开始到生成首token的耗时
2. model_prefill_latency（模型计算首token时延）：服务从开始计算首token到生成首token的耗时
3. avg_decode_latency（平均增量token时延）：服务计算增量token的平均耗时
4. time_in_queue（请求排队时间）：请求从到达服务开始到开始被调度的耗时
5. request_latency（请求总时延）：请求从到达服务开始到结束的耗时

以上指标单位均是ms，保留2位小数。

4.8.3.2 多模态模型推理性能测试

benchmark 方法介绍

静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
   |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。
   ```
   cd benchmark_tools
   ```
3. 多模态模型脚本相对路径是llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python modal_benchmark_parallel.py \
   --host ${docker_ip} \
   --port ${port} \
   --tokenizer /path/to/tokenizer  \
   --epochs 5 \
   --parallel-num 1 4 8 16 32 \
   --prompt-tokens 1024 2048 \
   --output-tokens 128 256 \
   ```

```
--height ${height} \
--width ${width} \
--benchmark-csv benchmark_parallel.csv
```

参数说明

- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --height: 图片长度（分辨率相关参数）。
- --width: 图片宽度（分辨率相关参数）。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

备注：当前版本仅支持语言+图片多模态性能测试。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中。

4.8.4 推理精度测试

本章节介绍两个精度测评工具。如何使用opencompass工具开展语言模型的推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen；以及使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等，该工具为离线测评，不需要启动推理服务，目前支持大语言模型。

约束限制

- 确保容器可以访问公网。
- 使用opencompass工具需用vllm接口启动在线服务。
- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

使用 Opencompass 精度测评工具

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh          #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。
```
conda activate python-3.9.10
bash install.sh
```
3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。
```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
pip install huggingface-hub==0.25.1
```
4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**5**进行评测。
```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```
5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。
```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```
参数说明:
   - vllm_path：构造vllm评测配置脚本名字，默认为vllm。
   - host：与起服务的host保持一致，比如起服务为0.0.0.0，host设置也为0.0.0.0。
   - service_port：服务端口，与启动服务时的端口保持，比如8080。
   - max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
   - eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
   - model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
   - benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

   参考命令：
```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen   model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```


6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size，推理时传入的prompts数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用ppl时必须为True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

   ```
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
   ```

   执行脚本如下：

   ```
   python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
   ```

   output_path 指定保存结果的路径。

   参考模型llama3系列模型，数据集mmlu为例，配置如下：

   表 4-70 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

```
# for llama3_8b
python run.py --datasets mmlu_ppl \
--hf-type base --hf-path {hf-path} \
--max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
-w {output_path} --debug
```

参数说明如下：

- --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
- --hf-type：HuggingFace模型权重类型(base,chat)，默认为chat，依据实际的模型选择。
- --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
- --max-seq-len：模型的最大序列长度。
- --max-out-len：模型的最大输出长度。
- --hf-num-gpus：需要使用的卡数。
- --batch-size：推理每次处理的输入数目。
- -w：存放输出结果的目录。

9. 查看精度测试结果。

   默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。

   总体打分结果参考txt和csv文件的最后一行，举例如下：

   npu：

   mmlu：46.6

   gpu：

   mmlu：47

   NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远（小于10）认为分数有效。

使用 Lm-eval 精度测评工具

使用lm-eval工具暂不支持qwen-7b、qwen-14b、qwen-72b、chatglm2-6b、chatglm3-6b模型。

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。

   ```
   rm -rf lm-evaluation-harness/
   git clone https://github.com/EleutherAI/lm-evaluation-harness.git
   cd lm-evaluation-harness
   git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
   pip install -e .
   pip install aiohttp==3.9.3
   ```

2. 执行如下精度测试命令，可以根据参数说明修改参数。

   ```
   lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization},distributed_executor_backend='ray' \
   --tasks ${task}  --batch_size ${batch_size}   --log_samples  --cache_requests true  --trust_remote_code  --output_path ${output_path}
   ```

   参数说明:

- model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
  - vllm_path是模型权重路径；
  - max_model_len 是最大模型长度，默认设置为4096；
  - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
  - tensor_parallel_size是使用的卡数；
  - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
  - distributed_executor_backend是开启多进程服务方式，选择ray开启。
- model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
- tasks：评测数据集任务，比如openllm。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
- output_path：结果保存路径。

使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,distributed_executor_backend='ray' \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant",distributed_executor_backend='ray' \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

使用 MME 精度测评工具

支持多模态模型精度测试。目前支持模型：llava, llava-next, minicpm，qwen-vl, internvl2, qwen2-vl.

1. MME数据集获取

   请用户自行获取**MME评估集**，将MME评估集放于llm_tools/llm_evaluation/mme_eval/data/eval/

2. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation/mme_eval目录中，代码目录结构如下。

   ```
   mme_eval
   ├──metric.py       #MME精度测试脚本
   ├──MME.sh          #运行MME脚本
   ```

3. 启动脚本：

   ```
   export MODEL_PATH=/data/nfs/model/InternVL2-8B/
   export MME_PATH=/llm_tools/llm_evaluation/mme_eval/data/eval/MME
   export MODEL_TYPE=internvl2
   export OUTPUT_NAME=internvl2-8B
   export ASCEND_RT_VISIBLE_DEVICES="0:1:2:3:4:5:6:7"
   bash MME.sh
   ```

参数说明:

a. MODEL_PATH：模型权重路径，默认为空；
b. MME_PATH：MME数据集路径,默认当前路径；
c. MODEL_TYPE：模型类型；
d. OUTPUT_NAME：输出结果文件名称， 默认llava；
e. ASCEND_RT_VISIBLE_DEVICES：表示支持多个模型服务实例,同时支持模型并行，如 0,1:2,3 默认0卡;
f. QUANTIZATION：为量化选项，不传入默认为None即不启用量化；支持w8a8、w8a16，需配套对应的权重使用。
g. GPU_MEMORY_UTILIZATION：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。

脚本运行完成后，测试结果输出在终端。

4.8.5 推理模型量化

4.8.5.1 使用 AWQ 量化

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-65**。多模态只支持hf上下载的awq权重，可跳过步骤一。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

步骤一 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

AutoAWQ量化工具的适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/AutoAWQ目录下。

1、使用该量化工具，需要切换conda环境，运行以下命令。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。

- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

步骤二 权重格式离线转换（可选）

在GPU上AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.8.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-65**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

参考**步骤六 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.8.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化、per-tensor+per-head静态量化以及per-token，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-65**。

per-tensor 静态量化场景

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数。

步骤1 使用tensorRT量化工具进行模型量化。

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

步骤2 抽取kv-cache量化系数。

该步骤的目的是将步骤**步骤1**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

步骤3 启动kv-cache-int8-per-tensor量化服务。

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8_pertensor  #只支持int8，表示kvint8 per-tensor量化
--quantization-param-path kv_cache_scales.json  #输入2. 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

----结束

per-tensor+per-head 静态量化场景

如需使用该场景量化方法，请自行准备kv-cache量化系数，格式和per-tensor静态量化所需的**2. 抽取kv-cache量化系数**生成的json文件一致，只需把每一层的量化系数修改为列表，列表的长度为kv的头数，列表中每一个值代表每一个kv头使用的量化系数。内容示例如下：

```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": [0.09965550899505615, 0.20214074850082397, 0.16
                "1": [0.07757135480642319, 0.15182086825370789, 0.13
```

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-tensor+per-head量化服务。

```
--kv-cache-dtype int8_pertensor_perhead  #只支持int8，表示kvint8 per-tensor+per-head量化
--quantization-param-path kv_cache_scales.json  #输入生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

per-token 动态量化场景

如需使用该场景量化方法，推理前向会自动计算kv-cache量化系数，并进行kv的量化。

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-token量化服务。

```
--kv-cache-dtype int8_pertoken  #只支持int8，表示kvint8 per-token量化
```

4.8.5.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**表4-65**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。

```
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
model_id = "meta-llama/CodeLlama-34b-hf"
tokenizer = AutoTokenizer.from_pretrained(model_id)
gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。

```
dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。

```
quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
quantization_config=gptq_config)
```

5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。

```
quantized_model.save_pretrained("CodeLlama-34b-hf")
tokenizer.save_pretrained("CodeLlama-34b-hf")

# if quantized with device_map set
quantized_model.to("cpu")
quantized_model.save_pretrained("CodeLlama-34b-hf")
```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**步骤六 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.8.5.5 使用 llm-compressor 工具量化

当前版本使用llm-compressor工具量化仅支持Deepseek-v2系列模型的W8A8量化。

本章节介绍如何在GPU的机器上使用开源量化工具**llm-compressor**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：

```
git clone https://github.com/vllm-project/llm-compressor.git
cd llm-compressor
pip install -e .
```

2. 修改examples/quantizing_moe/deepseek_moe_w8a8_int8.py中的代码：

   1）如果本地已有权重，请将MODEL_ID修改为权重路径；

   ```
   MODEL_ID = "deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct"
   ```

   2）如果量化Deepseek-V2-236B模型，请将num_gpus改为8；

   ```
   device_map = calculate_offload_device_map(
       MODEL_ID,
       reserve_for_hessians=True,
       num_gpus=8,
       torch_dtype=torch.bfloat16,
       trust_remote_code=True,
   )
   ```

   3）为减少量化时间，建议将以下参数设置为512；

   ```
   NUM_CALIBRATION_SAMPLES = 512
   ```

3. 执行权重量化：

   ```
   python deepseek_moe_w8a8_int8.py
   ```

说明

1、执行权重量化过程中，请保证使用的GPU卡上没有其他进程，否则可能出现OOM；

2、如果量化Deepseek-v2-236b模型，大致需要10+小时。

使用量化模型

使用量化模型需要在NPU的机器上运行。

启动vLLM前，请开启图模式（参考**步骤六 启动推理服务**中的配置环境变量），启动服务的命令和启动非量化模型一致。

4.8.6 Eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据训练eagle小模型，并使用自行训练的小模型进行eagle推理。支持llama1系列、llama2系列和Qwen2系列模型。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装Eagle。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的如下格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
{
"prefix":  "AAA"
"input": "BBB",
"output": "CCC"
}
```

执行convert_to_sharegpt.py 文件。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
```

```
--prefix_name instruction \
--input_name input  \
--output_name output \
--code_type  utf-8
```

其中：

- input_file_path：预训练json文件地址。
- out_file_name：输出的sharegpt格式文件地址。
- prefix_name：预训练json文件的前缀字段名称，例如：您是一个xxx专家，您需要回答下面问题。prefix_name可设置为None，此时预训练数据集只有input和output两段输入。
- input_name：预训练json文件的指令输入字段名称，例如：请问苹果是什么颜色。
- output_name output：预训练json文件的output字段名称，例如：苹果是红色的。
- code_type：预训练json文件编码，默认utf-8。

当转换为sharegpt格式时，prefix和input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

设置环境变量。

```
export EAGLE_TARIN_MODE=1
```

如果使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

如果使用其他数据集，需要先执行**步骤二：非sharegpt格式数据集转换（可选）**转换数据集格式为sharegpt格式。

执行如下脚本将sharegpt格式数据生成为训练data数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--npu_indices "0,1,2,3,4,5,6,7" \
--used_npus 8  \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048
--dtype bfloat16
```

其中

- outdir：生成的训练data地址。
- end_num：生成的data总条数。
- npu_indices：使用哪些NPU卡。
- used_npus：拉起的每个py脚本使用几个NPU，如果为70b则填写4或8，7b 13b则填1。
- model_type llama：使用模型类型，目前支持qwen2、llama1、llama2，其中llama1、llama2填写llama，qwen2填写为qwen2。

- model_name：模型地址。
- data_path：预训练数据集地址。
- seed 42：生成训练data所使用的seed，此处42为开源训练设定参数。
- max_length：模型的max_length。
- dtype：为模型dtype，默认为bfloat16。

执行完成后，记得unset环境变量，否则会导致后续推理服务启动出错。

```
unset EAGLE_TARIN_MODE
```

执行完成后，如果used_npus>1，则需要将训练生成data数据重新分配为8个文件夹，分配脚本为reassign_data_num.py。

```
python reassign_data_num.py --old_folder "./sharegpt_0_199_mufp16/" \
--new_folder "./sharegpt_0_199_mufp16/" \
--tp 8
```

- old_folder为上一步生成data的地址，填写到卡号的文件夹之前。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- new_folder为需要存储新的data的地址。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- tp为需要切分成的文件夹数量，默认为8。

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

- tmpdir：即为步骤三中的outdir，训练data地址
- cpdir：为训练生成权重的地址
- configpath：为模型config文件的地址
- basepath：为大模型权重地址
- bs：为batch大小

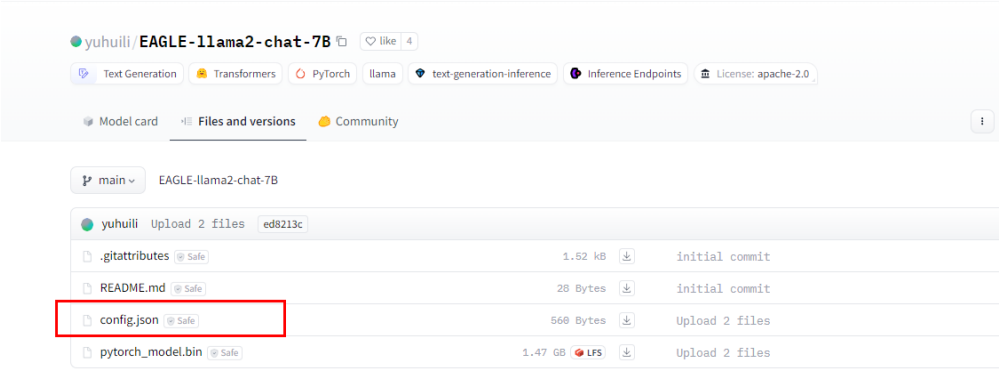
其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-115 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。

图 4-116 eagle config 文件



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

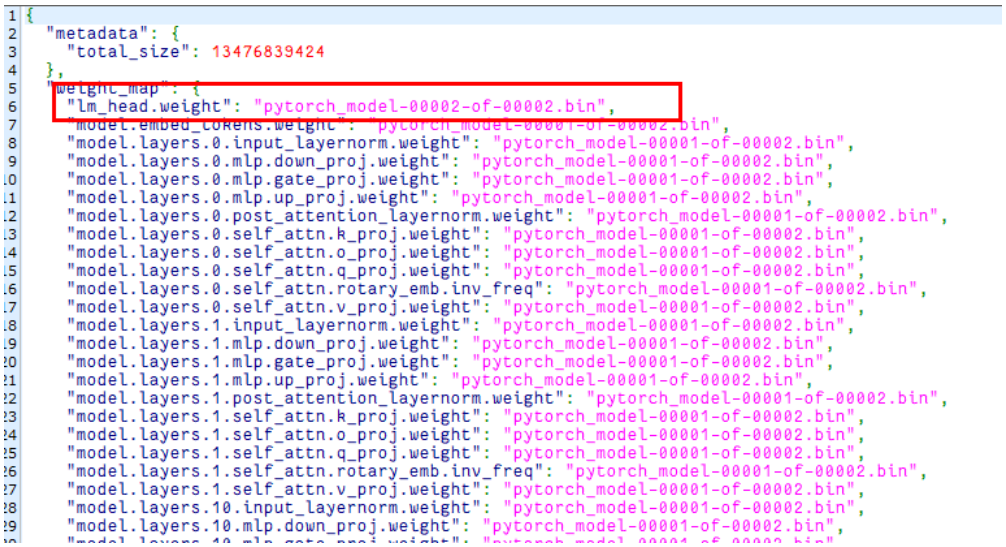
将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

然后在llm_tools/spec_decode/EAGLE文件夹，执行

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

- --base-path：为大模型权重地址，例如 ./llama2-7b-chat
- --draft-path：小模型权重地址，即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat
- --base-weight-name：为大模型包含lm_head的权重文件名，可以在base-path目录下的 model.safetensors.index.json 文件获取，例如llama2-7b-chat的权重名为pytorch_model-00001-of-00002.bin

图 4-117 权重文件名



--draft-weight-name 为小模型权重文件名，即刚才移动的.bin文件或者.safetensors文件。

4.8.7 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.3）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-71 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16

4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128
12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16
22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256

24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32
32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64
42	deepseek-llm-7b	1	16	1	32

43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32
51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	1	16`	1	32
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	internVL2-Llama3-76B	-	-	4	8
59	MiniCPM-v2.6	-	-	1	8

60	llama-3.1-405B-AWQ	-	-	8	32
61	qwen2-57b-a14b	-	-	2	16
62	deepseek-v2-lite-16b	2	4	1	4
63	deepseek-v2-236b	-	-	8	4
64	qwen2-vl-2B	1	8	1	8
65	qwen2-vl-7B	1	8	1	32
66	qwen2-vl-72B	-	-	4	32
67	qwen-vl	1	64	1	64
68	qwen-vl-chat	1	64	1	64
69	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.8.8 附录：大模型推理常见问题

问题 1：在推理预测过程中遇到 NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题 2：在推理预测过程中遇到 ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

问题 3：使用 llama3.1 系列模型进行推理时报错

使用llama3.1系模型进行推理时报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor':

1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

解决方法：

升级transformers版本到4.43.1

```
pip install transformers --upgrade
```

问题 4：使用 SmoothQuant 进行 W8A8 进行模型量化时报错

使用SmoothQuant进行W8A8进行模型量化时报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42

```
pip install transformers==4.42 --upgrade
```

问题 5：使用 AWQ 转换 llama3.1 系列模型权重出现报错

使用AWQ转换llama3.1系列模型权重出现报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor'

解决方法：

该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

问题 6：使用 Qwen2-7B、Qwen2-72B 模型有精度问题，重复输出感叹号

检查**步骤六中4. 配置环境变量**章节中，高精度模式的环境变量是否开启。

问题 7：使用 autoAWQ 进行 qwen-7b 模型量化时报错

使用autoAWQ进行qwen-7b模型量化时报错：TypeError: 'NoneType' object is not subscriptable

解决方法：

修改qwen-7b权重路径下modeling_qwen.py第39行为SUPPORT_FP16 = True

问题 8：使用 benchmark-tools 对 GLM 系列模型进行性能测试报错

使用benchmark-tools对GLM系列模型进行性能测试报错TypeError: _pad() got an unexpected keyword argument 'padding_side'

解决方法：

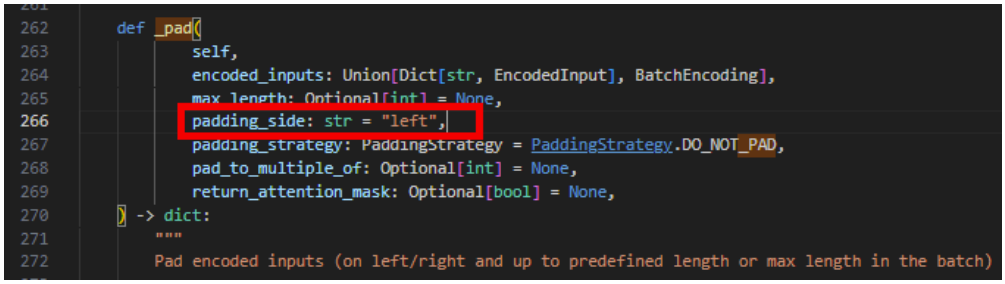
1、下载最新的tokenization_chatglm.py，替换原来权重里的tokenization_chatglm.py。

https://huggingface.co/THUDM/glm-4-9b-chat/blob/main/tokenization_chatglm.py

https://huggingface.co/THUDM/chatglm3-6b/blob/main/tokenization_chatglm.py

或者2、修改tokenization_chatglm.py，在266行增加padding_side: str = "left"，如图4-118所示。

图 4-118 tokenization_chatglm.py



问题 9：使用 benchmark-tools 访问推理服务返回报错

使用benchmark-tools访问推理服务时，输入输出的token和大于max_model_len，服务端返回报错Response payload is not completed，见图4-119。

再次设置输入输出的token和小于max_model_len访问推理服务，服务端响应200，见图4-120。

客户端仍返回报错Response payload is not completed，见图4-121。

图 4-119 服务端返回报错 Response payload is not completed

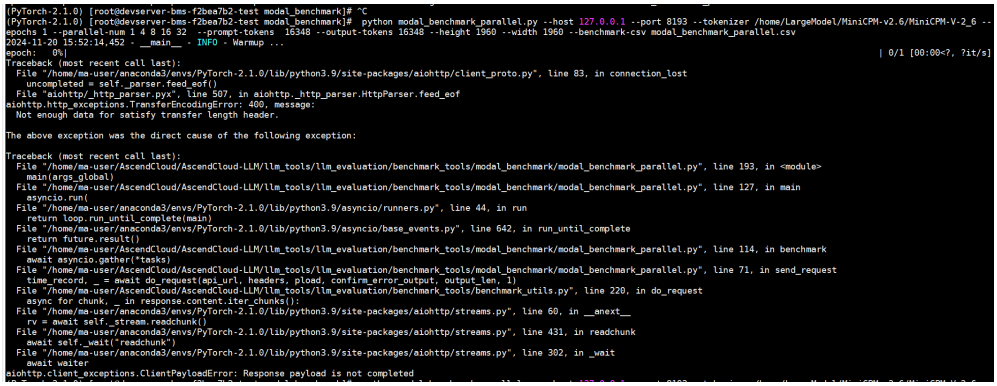


图 4-120 服务端响应 200

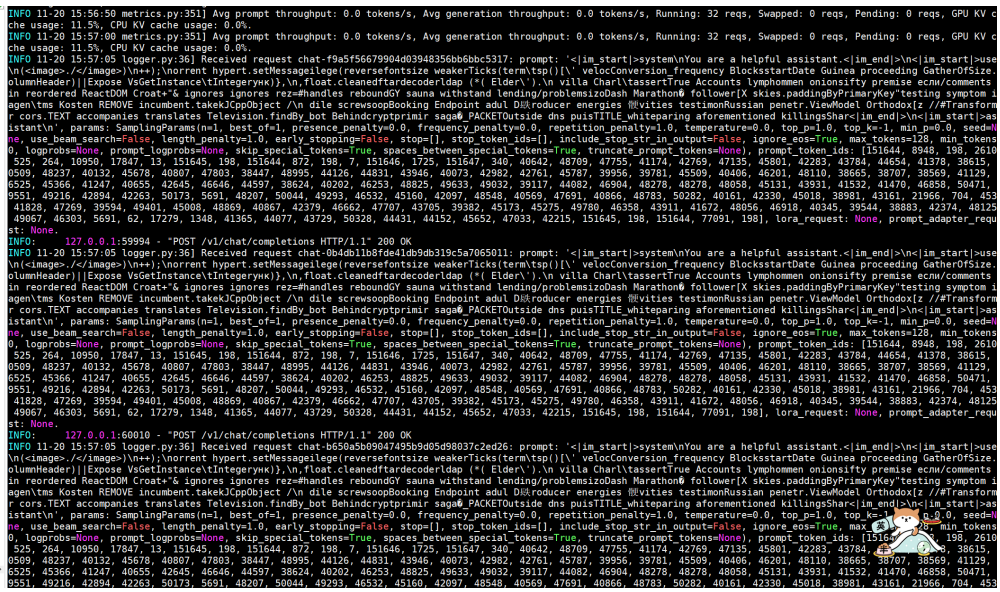
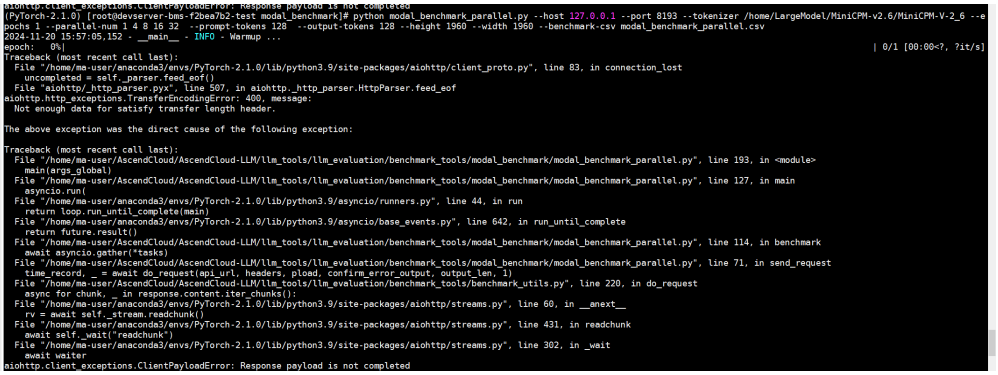


图 4-121 仍返回报错 Response payload is not completed



解决方法：

安装brotlipy后返回正确报错

```
pip install brotlipy
```

问题 10：使用 benchmark-tools 访问推理客户端返回报错或警告

使用benchmark-tools访问推理客户端返回报错或警告：actual output_tokens_length < expected output_len

图 4-122 benchmark-tools 访问推理客户端返回报错

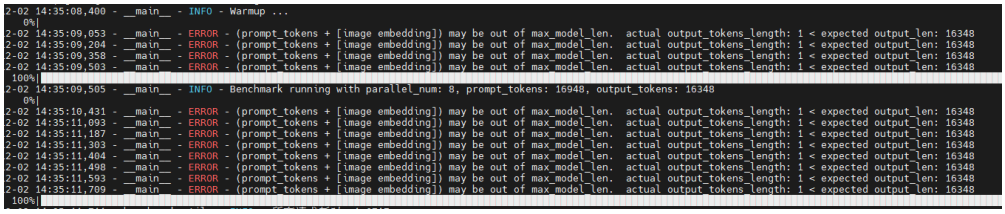
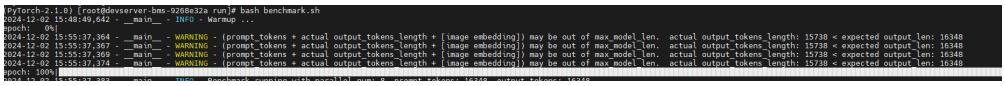


图 4-123 benchmark-tools 访问推理客户端返回警告



解决方法：

减少参数--prompt-tokens和--output-tokens的值，或者增大启动服务的参数--max-model-len的值。

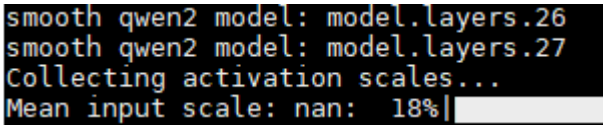
问题 11：使用离线推理时，性能较差或精度异常

解决方法：将block_size大小设置为128

```
from vllm import LLM, SamplingParams
llm = LLM(model="facebook/opt-125m", block_size=128)
```

问题 12：使用 SmoothQuant 做权重转换时，scale 显示为 nan 或推理时精度异常

图 4-124 权重转换 scale 显示为 nan



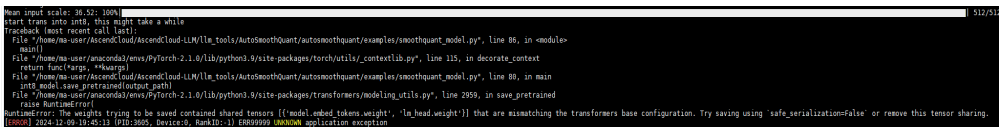
涉及模型：qwen2-1.5b, qwen2-7b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/utils/utils.py中的build_model_and_tokenizer函数，将torch_dtype类型从torch.float16改成torch.bfloat16

```
kwargs = {"torch_dtype": torch.bfloat16, "device_map": "auto"}
```

问题 13：使用 SmoothQuant 做权重转换时报错

图 4-125 权重转换报错



涉及模型：qwen2-1.5b, qwen2-0.5b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples/smoothquant_model.py中的main函数，保存模型时将safe_serialization指定为False

```
int8_model.save_pretrained(output_path,safe_serialization=False)
```

4.9 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.911）

4.9.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.3版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.3版本。不同vLLM版本支持的模型列表有差异，具体如**表4-72**所示。

表 4-72 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**
3	llama-65b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-65b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-34b-hf/tree/main
57	internvl2-8B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-8B/tree/main
58	internvl2-26B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-26B/tree/main
59	internvl2-40B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-40B/tree/main
60	internVL2-Llama3-76B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B/tree/main
61	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
62	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
63	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite
64	qwen2-vl-2B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
65	qwen2-vl-7B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
66	qwen2-vl-72B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct/tree/main
67	qwen-vl	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL
68	qwen-vl-chat	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL-Chat

69	MiniCPM-v2	√	x	x	x	x	https://huggingface.co/HwwwH/MiniCPM-V-2
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

操作流程

图 4-126 操作流程图

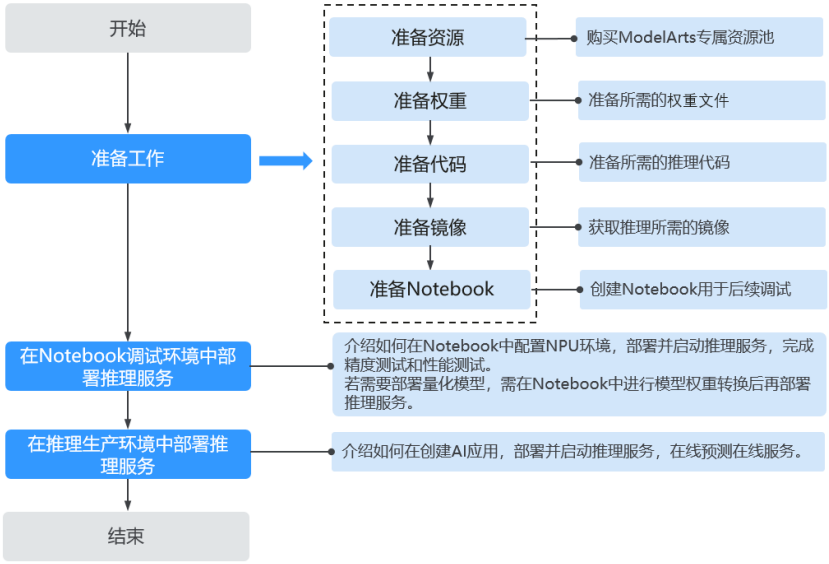


表 4-73 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.911-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 如果需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.9.2 准备工作

4.9.2.1 准备资源

创建专属资源池

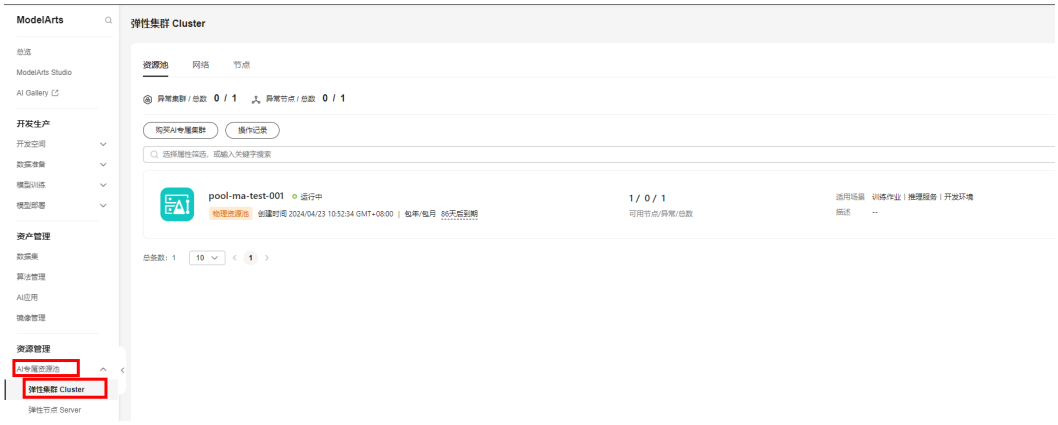
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 4-127 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 4-128 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.9.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.9.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-74**所示。

表 4-74 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.3-py3-none-any.whl   # 推理安装包
       ├── build.sh                # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
```

```
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
  ├──AutoAWQ            # W4A16量化工具
      ├──convert_awq_to_npu.py   # awq权重转换脚本
      ├──quantize.py   # 昇腾适配的量化转换脚本
      ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

4.9.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-75 基础容器镜像地址

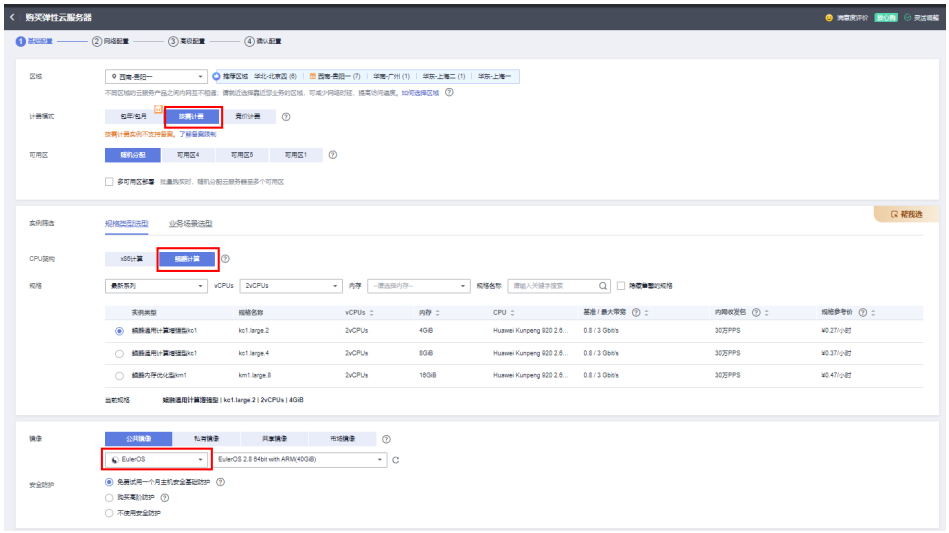
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-129 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-130 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-74**。

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip。

```
unzip AscendCloud-*.zip -d ./AscendCloud && cd ./AscendCloud && unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-2.1.0*.zip -d ./AscendCloud-OPP && cd .. && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```

执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

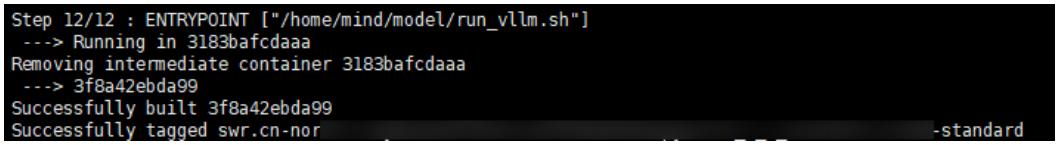
```
sh build_image.sh --base-image=${base_image} --image-name=${image_name} --specify-enrtypoint=True
```

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

图 4-131 成功构建镜像



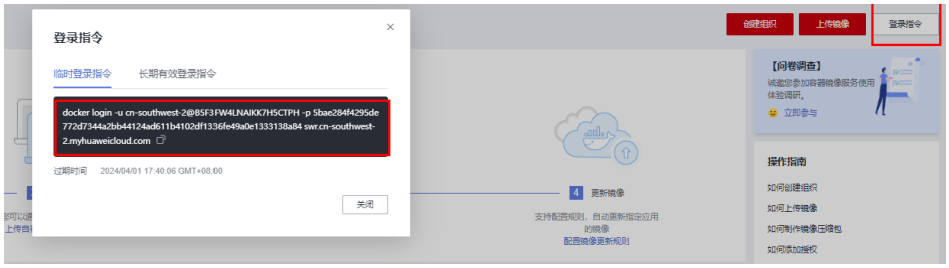
如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-132 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

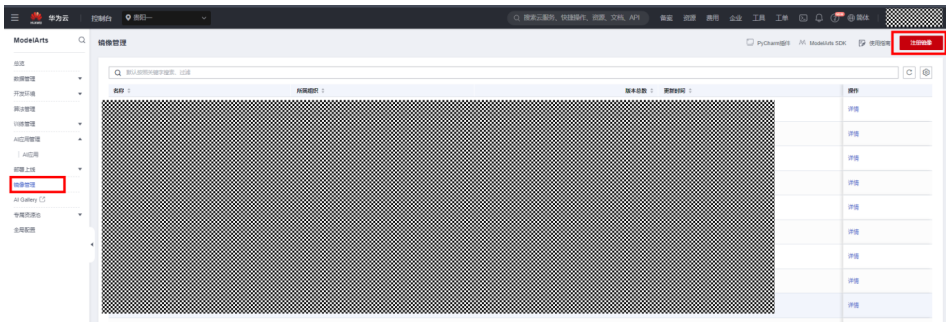
打印如下信息，表示上传镜像成功。

图 4-133 成功上传镜像

Step8 注册镜像

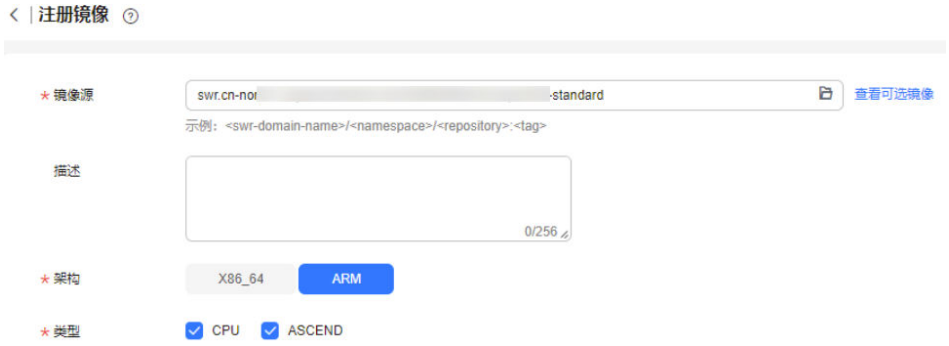
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中单击“注册镜像”。

图 4-134 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 4-135 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.9.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 4-136 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-137 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.9.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

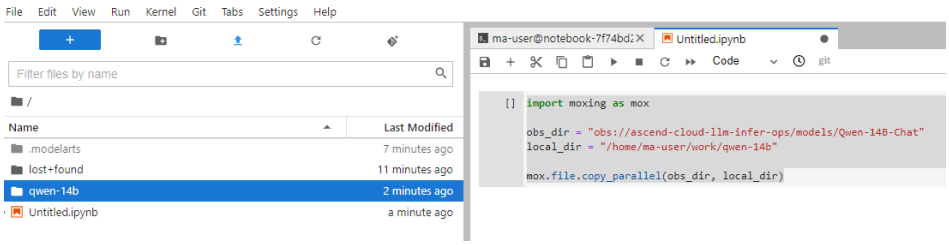
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-138 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-139 查询结果

```
+------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                       |
+---------------------------+---------------+----------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+==============================================+
| 4     910B2               | OK            | 91.4        50                0    / 0         |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+==============================================+
| 5     910B2               | OK            | 92.5        51                0    / 0         |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+==============================================+
+---------------------------+---------------+----------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+==============================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+==============================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+==============================================+
```

2. 配置环境变量。

```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

3. 如果要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1   # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32    # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500    # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV    #可选
```

设置动态分档位后，在PTA模式下不支持接收超过最大档的并发请求，超过后会导致推理服务终止。请将最大档（PTA_TORCHAIR_DECODE_GEAR_LIST参数的中设置的最大值）与模型启动时的max-num-seqs保持一致来进行规避。

在MoE模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存

文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

4. 如果要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，如果不开启，目前vllm0.6.3版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
xport ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
```

如果需要使用eagle投机推理功能，需要进入lm_tools/spec_decode/EAGLE 文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体可参考Eagle投机小模型训练章节中的**步骤五：训练生成权重转换成可以支持vLLM推理的格式**。

5. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。

6. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**

     在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     （1）非多模态

     ```
     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --num-scheduler-steps=8 \
     --trust-remote-code \
     --enforce-eager
     ```

     （2）多模态

     当前支持Llava、InternVL、MiniCPM-v2.6模型，具体模型和权重地址参见**支持的模型列表和权重文件**，推荐显卡数量参见**表4-79**。

     ```
     export VLLM_IMAGE_FETCH_TIMEOUT=100
     export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
     # PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
     # 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
     export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --chat-template ${chat_template_path} \
     ```

```
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：

  （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

– **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code \
--enforce-eager
```

推理服务基础参数说明如下：

– -model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。

– --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。

– --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。

– --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。

– --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

– --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。

- --block-size：PagedAttention的block大小，推荐设置为128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。
- --enforce-eager：未设置INFER_MODE环境变量时，部分模型会默认使用CANNGraph图模式启动来提升性能，设置该参数后将关闭图模式。CANNGraph图模式目前支持llama和qwen2系列大语言模型单卡场景，包含该系列AWQ量化模型，其他场景(如Multi-lora)暂未支持。小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
- --disable-async-output-proc：关闭异步后处理特性，关闭后性能会下降。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

  ```
  export USE_PREFIX_HIGH_PRECISION_MODE=1
  ```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

  ```
  --enable-lora \
  --lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
  --max-lora-rank=16 \
  --max-loras=32 \
  --max-cpu-loras=32
  ```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。

- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-76**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
```

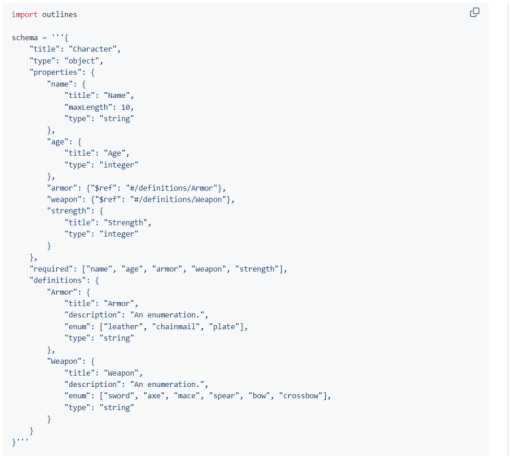
```
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-76 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-140** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-77 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-78 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.9.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

说明

- 如果需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
- 权重文件夹不要以"model"命名，如果以"model"命名会导致后续创建AI应用报错。

- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-141 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code \
--enforce-eager
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
```

```
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True：允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code \
--enforce-eager
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。
- --tensor-parallel-size：并行卡数。此处举例为1，表示使用单卡启动服务。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- -max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。

- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps：默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。
- --enforce-eager：未设置INFER_MODE环境变量时，部分模型会默认使用CANNGraph图模式启动来提升性能，设置该参数后将关闭图模式。CANNGraph图模式目前支持llama和qwen2系列大语言模型单卡场景，包含该系列AWQ量化模型，其他场景(如Multi-lora)暂未支持。小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
- --disable-async-output-proc：关闭异步后处理特性，关闭后性能会下降。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

```
export USE_PREFIX_HIGH_PRECISION_MODE=1
```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
--max-loras=32 \
--max-cpu-loras=32
```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

--max-cpu-loras要求配置和--max-loras相同。

发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 准备模型文件和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

可在run_vllm.sh增加如下环境变量开启高阶配置：

a. 配置环境变量。

```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

b. 如果要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

在MoE模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

c. 如果要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，如果不开启，目前vllm0.6.3版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
```

如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE 文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体可参考Eagle投机小模型训练章节**步骤五：训练生成权重转换成可以支持vLLM推理的格式**。

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 4-142 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-143 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

   首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

图 4-144 创建完成

说明

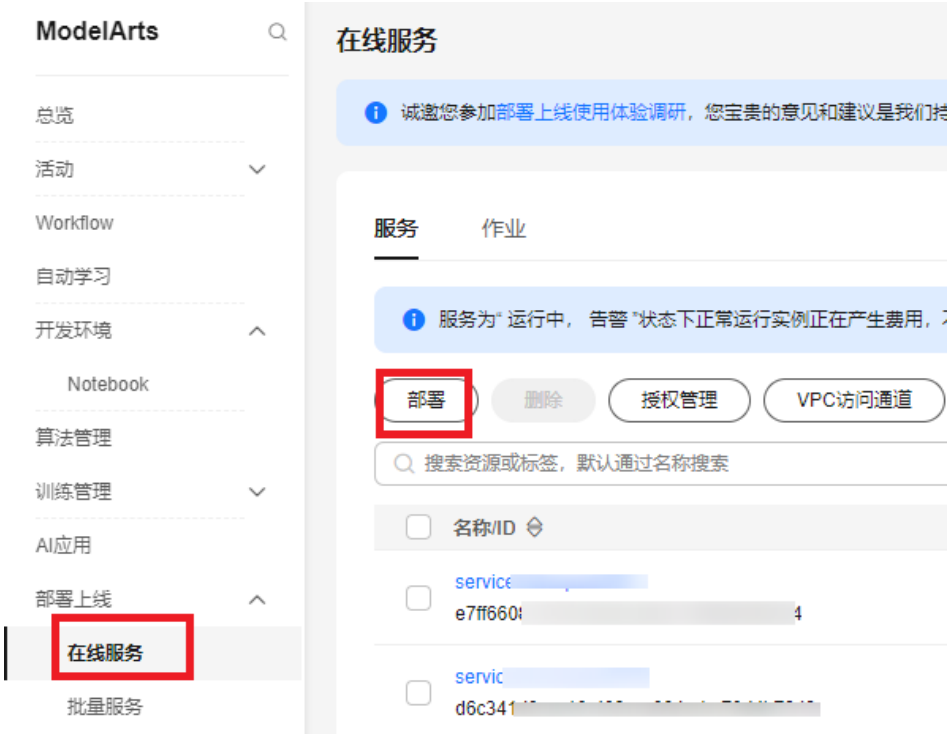
如果权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 4-145 部署在线服务

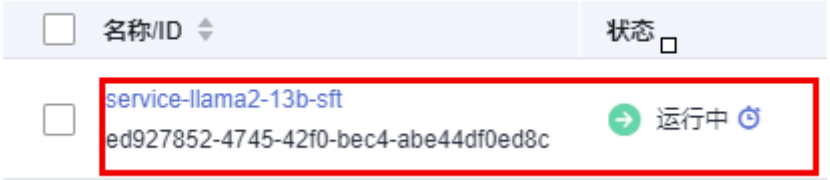


2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-146 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 4-147 服务部署完成

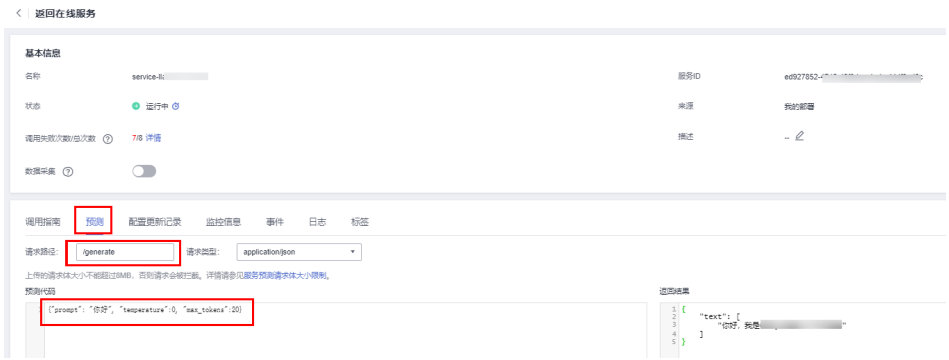


Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

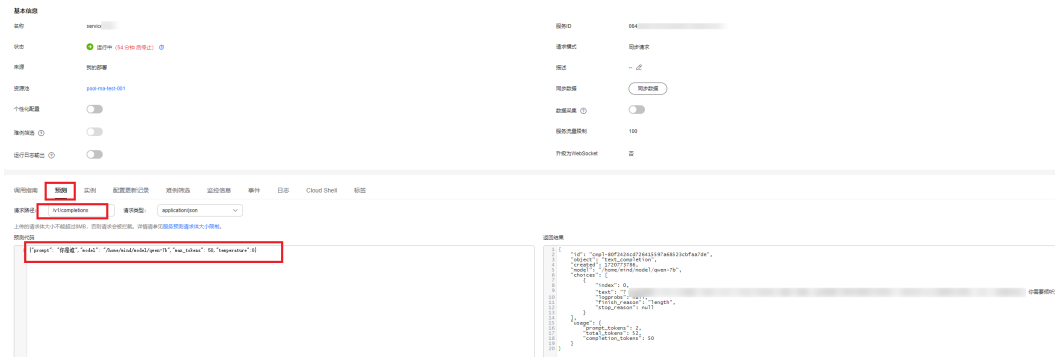
如果以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-148 预测-vllm



如果以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-149 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.9.5 推理精度测试

本章节介绍如何使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等。

约束限制

- 确保容器可以访问公网。

- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

步骤一：配置精度测试环境

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。
```
rm -rf lm-evaluation-harness/
git clone https://github.com/EleutherAI/lm-evaluation-harness.git
cd lm-evaluation-harness
git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
pip install -e .
```
2. 执行如下精度测试命令，可以根据参数说明修改参数。
```
lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization} \
--tasks ${task}  --batch_size ${batch_size}   --log_samples  --cache_requests true  --trust_remote_code --output_path ${output_path}
```
参数说明:
   - model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
     - vllm_path是模型权重路径；
     - max_model_len 是最大模型长度，默认设置为4096；
     - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
     - tensor_parallel_size是使用的卡数；
     - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
   - model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
   - tasks：评测数据集任务，比如openllm。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
   - output_path：结果保存路径。

   使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：
```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096 \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```
   使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：
```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant" \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

4.9.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。如果需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，如果静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py    # 工具函数集
├── benchmark.py          # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

目前性能测试已经支持投机推理能力。

执行性能测试脚本前，需先安装相关依赖。

```
conda activate python-3.9.10
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend openai --host 127.0.0.1 --port 8080 \
--tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

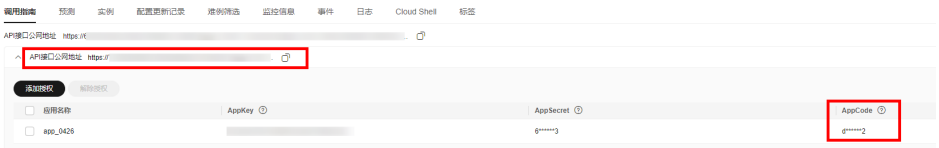
生产环境中进行测试：

```
python benchmark_parallel.py --backend openai --url xxx --app-code xxx \
--tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-150 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
- --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。
- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表

示使用human-eval数据集进行测试。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。

- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-151 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_dataset.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

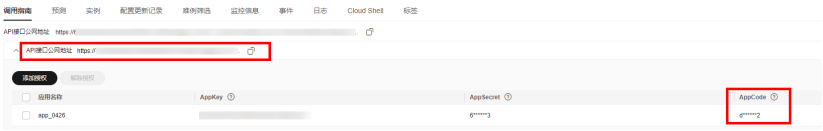
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend openai --host 127.0.0.1 --port 8080 --dataset
   custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8
   10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens
   3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend openai --url xxx --app-code xxx --dataset
   custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8
   10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens
   3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-152 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。

- --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-153 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.9.7 推理模型量化

4.9.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel, W8A16 per-channel

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、使用该量化工具，需要切换conda环境。

```
conda activate awq
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.9.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.9.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

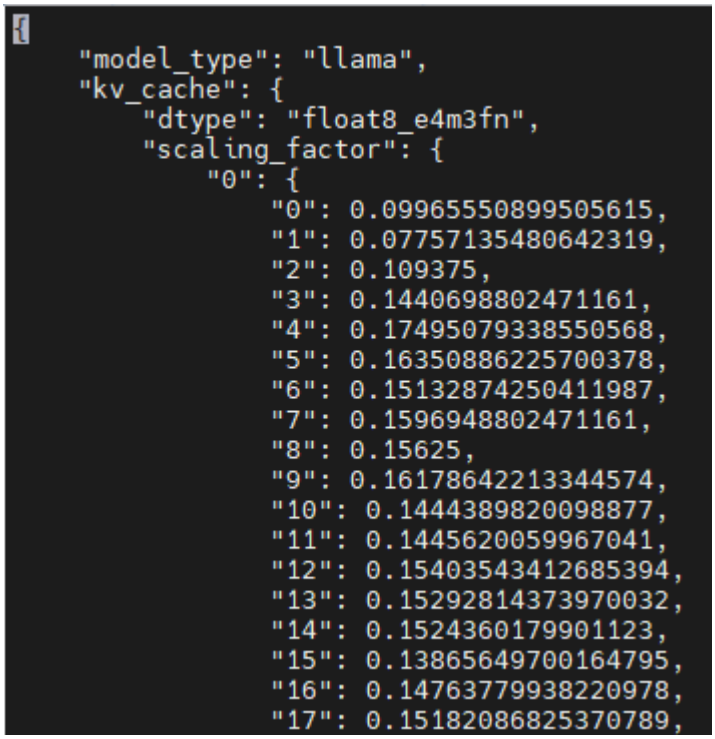
该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-154 抽取 kv-cache 量化系数



注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.9.7.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
```

```
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.9.8 Eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据训练eagle小模型，并使用自行训练的小模型进行eagle推理。支持llama1系列、llama2系列和Qwen2系列模型。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装Eagle。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的如下格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
{
"prefix":  "AAA"
"input": "BBB",
"output": "CCC"
}
```

执行convert_to_sharegpt.py 文件。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

其中：

- input_file_path：预训练json文件地址。
- out_file_name：输出的sharegpt格式文件地址。
- prefix_name：预训练json文件的前缀字段名称，例如：您是一个xxx专家，您需要回答下面问题。prefix_name可设置为None，此时预训练数据集只有input和output两段输入。
- input_name：预训练json文件的指令输入字段名称，例如：请问苹果是什么颜色。

- output_name output：预训练json文件的output字段名称，例如：苹果是红色的。
- code_type：预训练json文件编码，默认utf-8。

当转换为sharegpt格式时，prefix和input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

设置环境变量。

```
export EAGLE_TARIN_MODE=1
```

如果使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

如果使用其他数据集，需要先执行**步骤二：非sharegpt格式数据集转换（可选）**转换数据集格式为sharegpt格式。

执行如下脚本将sharegpt格式数据生成为训练data数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--npu_indices "0,1,2,3,4,5,6,7" \
--used_npus 8  \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048
--dtype bfloat16
```

其中

- outdir：生成的训练data地址。
- end_num：生成的data总条数。
- npu_indices：使用哪些NPU卡。
- used_npus：拉起的每个py脚本使用几个NPU，如果为70b则填写4或8，7b 13b则填1。
- model_type llama：使用模型类型，目前支持qwen2、llama1、llama2，其中llama1、llama2填写llama，qwen2填写为qwen2。
- model_name：模型地址。
- data_path：预训练数据集地址。
- seed 42：生成训练data所使用的seed，此处42为开源训练设定参数。
- max_length：模型的max_length。
- dtype：为模型dtype，默认为bfloat16。

执行完成后，记得unset环境变量，否则会导致后续推理服务启动出错。

```
unset EAGLE_TARIN_MODE
```

执行完成后，如果used_npus>1，则需要将训练生成data数据重新分配为8个文件夹，分配脚本为reassign_data_num.py。

```
python reassign_data_num.py --old_folder "./sharegpt_0_199_mufp16/" \
--new_folder "./sharegpt_0_199_mufp16/" \
--tp 8
```

- old_folder为上一步生成data的地址，填写到卡号的文件夹之前。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- new_folder为需要存储新的data的地址。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- tp为需要切分成的文件夹数量，默认为8。

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

- tmpdir：即为步骤三中的outdir，训练data地址
- cpdir：为训练生成权重的地址
- configpath：为模型config文件的地址
- basepath：为大模型权重地址
- bs：为batch大小

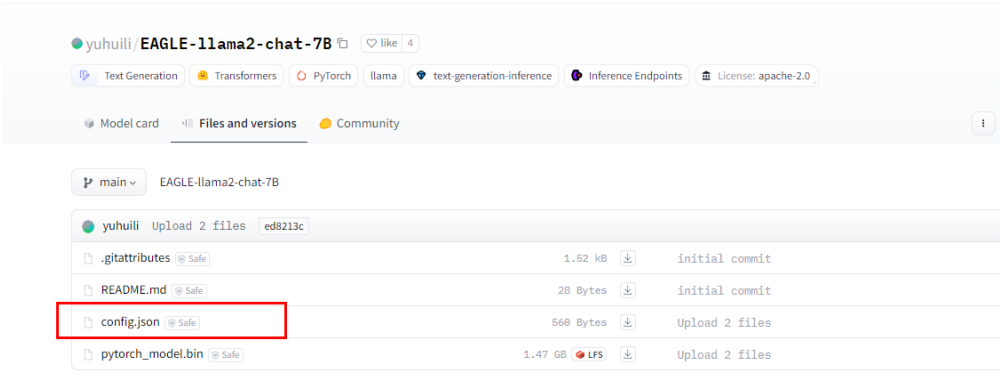
其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-155 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。

图 4-156 eagle config 文件



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

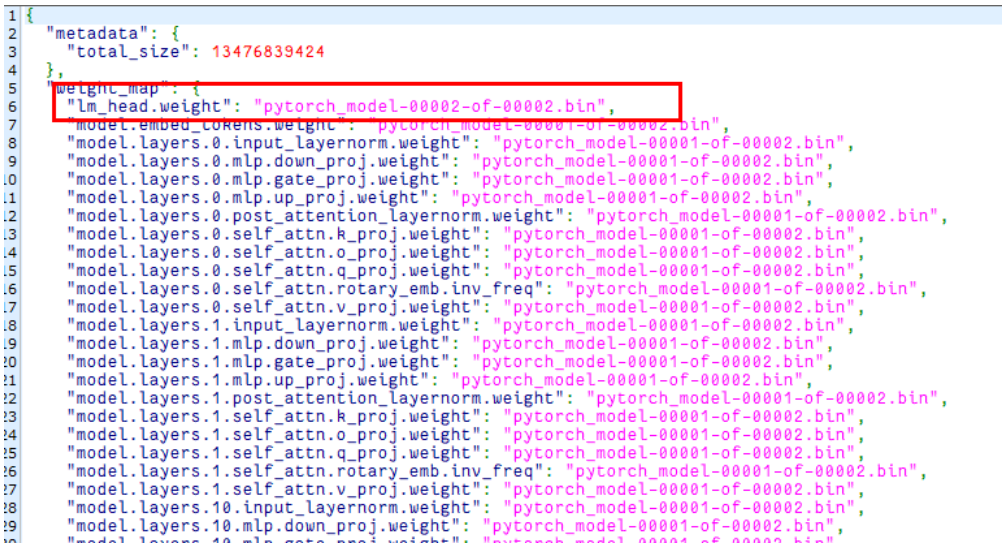
将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

然后在llm_tools/spec_decode/EAGLE文件夹，执行

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

- --base-path：为大模型权重地址，例如 ./llama2-7b-chat
- --draft-path：小模型权重地址，即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat
- --base-weight-name：为大模型包含lm_head的权重文件名，可以在base-path目录下的 model.safetensors.index.json 文件获取，例如llama2-7b-chat的权重名为pytorch_model-00001-of-00002.bin

图 4-157 权重文件名



--draft-weight-name 为小模型权重文件名，即刚才移动的.bin文件或者.safetensors文件。

4.9.9 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.3）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-79 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128

12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16
22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32

32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64
42	deepseek-llm-7b	1	16	1	32
43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32

51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	1	16`	1	32
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	internVL2-Llama3-76B	-	-	4	8
59	MiniCPM-v2.6	-	-	1	8
60	llama-3.1-405B-AWQ	-	-	8	32
61	qwen2-57b-a14b	-	-	2	16
62	deepseek-v2-lite-16b	2	4	1	4
63	deepseek-v2-236b	-	-	8	4
64	qwen2-vl-2B	1	8	1	8
65	qwen2-vl-7B	1	8	1	32
66	qwen2-vl-72B	-	-	4	32
67	qwen-vl	1	64	1	64
68	qwen-vl-chat	1	64	1	64

69	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.9.10 附录：大模型推理常见问题

问题 1：在推理预测过程中遇到 NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题 2：在推理预测过程中遇到 ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

问题 3：使用 llama3.1 系列模型进行推理时报错

使用llama3.1系模型进行推理时报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

解决方法：

升级transformers版本到4.43.1

```
pip install transformers --upgrade
```

问题 4：使用 SmoothQuant 进行 W8A8 进行模型量化时报错

使用SmoothQuant进行W8A8进行模型量化时报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42

```
pip install transformers==4.42 --upgrade
```

问题 5：使用 AWQ 转换 llama3.1 系列模型权重出现报错

使用AWQ转换llama3.1系列模型权重出现报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor'

解决方法：

该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

问题 6：使用 Qwen2-7B、Qwen2-72B 模型有精度问题，重复输出感叹号

检查**步骤六中4. 配置环境变量**章节中，高精度模式的环境变量是否开启。

问题 7：使用 autoAWQ 进行 qwen-7b 模型量化时报错

使用autoAWQ进行qwen-7b模型量化时报错：TypeError: 'NoneType' object is not subscriptable

解决方法：

修改qwen-7b权重路径下modeling_qwen.py第39行为SUPPORT_FP16 = True

问题 8：使用 benchmark-tools 对 GLM 系列模型进行性能测试报错

使用benchmark-tools对GLM系列模型进行性能测试报错TypeError: _pad() got an unexpected keyword argument 'padding_side'

解决方法：

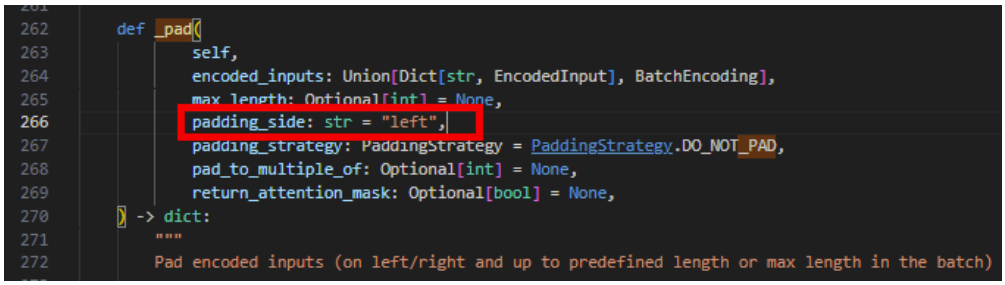
1、下载最新的tokenization_chatglm.py，替换原来权重里的tokenization_chatglm.py。

https://huggingface.co/THUDM/glm-4-9b-chat/blob/main/tokenization_chatglm.py

https://huggingface.co/THUDM/chatglm3-6b/blob/main/tokenization_chatglm.py

或者2、修改tokenization_chatglm.py，在266行增加padding_side: str = "left"，如**图4-158**所示。

图 4-158 tokenization_chatglm.py



问题 9：使用 benchmark-tools 访问推理服务返回报错

使用benchmark-tools访问推理服务时，输入输出的token和大于max_model_len，服务端返回报错Response payload is not completed，见**图4-159**。

再次设置输入输出的token和小于max_model_len访问推理服务，服务端响应200，见**图4-160**。

客户端仍返回报错Response payload is not completed，见**图4-161**。

图 4-159 服务端返回报错 Response payload is not completed

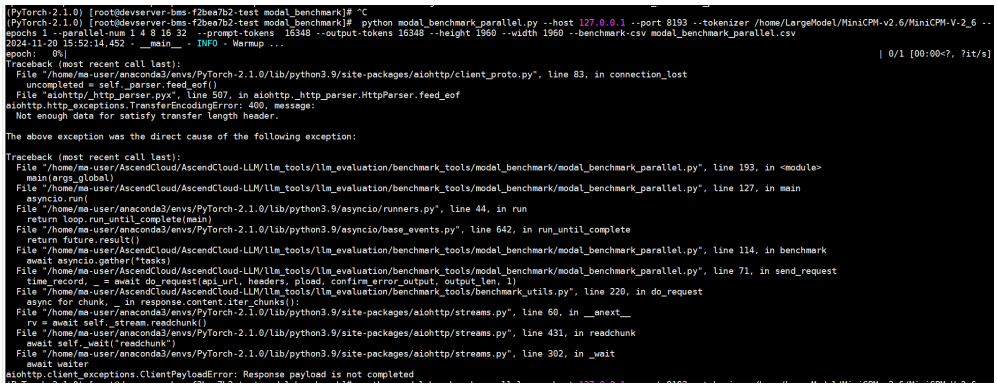


图 4-160 服务端响应 200

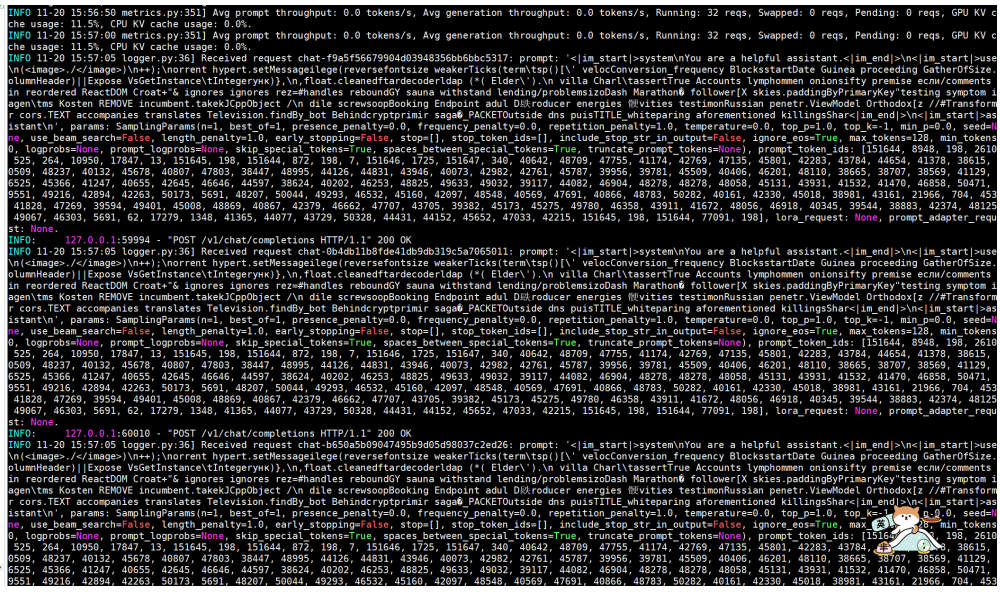
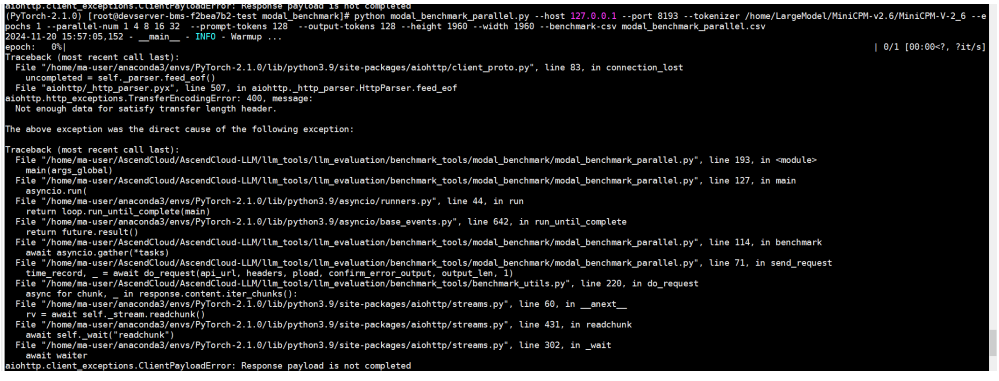


图 4-161 仍返回报错 Response payload is not completed



解决方法：

安装brotlipy后返回正确报错

```
pip install brotlipy
```

问题 10：使用 benchmark-tools 访问推理客户端返回报错或警告

使用benchmark-tools访问推理客户端返回报错或警告：actual output_tokens_length < expected output_len

图 4-162 benchmark-tools 访问推理客户端返回报错

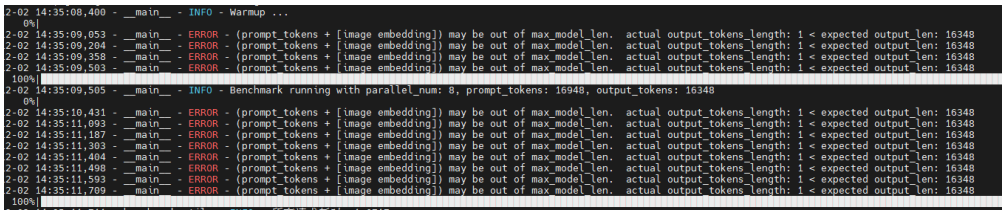
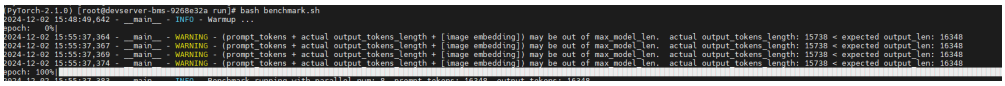


图 4-163 benchmark-tools 访问推理客户端返回警告



解决方法：

减少参数--prompt-tokens和--output-tokens的值，或者增大启动服务的参数--max-model-len的值。

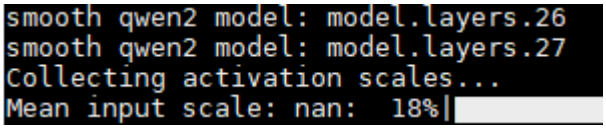
问题 11：使用离线推理时，性能较差或精度异常

解决方法：将block_size大小设置为128

```
from vllm import LLM, SamplingParams
llm = LLM(model="facebook/opt-125m", block_size=128)
```

问题 12：使用 SmoothQuant 做权重转换时，scale 显示为 nan 或推理时精度异常

图 4-164 权重转换 scale 显示为 nan



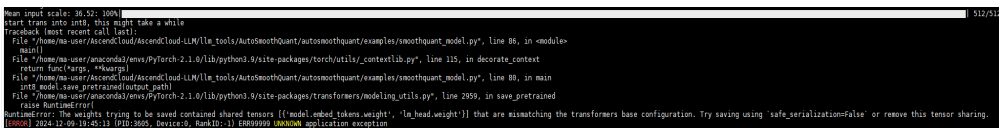
涉及模型：qwen2-1.5b, qwen2-7b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/utils/utils.py中的build_model_and_tokenizer函数，将torch_dtype类型从torch.float16改成torch.bfloat16

```
kwargs = {"torch_dtype": torch.bfloat16, "device_map": "auto"}
```

问题 13：使用 SmoothQuant 做权重转换时报错

图 4-165 权重转换报错



涉及模型：qwen2-1.5b, qwen2-0.5b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples/smoothquant_model.py中的main函数，保存模型时将safe_serialization指定为False

```
int8_model.save_pretrained(output_path,safe_serialization=False)
```

4.10 主流开源大模型基于 Lite Cluster 适配 PyTorch NPU 推理指导（6.3.911）

4.10.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite k8s Cluster上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Lite k8s Cluster和昇腾Snt9B资源。
- 本文档中的CCE集群版本选择v1.27~1.28。版本使用的容器引擎为**Containerd。**
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.3版本。
- 支持FP16和BF16数据类型推理。
- Lite k8s Cluster驱动版本推荐为23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.3版本。不同vLLM版本支持的模型列表有差异，具体如**表4-80**所示。

表 4-80 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-34b-hf/tree/main
57	internvl2-8B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-8B/tree/main
58	internvl2-26B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-26B/tree/main
59	internvl2-40B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-40B/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
60	internVL2-Llama3-76B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B/tree/main
61	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
62	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
63	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite
64	qwen2-vl-2B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
65	qwen2-vl-7B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
66	qwen2-vl-72B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct/tree/main
67	qwen-vl	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL
68	qwen-vl-chat	√	x	x	x	x	https://huggingface.co/Qwen/Qwen-VL-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址

69	MiniCPM-v2	√	x	x	x	x	**https://huggingface.co/HwwwH/MiniCPM-V-2**
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

4.10.2 准备工作

4.10.2.1 准备环境

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**Lite Cluster资源开通**，熟悉集群资源开通流程，再开始操作购买k8s Cluster资源。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

图 4-166 购买 Lite 专属池

k8s Cluster 资源配置

如果已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

kubectl 访问集群配置

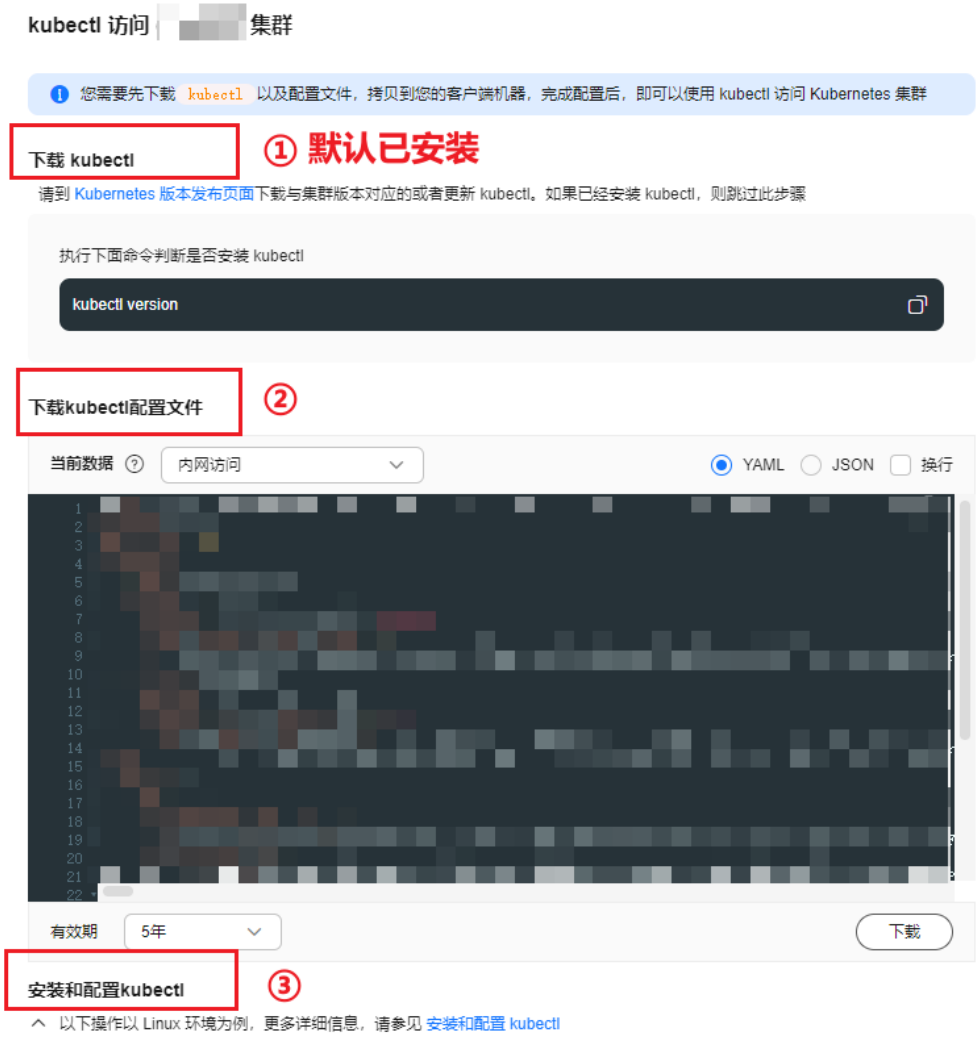
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的CCE集群控制版面中。根据**图4-167**的步骤进行操作，单击kubectl配置时，会弹出**图4-168**步骤页面。

图 4-167 配置中心

2. 根据图4-168，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-168 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。如果显示如图**图4-169**的内容，则配置成功。

```
kubectl cluster-info
```

图 4-169 查看 Kubernetes 集群信息正确弹出内容

4.10.2.2 准备代码

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-81**所示。

表 4-81 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.6.3-py3-none-any.whl   # 推理安装包
        ├── build.sh               # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
  ├──llm_tools          # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
    ├──AutoAWQ            # W4A16量化工具
        ├──convert_awq_to_npu.py   # awq权重转换脚本
        ├──quantize.py   # 昇腾适配的量化转换脚本
        ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py            #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考《 主流开源大模型基于Lite Cluster适配PyTorch训练指导 》。

4.10.2.3 准备镜像

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-82 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v   # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择 containerd 作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

   buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

   buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

   a. 下载并解压buildkit程序。
      ```
      # 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
      wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

      # 创建解压的目录
      mkdir /usr/local/buildkit

      # 解压到指定的目录
      tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit
      ```

```
# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

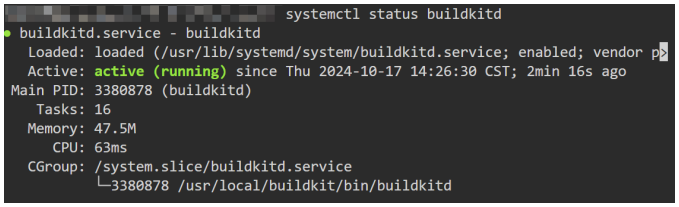
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 如果buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-82**。

containerd 容器引擎有命名空间的概念。**Kubernetes 下使用的 containerd 默认命名空间是 k8s.io**。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。

  ```
  ctr -n k8s.io images pull {image_url}
  ```

- 使用 nerdctl 工具进行镜像拉取。

  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

注意：集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

Step3 制作推理镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-81**。

1. 解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.911-xxx.zip和算子包AscendCloud-OPP-6.3.911-xxx.zip，并直接进入到llm_inference/ascend_vllm文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && cd ./AscendCloud && unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-2.1.0*.zip -d ./AscendCloud-OPP && cd .. && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
   ```
2. 执行以下命令制作推理镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。
   ```
   nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称> --build-arg BASE_IMAGE=${base_image} .
   ```
   注意：nerdctl build会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以使用nerdctl pull命令拉取测试镜像，查看是否能拉取成功。
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
   - ${base_image}为基础镜像地址。

如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

4.10.3 部署推理服务

本章节介绍如何使用vLLM 0.6.3框架部署并启动推理服务。

前提条件

- 已准备好Lite k8s Cluster环境，具体参考**准备环境**。推荐使用“西南-贵阳一”Region上的Cluster和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保集群可以访问公网。

Step1 上传权重文件

将权重文件上传到集群节点机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**支持的模型列表和权重文件**。

如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step2 配置 pod

在节点自定义目录${node_path}下创建config.yaml文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
     app: infers
spec:
  replicas: 1
  selector:
    matchLabels:
      app: infers
  template:
    metadata:
      labels:
        app: infers
    spec:
      schedulerName: volcano
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      containers:
      - image: ${image_name}                  # 推理镜像名称
        imagePullPolicy: IfNotPresent
        name: ${container_name}
        securityContext:
          runAsUser: 0
        ports:
        - containerPort: 8080
        command: ["/bin/bash", "-c"]
        args: ["${node-path}/run_vllm.sh"]          # 节点自定义目录，该目录下包含pod配置文件config.yaml和推
理服务启动脚本run_vllm.sh
        resources:
          requests:
            huawei.com/ascend-1980: "8"             # 需求卡数，key保持不变。
          limits:
            huawei.com/ascend-1980: "8"             # 限制卡数，key保持不变。
        volumeMounts:                           # 容器内部映射路径
        - name: ascend-driver               #驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons          #驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: hccn                          #驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: localtime
          mountPath: /etc/localtime
        - name: npu-smi                           # npu-smi
          mountPath: /usr/local/sbin/npu-smi
        - name: model-path                      # 模型权重路径
          mountPath: ${model-path}
        - name: node-path
          mountPath: ${node-path}
      volumes:                                  # 物理机外部路径
      - name: ascend-driver
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
```

```
  - name: model-path
    hostPath:
      path: ${model-path}
  - name: node-path
    hostPath:
      path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。
- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml和推理服务启动脚本run_vllm.sh，run_vllm.sh内容见**Step3 创建服务启动脚本**。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

Step3 创建服务启动脚本

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code \
--enforce-eager
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  多模态推理服务启动模板参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。

- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code \
--enforce-eager
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：**Step1 上传权重文件**中上传的模型权重路径。
- --tensor-parallel-size：并行卡数。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启投机推理后无需配置该参数，否则会导致投机推理启动报错。
- --multi-step-stream-outputs: 设置false后，mult-step会关闭流式输出提升性能，一次将返回num-scheduler-steps个token。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。
- --enforce-eager：未设置INFER_MODE环境变量时，部分模型会默认使用CANNGraph图模式启动来提升性能，设置该参数后将关闭图模式。CANNGraph图模式目前支持llama和qwen2系列大语言模型单卡场景，包含该系列AWQ量化模型，其他场景(如Multi-lora)暂未支持。小模型如Qwen2-1.5B和Qwen2-0.5B推荐不设置该参数。
- --disable-async-output-proc：关闭异步后处理特性，关闭后性能会下降。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

  ```
  export USE_PREFIX_HIGH_PRECISION_MODE=1
  ```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

  ```
  --enable-lora \
  --lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
  --max-lora-rank=16 \
  --max-loras=32 \
  --max-cpu-loras=32
  ```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 上传权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --served-model-name：vllm服务后台id。

  可在run_vllm.sh增加如下环境变量开启高阶配置：

  i. 配置环境变量。

  ```
  export USE_PFA_HIGH_PRECISION_MODE=1
  # PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针
  ```

```
对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

ii. 如果要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

在MoE模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

iii. 如果要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，如果不开启，目前vllm0.6.3版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
export ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
```

如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体可参考Eagle投机小模型训练章节**步骤五：训练生成权重转换成可以支持vLLM推理的格式**。

Step4 创建 pod

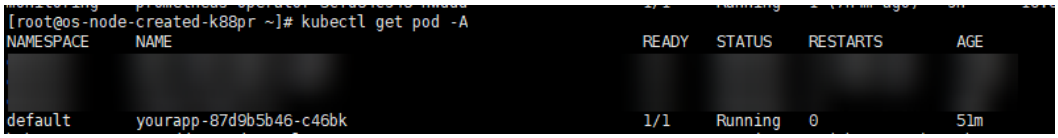
在节点自定义目录${node_path}下执行如下命令创建pod。

```
kubectl apply -f config.yaml
```

检查pod启动情况，执行下述命令。如果显示“1/1 running”状态代表启动成功。

```
kubectl get pod -A
```

图 4-170 启动 pod 成功



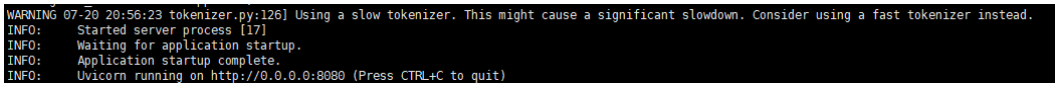
执行如下命令查看pod日志，如果打印类似下图信息表示服务启动成功。

```
kubectl logs -f ${pod_name}
```

参数说明：

- ${pod_name}：pod名，例如图4-170${pod_name}为yourapp-87d9b5b46-c46bk。

图 4-171 启动服务成功



Step5 推理请求

执行如下命令进入容器。

```
kubectl exec -it {pod_name} bash
```

参数说明：

- ${pod_name}：pod名，例如图4-170${pod_name}为yourapp-87d9b5b46-c46bk。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见表4-83。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
```

```
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step3 创建服务启动脚本**中设置的宿主机端口保持一致。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

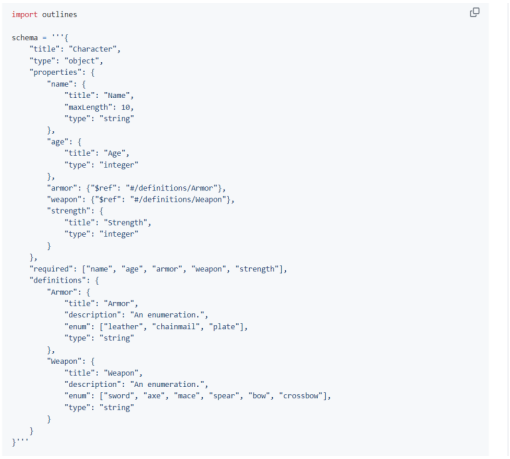
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-83 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。

参数	是否必选	默认值	参数类型	描述
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-172** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-84 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-85 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

4.10.4 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step3 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 执行如下命令进入容器。
   ```
   kubectl exec -it {pod_name} bash
   ```
   ${pod_name}：pod名，例如**图4-170**${pod_name}为yourapp-87d9b5b46-c46bk。
3. 进入benchmark_tools目录下，切换conda环境并安装依赖。
   ```
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
   conda activate python-3.9.10
   pip install -r requirements.txt
   ```
4. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python benchmark_parallel.py --backend openai --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```
   参数说明
   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务部署的IP。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
   - --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
   - --num-scheduler-steps: 服务启动如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。
   - --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
   - --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。

- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。
- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

5. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-173 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 执行如下命令进入容器。

```
kubectl exec -it {pod_name} bash
```

${pod_name}：pod名，例如图4-170${pod_name}为yourapp-87d9b5b46-c46bk。

2. 进入benchmark_tools目录下，切换conda环境。

```
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
conda activate python-3.9.10
```

3. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   generate_dataset.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。
   - --num-requests：输出数据集的数量，可以根据实际需求设置。

4. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   ```
   python benchmark_serving.py --backend openai --host 127.0.0.1 --port 8080 --dataset
   custom_datasets.json --dataset-type custom \
   --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
   1000 1000 1000 \
   --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。

- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 服务启动时如果配置了--num-scheduler-steps和--multi-step-stream-outputs=false，则需配置此参数与服务启动时--num-scheduler-steps一致。

5. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-174 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.10.5 推理精度测试

本章节介绍如何使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等。

约束限制

- 确保容器可以访问公网。
- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

步骤一：配置精度测试环境

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。

```
rm -rf lm-evaluation-harness/
git clone https://github.com/EleutherAI/lm-evaluation-harness.git
cd lm-evaluation-harness
git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
pip install -e .
```

2. 执行如下精度测试命令，可以根据参数说明修改参数。

```
lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization} \
--tasks ${task}  --batch_size ${batch_size}   --log_samples  --cache_requests true  --trust_remote_code  --output_path ${output_path}
```

参数说明:

- model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
  - vllm_path是模型权重路径；
  - max_model_len 是最大模型长度，默认设置为4096；
  - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
  - tensor_parallel_size是使用的卡数；
  - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
- model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
- tasks：评测数据集任务，比如openllm。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
- output_path：结果保存路径。

使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096 \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant" \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

4.10.6 推理模型量化

4.10.6.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

Step1 环境准备

1. 在节点自定义目录${node_path}下创建config.yaml文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
     app: infers
spec:
  replicas: 1
```

```
selector:
  matchLabels:
    app: infers
template:
  metadata:
    labels:
      app: infers
  spec:
    schedulerName: volcano
    nodeSelector:
      accelerator/huawei-npu: ascend-1980
    containers:
    - image: ${image_name}                  # 推理镜像名称
      imagePullPolicy: IfNotPresent
      name: ${container_name}
      securityContext:
        runAsUser: 0
      ports:
      - containerPort: 8080
      command:
      - "sleep"
      - "10000000000000000000"
      resources:
        requests:
          huawei.com/ascend-1980: "8"            # 需求卡数，key保持不变。
        limits:
          huawei.com/ascend-1980: "8"            # 限制卡数，key保持不变。
      volumeMounts:                             # 容器内部映射路径
      - name: ascend-driver                 #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/driver
      - name: ascend-add-ons              #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/add-ons
      - name: hccn                           #驱动hccn配置，保持不动
        mountPath: /etc/hccn.conf
      - name: localtime
        mountPath: /etc/localtime
      - name: npu-smi                           # npu-smi
        mountPath: /usr/local/sbin/npu-smi
      - name: model-path                        # 模型权重路径
        mountPath: ${model-path}
      - name: node-path                         # 节点自定义目录，该目录下包含pod配置文件config.yaml
        mountPath: ${node-path}
    volumes:                                  # 物理机外部路径
    - name: ascend-driver
      hostPath:
        path: /usr/local/Ascend/driver
    - name: ascend-add-ons
      hostPath:
        path: /usr/local/Ascend/add-ons
    - name: hccn
      hostPath:
        path: /etc/hccn.conf
    - name: localtime
      hostPath:
        path: /etc/localtime
    - name: npu-smi
      hostPath:
        path: /usr/local/sbin/npu-smi
    - name: model-path
      hostPath:
        path: ${model-path}
    - name: node-path
      hostPath:
        path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。

- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

2. 参考**Step4 创建pod**创建pod以用于后续进行模型量化

Step2 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 执行如下命令进入容器，并进入AutoAWQ目录下， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
kubectl exec -it {pod_name} bash
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoAWQ
bash build.sh
```

2. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step3 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step4 启动 AWQ 量化服务

参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q awq 或者--quantization awq
```

4.10.6.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 参考**Step1 环境准备**创建pod准备量化环境。
2. 执行如下命令进入容器，并进入AutoSmoothQuant目录下
   ```
   kubectl exec -it {pod_name} bash
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples
   ```
3. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-175 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 92.5        51                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+

+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+====================================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

4. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

5. 启动smoothQuant量化服务。

参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.10.6.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.10.6.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 创建服务启动脚本**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.10.7 Eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据训练eagle小模型，并使用自行训练的小模型进行eagle推理。支持llama1系列、llama2系列和Qwen2系列模型。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装Eagle。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的如下格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
{
"prefix":  "AAA"
"input": "BBB",
"output": "CCC"
}
```

执行convert_to_sharegpt.py 文件。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

其中：

- input_file_path：预训练json文件地址。
- out_file_name：输出的sharegpt格式文件地址。
- prefix_name：预训练json文件的前缀字段名称，例如：您是一个xxx专家，您需要回答下面问题。prefix_name可设置为None，此时预训练数据集只有input和output两段输入。
- input_name：预训练json文件的指令输入字段名称，例如：请问苹果是什么颜色。
- output_name output：预训练json文件的output字段名称，例如：苹果是红色的。
- code_type：预训练json文件编码，默认utf-8。

当转换为sharegpt格式时，prefix和input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

设置环境变量。

```
export EAGLE_TARIN_MODE=1
```

如果使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

如果使用其他数据集，需要先执行**步骤二：非sharegpt格式数据集转换（可选）**转换数据集格式为sharegpt格式。

执行如下脚本将sharegpt格式数据生成为训练data数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--npu_indices "0,1,2,3,4,5,6,7" \
--used_npus 8  \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048
--dtype bfloat16
```

其中

- outdir：生成的训练data地址。
- end_num：生成的data总条数。
- npu_indices：使用哪些NPU卡。
- used_npus：拉起的每个py脚本使用几个NPU，如果为70b则填写4或8，7b 13b则填1。
- model_type llama：使用模型类型，目前支持qwen2、llama1、llama2，其中llama1、llama2填写llama，qwen2填写为qwen2。
- model_name：模型地址。
- data_path：预训练数据集地址。
- seed 42：生成训练data所使用的seed，此处42为开源训练设定参数。
- max_length：模型的max_length。
- dtype：为模型dtype，默认为bfloat16。

执行完成后，记得unset环境变量，否则会导致后续推理服务启动出错。

```
unset EAGLE_TARIN_MODE
```

执行完成后，如果used_npus>1，则需要将训练生成data数据重新分配为8个文件夹，分配脚本为reassign_data_num.py。

```
python reassign_data_num.py --old_folder "./sharegpt_0_199_mufp16/" \
--new_folder "./sharegpt_0_199_mufp16/" \
--tp 8
```

- old_folder为上一步生成data的地址，填写到卡号的文件夹之前。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。
- new_folder为需要存储新的data的地址。命令中的./sharegpt_0_199_mufp16/"为举例，需要替换为实际地址。

- tp为需要切分成的文件夹数量，默认为8。

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

- tmpdir：即为步骤三中的outdir，训练data地址
- cpdir：为训练生成权重的地址
- configpath：为模型config文件的地址
- basepath：为大模型权重地址
- bs：为batch大小

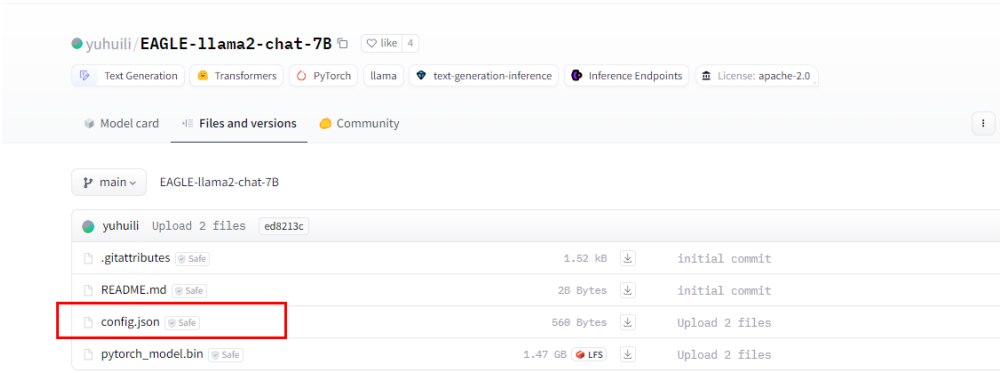
其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-176 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。

图 4-177 eagle config 文件



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

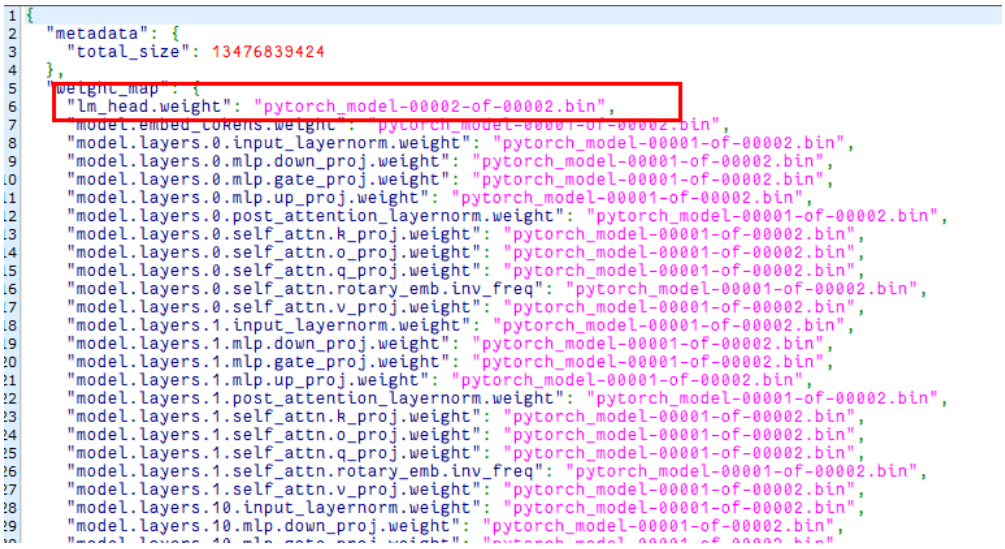
将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

然后在llm_tools/spec_decode/EAGLE文件夹，执行

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

- --base-path：为大模型权重地址，例如 ./llama2-7b-chat
- --draft-path：小模型权重地址，即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat
- --base-weight-name：为大模型包含lm_head的权重文件名，可以在base-path目录下的 model.safetensors.index.json 文件获取，例如llama2-7b-chat的权重名为pytorch_model-00001-of-00002.bin

图 4-178 权重文件名



--draft-weight-name 为小模型权重文件名，即刚才移动的.bin文件或者.safetensors文件。

4.10.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（ v0.6.3 ）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-86 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128

12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16
22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32

32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64
42	deepseek-llm-7b	1	16	1	32
43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32

51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	1	16`	1	32
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	internVL2-Llama3-76B	-	-	4	8
59	MiniCPM-v2.6	-	-	1	8
60	llama-3.1-405B-AWQ	-	-	8	32
61	qwen2-57b-a14b	-	-	2	16
62	deepseek-v2-lite-16b	2	4	1	4
63	deepseek-v2-236b	-	-	8	4
64	qwen2-vl-2B	1	8	1	8
65	qwen2-vl-7B	1	8	1	32
66	qwen2-vl-72B	-	-	4	32
67	qwen-vl	1	64	1	64
68	qwen-vl-chat	1	64	1	64

69	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.10.9 附录：大模型推理常见问题

问题 1：在推理预测过程中遇到 NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题 2：在推理预测过程中遇到 ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

问题 3：使用 llama3.1 系列模型进行推理时报错

使用llama3.1系模型进行推理时报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

解决方法：

升级transformers版本到4.43.1

```
pip install transformers --upgrade
```

问题 4：使用 SmoothQuant 进行 W8A8 进行模型量化时报错

使用SmoothQuant进行W8A8进行模型量化时报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42

```
pip install transformers==4.42 --upgrade
```

问题 5：使用 AWQ 转换 llama3.1 系列模型权重出现报错

使用AWQ转换llama3.1系列模型权重出现报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor'

解决方法：

该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

问题 6：使用 Qwen2-7B、Qwen2-72B 模型有精度问题，重复输出感叹号

检查**步骤六中4. 配置环境变量**章节中，高精度模式的环境变量是否开启。

问题 7：使用 autoAWQ 进行 qwen-7b 模型量化时报错

使用autoAWQ进行qwen-7b模型量化时报错：TypeError: 'NoneType' object is not subscriptable

解决方法：

修改qwen-7b权重路径下modeling_qwen.py第39行为SUPPORT_FP16 = True

问题 8：使用 benchmark-tools 对 GLM 系列模型进行性能测试报错

使用benchmark-tools对GLM系列模型进行性能测试报错TypeError: _pad() got an unexpected keyword argument 'padding_side'

解决方法：

1、下载最新的tokenization_chatglm.py，替换原来权重里的tokenization_chatglm.py。

https://huggingface.co/THUDM/glm-4-9b-chat/blob/main/tokenization_chatglm.py

https://huggingface.co/THUDM/chatglm3-6b/blob/main/tokenization_chatglm.py

或者2、修改tokenization_chatglm.py，在266行增加padding_side: str = "left"，如**图4-179**所示。

图 4-179 tokenization_chatglm.py

```
262     def _pad(
263             self,
264             encoded_inputs: Union[Dict[str, EncodedInput], BatchEncoding],
265             max_length: Optional[int] = None,
266             padding_side: str = "left",
267             padding_strategy: PaddingStrategy = PaddingStrategy.DO_NOT_PAD,
268             pad_to_multiple_of: Optional[int] = None,
269             return_attention_mask: Optional[bool] = None,
270     ) -> dict:
271         """
272         Pad encoded inputs (on left/right and up to predefined length or max length in the batch)
```

问题 9：使用 benchmark-tools 访问推理服务返回报错

使用benchmark-tools访问推理服务时，输入输出的token和大于max_model_len，服务端返回报错Response payload is not completed，见**图4-180**。

再次设置输入输出的token和小于max_model_len访问推理服务，服务端响应200，见**图4-181**。

客户端仍返回报错Response payload is not completed，见**图4-182**。

图 4-180 服务端返回报错 Response payload is not completed

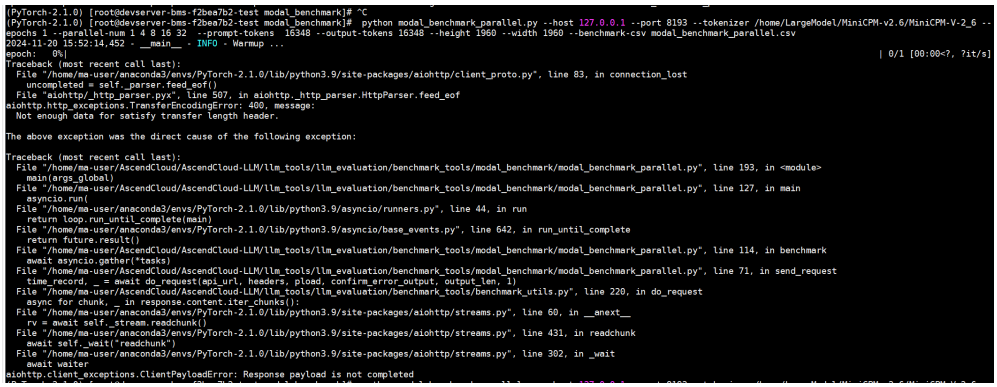


图 4-181 服务端响应 200

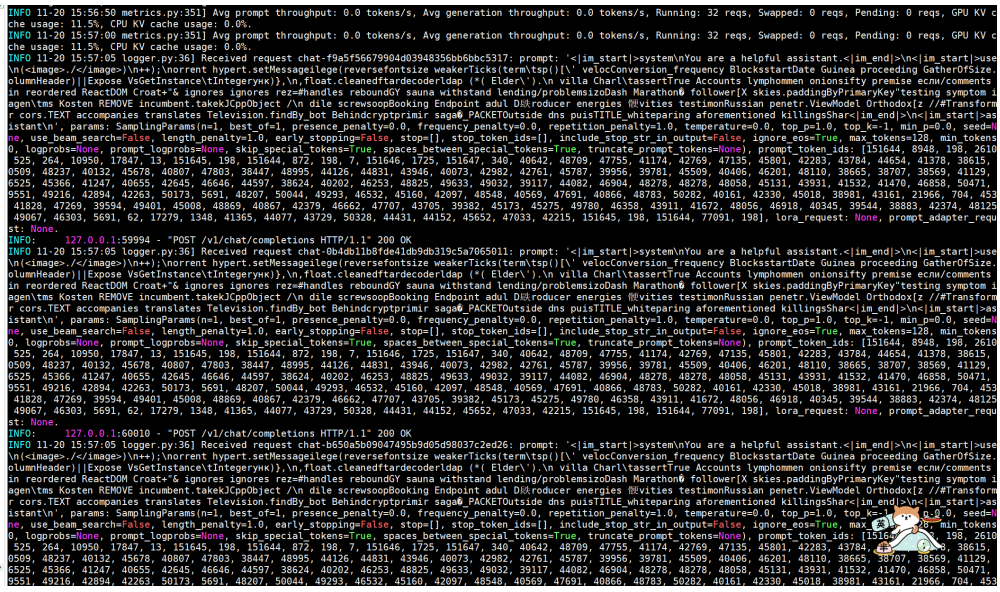
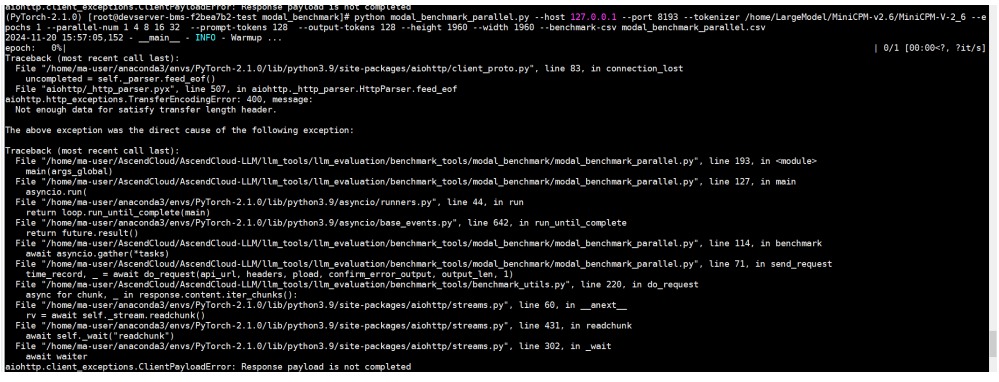


图 4-182 仍返回报错 Response payload is not completed



解决方法：

安装brotlipy后返回正确报错

```
pip install brotlipy
```

问题 10：使用 benchmark-tools 访问推理客户端返回报错或警告

使用benchmark-tools访问推理客户端返回报错或警告：actual output_tokens_length < expected output_len

图 4-183 benchmark-tools 访问推理客户端返回报错

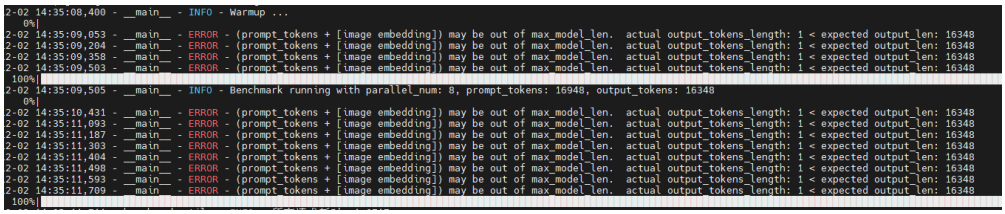
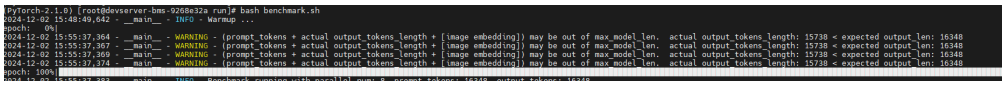


图 4-184 benchmark-tools 访问推理客户端返回警告



解决方法：

减少参数--prompt-tokens和--output-tokens的值，或者增大启动服务的参数--max-model-len的值。

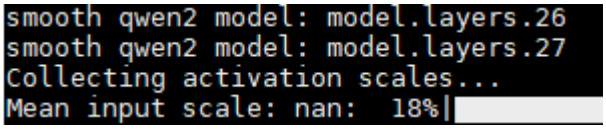
问题 11：使用离线推理时，性能较差或精度异常

解决方法：将block_size大小设置为128

```
from vllm import LLM, SamplingParams
llm = LLM(model="facebook/opt-125m", block_size=128)
```

问题 12：使用 SmoothQuant 做权重转换时，scale 显示为 nan 或推理时精度异常

图 4-185 权重转换 scale 显示为 nan



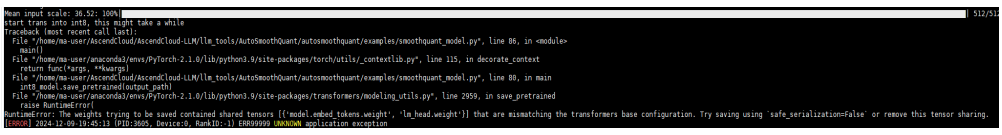
涉及模型：qwen2-1.5b, qwen2-7b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/utils/utils.py中的build_model_and_tokenizer函数，将torch_dtype类型从torch.float16改成torch.bfloat16

```
kwargs = {"torch_dtype": torch.bfloat16, "device_map": "auto"}
```

问题 13：使用 SmoothQuant 做权重转换时报错

图 4-186 权重转换报错



涉及模型：qwen2-1.5b, qwen2-0.5b

解决方法：修改AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples/smoothquant_model.py中的main函数，保存模型时将safe_serialization指定为False

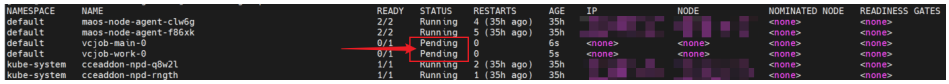
```
int8_model.save_pretrained(output_path,safe_serialization=False)
```

4.10.10 附录：工作负载 Pod 异常问题和解决方法

Pod 状态为 Pending

当Pod状态长时间为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。

图 4-187 pod 状态 pending



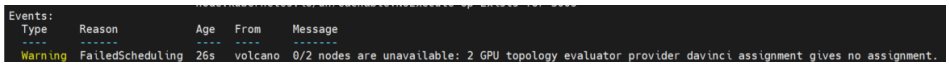
通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。

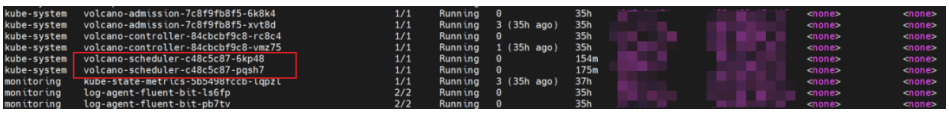
图 4-188 volcano 资源争抢



解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```

   图 4-189 scheduler

3. 若重启后，还是会Pending，建议多重复重启几次。

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

4.11 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.911）

4.11.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**表4-89**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。
- Lite Server驱动版本要求23.0.6
- PyTorch版本：2.1.0
- 确保容器可以访问公网。

文档更新内容

6.3.911版本相对于6.3.910版本新增如下内容：

- 文档中新增在数据预处理时，支持LLama-Factory格式的模板：
  - 支持Alpaca格式的数据，DATA_TYPE 环境变量需设置为AlpacaStyleInstructionHandler
  - 支持Sharegpt格式的数据，DATA_TYPE 环境变量需设置为SharegptStyleInstructionHandler
  - 已支持的系列模型模板：
    - qwen2.5系列
    - qwen2系列
    - qwen1.5系列
    - llama3.2系列
    - llama3.1系列
    - llama3系列
    - llama2系列

- glm4-9b
- mixtral-8x7b
- baichuan2-13b

训练支持的模型列表

本方案支持以下模型的训练，如表4-87所示。

表 4-87 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-190 操作流程图

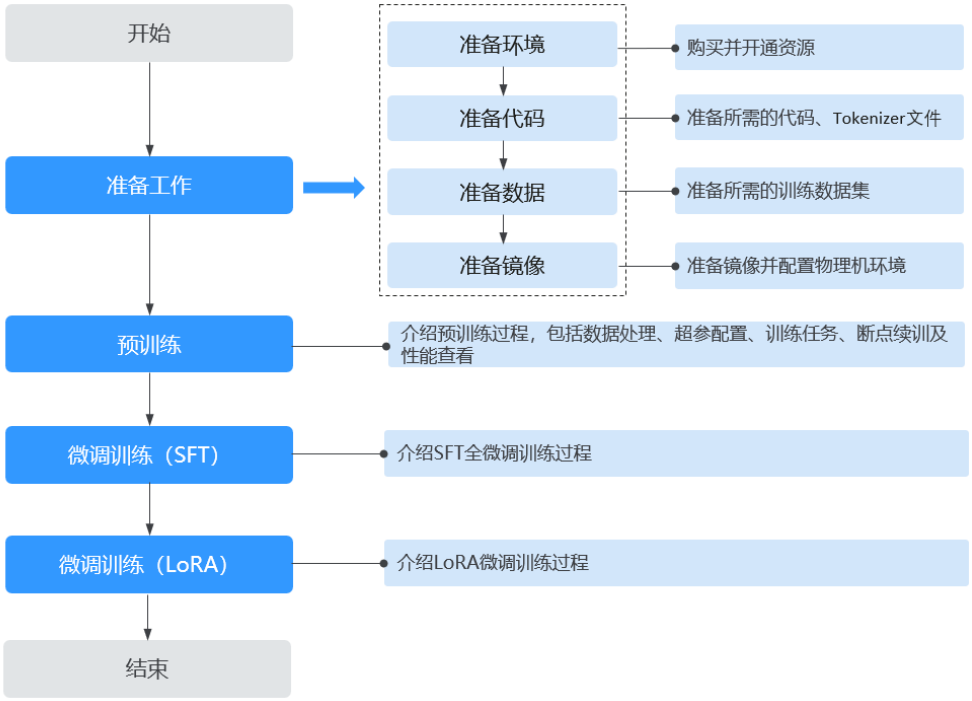


表 4-88 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。

阶段	任务	说明
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.11.2 准备工作

4.11.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-97**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.11.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-89**所示，模型列表、对应的开源权重获取地址如**表4-90**所示。

表 4-89 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

表 4-90 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了git clone repo_url 的方式下载，但是不支持断点续传，并且clone会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
   ├──llm_train                # 模型训练代码包
      ├──AscendSpeed           # 基于AscendSpeed的训练代码
         ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         ├──scripts/             # 训练需要的启动脚本
            ├──llama2            # llama2系列模型执行脚本的文件夹
            ├──llama3            # llama3系列模型执行脚本的文件夹
            ├──qwen              # Qwen系列模型执行脚本的文件夹
            ├──qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
            ├── ...
            ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
            ├── install.sh       # 环境部署脚本
         ├──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
```

```
|──llm_inference                  # 推理代码包
|──llm_tools                     # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizers                   #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                       #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                 #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

   说明

   多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.11.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS指令微调数据**：本案例中还支持MOSS格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他
  ```

```
人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

- **LLama-Factory Alpaca指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction对应的内容会与input对应的内容拼接后作为指令，即指令为instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
```

```
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

```
mkdir training_data
```

数据存放参考目录结构如下：

```
${workdir}（例如/home/ma-user/ws ）
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

> **说明**
>
> 多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.11.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-91 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b

表 4-92 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
```

```
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```

⚠ 注意

- 在执行scripts/install.sh安装命令时，需要确认机器是否已连通网络。若无法连通网络，可使用离线安装的方式，具体参考**离线训练安装包准备说明**。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 脚本中的 transformers 的版本。由默认 transformers==4.45.0 修改为：transformers==4.44.2
- 为了避免因使用不同版本的transformers库进行训练和推理而导致冲突的问题，建议用户分别为训练和推理过程创建独立的容器环境。

通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/            # 训练需要的启动脚本
  |——src/                # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/        # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/          # MindSpeed昇腾大模型加速库
  |——ModelLink/          # ModelLink端到端的大语言模型方案
      |——megatron/      # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

如果git下载代码时报错，请参见**Git下载代码时报错**解决。

4.11.3 执行预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和**0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-93**所示。其他超参均有默认值，可以参考**表4-96**按照实际需求修改。

表 4-93 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**步骤二 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
  <NNODES=4> <NODE_RANK=0>
  示例:
  # 第一台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

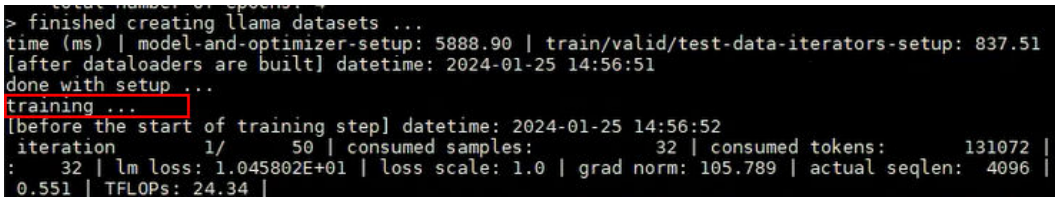
启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  ```
  示例：
  ```
  sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
  ```
  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_pretrain_7b.sh
  ```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-191 等待模型载入



最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

4.11.4 执行 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和**0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-93所示。其他超参均有默认值，可以参考表4-96按照实际需求修改。

表 4-94 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至ORIGINAL_HF_WEIGHT的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例：
  ```
  #第一台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  示例:
  sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
  ```
  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_sft_7b.sh
  ```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.11.5 执行 LoRA 微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-93**所示。其他超参均有默认值，可以参考**表4-96**按照实际需求修改。

表 4-95 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至ORIGINAL_HF_WEIGHT的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-97**所示。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:
  #第一台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
```

如果单机运行需要指定使用NPU卡的数量，可提前定义变量 NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令。

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/
llama2/0_pl_lora_7b.sh
```

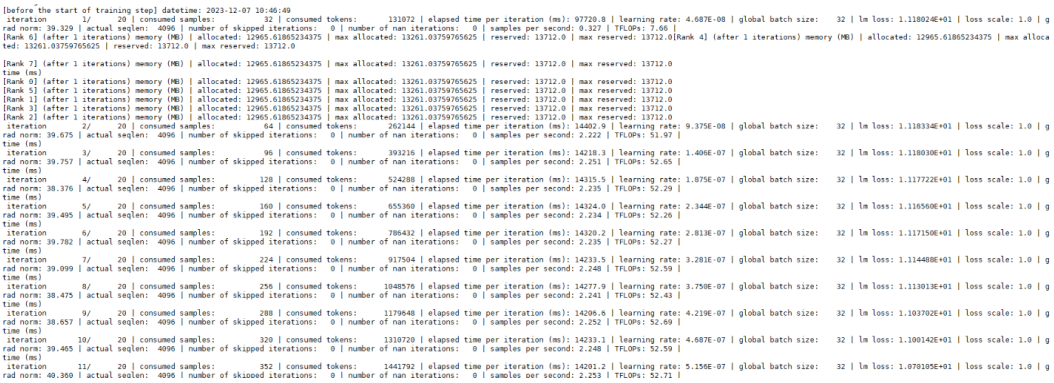
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

4.11.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-192 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

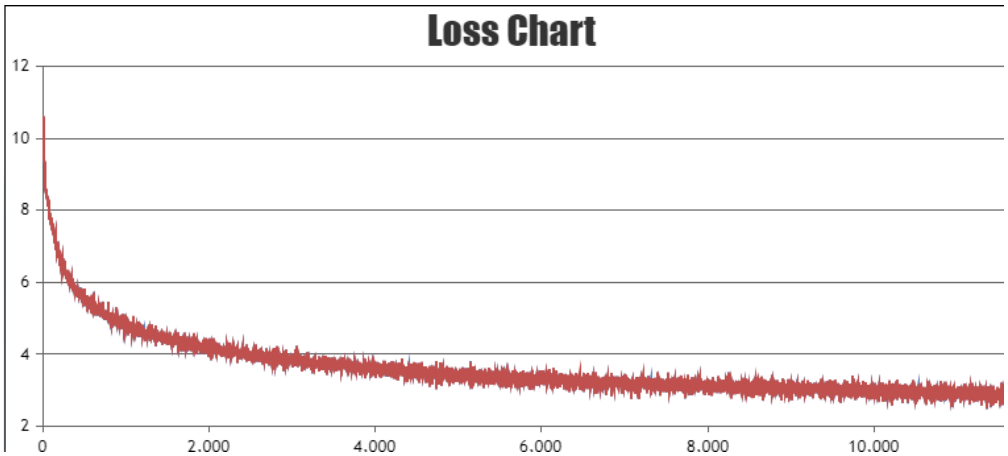
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-96**。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-193**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-193 Loss 收敛情况（示意图）



4.11.7 训练脚本说明参考

4.11.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以下参数取值主要以**llama2-70b预训练**为例，请根据实际模型修改。

表 4-96 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，会进行自动计算得出。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-97**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-97 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.11.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

ModelLink 预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/**

ModelLink 微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/**

LLama-Factory 微调数据集预处理参数说明

ModelLink开源仓已经支持LLama-Factory格式的数据预处理，目前仅支持sft全参微调，lora微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。可选项有['AlpacaStyleInstructionHandler ShareggptStyleInstructionHandler']。
  - AlpacaStyleInstructionHandler ：用于处理Alpaca风格的数据集。
  - ShareggptStyleInstructionHandler：用于处理sharegpt风格的数据集。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。
- --prompt-type：需要指定使用模型的template。已支持的系列模型可查看：**文档更新内容**。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/tasks/preprocess/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample

– self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

  – 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

  – 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

  – 推理prompt构造：由于GeneralPretrainHandler中未定义prompt格式用于训练，因此在推理时可直接自定义prompt内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  – 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

- instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
- input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
- output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

– 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

– 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) + self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话
c. 在单轮对话中
   i. 对user和assistant的文本进行清洗
   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
   iii. input_ids是user_ids和assistant_ids的拼接
   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
d. attention_mask是和input_ids等长的全1序列
e. 返回input_ids\attention_mask\labels的字典
f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context.
  \n Write a response that appropriately completes the request. \n Please note that you need to
  think through your response logically and step by step."

  "### Instruction:"
  ```

```
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **LlamaFactoryInstructionHandler解析**

  LlamaFactoryInstructionHandler是处理数据集的一个基类，继承自BaseDatasetHandler，实现对Llama-Factory格式数据集的处理。

  注意：仅支持微调场景，如：sft全参微调，lora微调。已支持的系列模型可查看：**文档更新内容**。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    tokenized_full_prompt = self._tokenize_prompt(messages, self.llama_factory_template, self.tokenizer.tokenizer)

    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)

    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  继承LlamaFactoryInstructionHandler的类都会复用 _filter 函数。根据 self.llama_factory_template 来获取模型的模板，随后通过 self._tokenize_prompt 函数将数据集中的关键内容进行拼接，并用于训练。若想详细了解 self._tokenize_prompt ，可查看具体代码。

- **AlpacaStyleInstructionHandler解析**

  AlpacaStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为Alpaca格式，格式如下：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **SharegptStyleInstructionHandler解析**

  SharegptStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为**Sharegpt**格式，格式如下：

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-98 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.11.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/** 目录下查看转换后的权重文件。

⚠ 注意

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-99 权重转换脚本中的环境变量

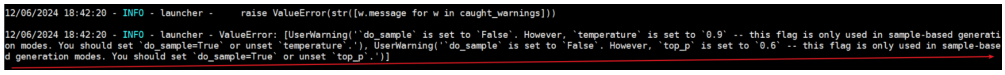
参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.11.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.45.0），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-194所示。

图 4-194 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-195所示。

图 4-195 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-196 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-197所示。

图 4-197 修改 ChatGLMv4-9B tokenizer 文件

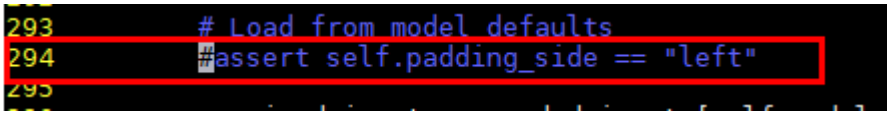
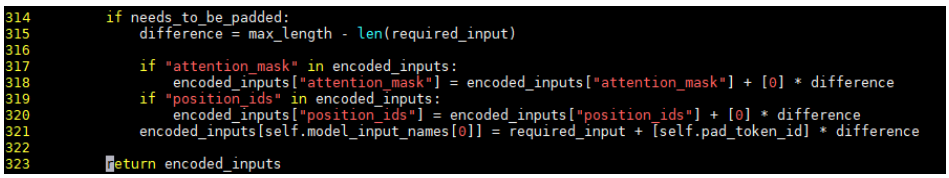


图 4-198 修改 ChatGLMv4-9B tokenizer 文件

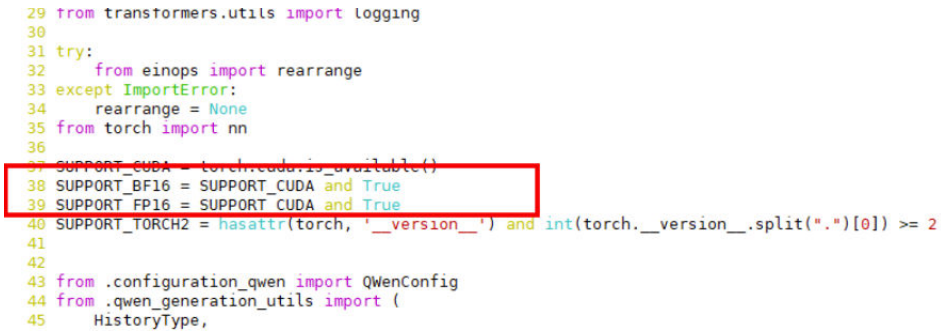


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-199所示。

图 4-199 修改 Qwen tokenizer 文件



4.11.7.5 离线训练安装包准备说明

在华为公有云平台，申请的资源一般要求连通网络。因此用户在准备环境时可以运行 **scripts/install.sh** 直接下载安装资源，或通过 **Dockerfile** 下载安装资源并构建一个新的镜像。

若用户的机器或资源池无法连通网络，并无法git clone下载代码、安装python依赖包的情况下，用户则需要找到已联网的机器（本章节以Linux系统机器为例）提前下载资源，以实现离线安装。用户可遵循以下步骤：

步骤一：资源下载

1. Python依赖包下载：进入 **scripts/install.sh** 文件中，找到需要安装的pip文件，如下列所示。直接下载pip文件，**注意：** 下载要求的版本。
```
pip install numpy==1.22.0 \
            transformers_stream_generator==0.0.5 \
            ...
```
2. 代码下载：访问 **scripts/install.sh** 文件中，找到需要git clone的文件，如下列所示。运行git clone命令，并git checkout切换到指定的版本。**注意：针对** Megatron-LM下载完成后，需要将megatron文件夹复制至ModelLink中。
```
git clone https://gitee.com/ascend/ModelLink.git
cd ModelLink
git checkout 8f50777
cd ..

git clone https://gitee.com/lmzwhu/Megatron-LM.git
cd Megatron-LM
git checkout -f core_r0.6.0
cp -r megatron ../ModelLink/
cd ..

git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout 4ea42a23
cd ..
```
完整的源码目录结构如下：
```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/           # MindSpeed昇腾大模型加速库
       |——ModelLink/           # ModelLink端到端的大语言模型方案
            |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
            |——...
```

步骤二：资源安装

1. 将资源上传至机器中，确保容器能够访问，并进入已创建的容器。
2. Python依赖包本地安装：进入pip文件所在的路径，并运行安装命令。如下列所示。
```
pip install numpy
pip install transformers_stream_generator
...
```
3. 代码安装：访问 **scripts/install.sh** 文件，在最后执行的命令中需要分别进入ModelLink、MindSpeed、AscendSpeed目录，并运行以下命令。其中${INSTALL_DIR}为AscendSpeed所在路径。
```
cd  ${INSTALL_DIR}/ModelLink
pip install -e .
cd ${INSTALL_DIR}/MindSpeed
pip3 install -e .
cd ${INSTALL_DIR}
pip install -e .
```

4.11.8 常见错误原因和解决方法

4.11.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-97**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

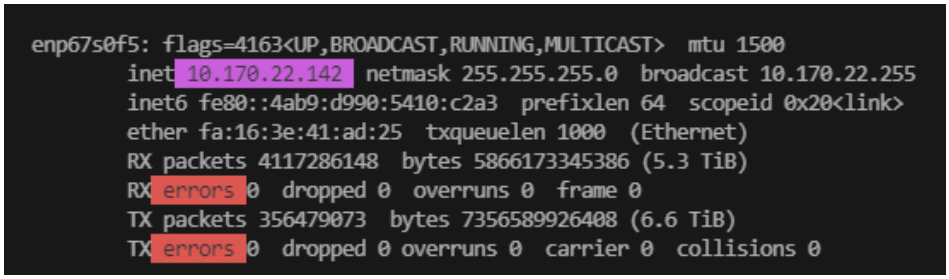
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.11.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-200 网卡名称错误



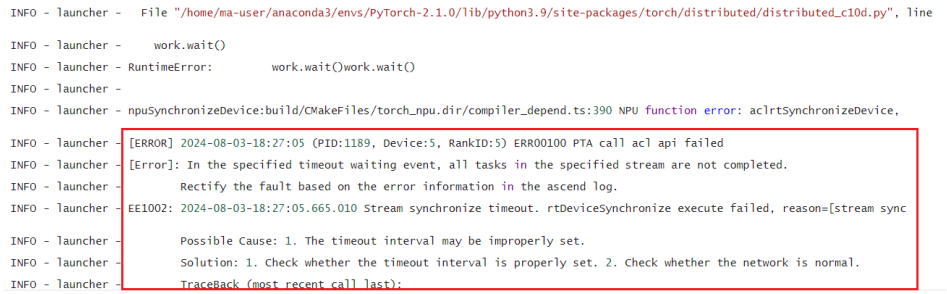

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.11.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-201 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际保存的权重。

4.11.8.4 Git 下载代码时报错

在执行scripts/install.sh安装命令或使用Dockerfile构建镜像时，如遇到git下载代码出现以下类似的报错信息，关闭git验证即可。

报错信息：

```
fatal: unable to access 'https://gitee.com/ascend/ModelLink.git/': error setting certificate verify locations:
CAfile: /etc/pki/tls/certs/ca-bundle.crt CApath: none
```

关闭git验证命令如下：

```
git config --global http.sslverify false
```

4.12 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.911）

4.12.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的不同训练阶段方案，包括指令监督微调、DPO偏好训练、RM奖励模型训练、PPO强化训练方案。

- DPO(Direct Preference Optimization)：直接偏好优化方法，通过直接优化语言模型来实现对大模型输出的精确把控，不用进行强化学习，也可以准确判断和学习到使用者的偏好，最后，DPO算法还可以与其他优化算法相结合，进一步提高深度学习模型的性能。
- RM奖励模型(Reward Model)：是强化学习过程中一个关键的组成部分。它的主要任务是根据给定的输入和反馈来预测奖励值，从而指导学习算法的方向，帮助强化学习算法更有效地优化策略
- PPO强化学习(Proximal Policy Optimization)：是一种在强化学习中广泛使用的策略优化算法。它属于策略梯度方法的一种，旨在通过限制新策略和旧策略之间

的差异来稳定训练过程。PPO通过引入一个称为“近端策略优化”的技巧来避免过大的策略更新，从而减少了训练过程中的不稳定性和样本复杂性。

- 指令监督式微调(Self-training Fine-tuning)：是一种利用有标签数据进行模型训练的方法。 它基于一个预先训练好的模型，通过调整模型的参数，使其能够更好地拟合特定任务的数据分布。 与从头开始训练模型相比，监督式微调能够充分利用预训练模型的知识和特征表示，从而加速训练过程并提高模型的性能。

训练阶段下有不同的训练策略，分为全参数训练、部分参数训练、LoRA、QLoRA，本文档主要支持全参数（Full）和LoRA、LoRA+。

- LoRA(Low-Rank Adaptation)： 这种策略主要针对如何在保持模型大部分参数固定的同时，通过引入少量可训练参数来调整模型以适应特定任务。
- LoRA+(Efficient Low Rank Adaptation of Large Models)：延续了LoRA的精髓进一步提升了在复杂任务上对大模型进行微调的效率和性能，核心在于其独特的学习率比率（loraplus_lr_ratio）机制，适用于那些需要精确控制模型微调过程的场景，当前该策略仅支持qwen1.5-7B指令监督式微调。
- 全参训练（Full）：这种策略主要对整个模型进行微调。这意味着在任务过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**表4-102**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.RC3。
- Server驱动版本要求23.0.6
- PyTorch版本：2.2.0
- Python版本：3.10
- 确保容器可以访问公网。
- 仅支持313T、376T、400T

训练支持的模型列表

本方案支持以下模型的训练，如**表4-100**所示。

表 4-100 支持的模型列表及权重文件地址

支持模型	支持模型参数量	权重文件获取地址
Llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
	llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
	llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-chat-hf**

支持模型	支持模型参数量	权重文件获取地址
Llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
	llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
Llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct/tree/main
	llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct/tree/main
Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
	qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
	qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
	qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
	yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
	qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
	qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
	qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
Qwen2_VL（支持多模态数据集）	qwen2_vl-2b	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
	qwen2_vl-7b	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
Falcon2	falcon-11B	https://huggingface.co/tiiuae/falcon-11B
GLM-4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本 4b556ad4d70c38924cb8c120adbf21a0012de6ce

支持模型	支持模型参数量	权重文件获取地址
Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
	qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
	qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
	qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
	qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
	llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

表 4-101 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
训练	启动训练	介绍各个训练阶段：指令微调、PPO强化训练、RM奖励模型、DPO偏好训练使用全参/lora训练策略进行训练任务、性能查看。

4.12.2 准备工作

4.12.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-107**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户如果购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.12.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-102**所示，模型列表、对应的开源权重获取地址如**表4-100**所示。

表 4-102 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                      # 模型训练代码包
       |——LLaMAFactory                 # 基于LLaMAFactory的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——demo.yaml            # 样例yaml配置文件
            |——demo.sh              # 指令微调启动shell脚本
            |——intall.sh            # 需要的依赖包
            |——LLaMA-Factory        # LLaMAFactory的代码目录
       |——AscendSpeed                # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws）
|——llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
    |—— LLaMAFactory              # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——demo.sh              # 指令微调启动shell脚本
          |——demo.yaml            # 样例yaml配置文件
          |——intall.sh            # 需要的依赖包
          |——LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考步骤三：启动容器镜像
              |——data             # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|—— tokenizers                    #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
    |—— Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表4-105表格中output_dir参数值
|—— saved_dir_for_output_lf         # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |—— ${model_name}               # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录服务器。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen2-72B
   ```

4.12.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-103 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241114095658-d7e26d8

表 4-104 模型镜像版本

模型	版本
CANN	cann_8.0.RC3
驱动	23.0.6
PyTorch	2.3.0

步骤一：检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三：启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如： chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

4.12.2.4 DockerFile 构建镜像（可选）

本章节主要介绍通过DockerFile文件构建训练镜像，将训练过程中依赖包封装使用，过程中需要连接互联网git clone，请确保环境可以访问公网，详解操作如下：

进入代码包Dockerfile文件同级目录：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory
```

构建新镜像：

```
docker build -t <镜像名称>:<版本名称> .
```

如无法访问公网则需配置代理，增加`--build-arg`参数指定代理地址确保访问公网。

```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx"
--network=host -t <镜像名称>:<版本名称> .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_2_ascend:20241106

注意

构建镜像前需保证Dockerfile文件内容中镜像名与本文档**镜像**保持一致，如不同则需修改为一致。

```
# 修改以下内容：
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/xxx
```

4.12.2.5 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前支持alpaca格式和sharegpt格式的微调数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加数据集描述；具体示例如下。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。
   ```
   cd /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
   ```
2. 将自定义原始数据（指令监督微调样例数据集：alpaca_gpt4_data.json.json）按照下面的数据存放目录要求放置。

   说明

   指令微调样例数据集alpaca_gpt4_data.json.json的下载链接：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws/llm_train ）
    |── LLaMAFactory/data
        |── alpaca_en_demo.json                # 代码原有数据集
        |── identity.json                      # 代码原有数据集
   ```

```
...
|── alpaca_gpt4_data.json                # 自定义数据集
```

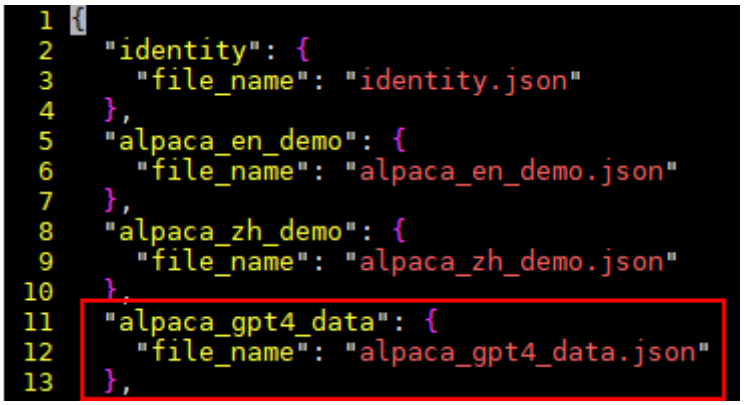
3. 更新代码目录下data/dataset_info.json文件。如使用以下示例数据集则命令如下。关于数据集文件格式及配置，更多样例格式信息请参考**data/README_zh.md** 的内容。

```
vim dataset_info.json
```

新加配置参数如下：

```
"alpaca_gpt4_data": {
  "file_name": "alpaca_gpt4_data.json"
 },
```

样例截图：



4.12.3 执行训练任务

步骤一：上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

步骤二：修改训练 yaml 文件配置

LlamaFactroy配置文件为Yaml文件，启动训练前需修改Yaml配置文件，Yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示。

1. 选择训练阶段类型。
   - 指令监督微调，复制**tune_yaml样例模板**内容覆盖demo.yaml文件内容。
   - DPO偏好训练，复制**dpo_yaml样例模板**内容覆盖demo.yaml文件内容。
   - PPO强化训练，先进行**RM奖励训练**任务后，复制**ppo_yaml样例模板**内容覆盖demo.yaml内容。
   - RM奖励训练，复制**rm_yaml样例模板**内容覆盖demo.yaml文件内容。

   说明

   1. DPO偏好训练、Reward奖励模型训练、PPO强化学习目前仅限制支持llama3系列。
   2. PPO训练暂不支持llama3-70B，存在已知的内存OOM问题，待社区版本修复。

2. 训练策略类型
   - 全参full，配置如下：
     ```
     finetuning_type: full
     ```
   - lora，如dpo仅支持此策略；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     ```
   - lora+，目前仅支持qwen1.5-7B**指令监督微调**；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     loraplus_lr_ratio: 16.0
     ```
3. 修改**yaml文件**(demo.yaml)的参数如**表4-105**所示。

表 4-105 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-107**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	**可修改**。指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配。取值可参考**表4-107**中**梯度累积值**列。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。

参数	示例值	参数说明
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

4. 是否选择加速深度学习训练框架Deepspeed，可参考**表4-107**选择不同的框架。
   - 是，选用ZeRO (Zero Redundancy Optimizer)优化器。
     - ZeRO-0，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z0_config.json
       ```
     - ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
       ```
       deepspeed: examples/deepspeed/ds_z1_config.json
       ```
     - ZeRO-2，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z2_config.json
       ```
     - ZeRO-2-Offload，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z2_offload_config.json
       ```
     - ZeRO-3，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z3_config.json
       ```
     - ZeRO-3-Offload，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z3_offload_config.json
       ```
   - 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。
5. 是否开启NPU FlashAttention融合算子，具体约束详见**NPU_Flash_Attn融合算子约束**
   - 是，配置以下参数。
     ```
     flash_attn: sdpa
     ```
   - 否，配置以下参数关闭。
     ```
     flash_attn: disabled
     ```
6. 是否使用固定句长。
   - 是，配置以下参数
     ```
     packing: true
     ```
   - 否，默认使用动态句长，**注释**掉packing参数。
7. 选用数据精度格式bf16或fp16二者选一，两者区别可查看**BF16和FP16说明**。
   - bf16，配置以下参数。
     ```
     bf16: true
     ```
   - fp16，相比bf16还需配置loss scale参数，配置如下。
     - 设置fp16为True。
       ```
       fp16: true
       ```
     - 修改deepspeed的"loss_scale"参数，配置如下。
       - 修改**ZeRO优化器**配置文件，如ZeRO2命令如下。
         ```
         cd /home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
         vim ds_z2_config.json
         ```
       - 使用fp16容易出现数值溢出，因此配置loss scale建议配置4096或4096以上：


```
"loss_scale": 4096,
```

8. 是否使用自定义数据集。
   - 是，参考**准备数据（可选）**，以指令监督微调数据集为例，配置以下参数：参考**表4-105**dataset_dir和dataset参数说明；如alpaca_gpt4_data.json数据集前缀则为alpaca_gpt4_data。
     ```
     dataset: alpaca_gpt4_data
     dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
     ```
   - 否，使用代码包自带数据集，注释掉dataset_dir参数，配置参数如下。
     - 指令监督微调/PPO数据集
       ```
       dataset: identity,alpaca_en_demo
       ```
     - 多模态数据集，如qwen2_vl系列模型
       ```
       dataset: mllm_demo,identity
       ```
     - RM/DPO，目前仅支持llama3系列模型
       ```
       dataset: dpo_en_demo
       ```
9. 是否使用falcon-11b、qwen2_vl系列、glm4-9b模型。
   - 是，更新配置或命令。
     - falcon-11b，参考**falcon-11B模型**替换文件。
     - glm4-9b，参考**glm4-9b模型**修改文件内容。
     - qwen2_vl系列，数据集为**多模态数据集**，如果前面步骤已配置请忽略。具体配置如下：

       数据集dataset配置：
       ```
       dataset: mllm_demo,identity
       ```
   - 否，忽略此步骤，执行下一步。
10. 如需其他配置参数，可参考**表4-106**按照实际需求修改。

步骤三：启动训练脚本

修改完yaml配置文件后，启动训练脚本。模型不同最少NPU卡数不同，NPU卡数建议值可参考**表4-107**。

1. **修改启动脚本demo.sh**

   进入代码目录{work_dir}/llm_train/LLaMAFactory下**修改启动脚本**，其中{work_dir}为容器挂载路径。

   a. 是否为PPO强化训练。
      - 是，demo.sh添加变量；
        ```
        export PYTORCH_NPU_ALLOC_CONF = expandable_segments:False
        ```
      - 否，demo.sh添加变量，开启虚拟显存。
        ```
        export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
        ```

   b. 修改yaml文件路径：修改demo.sh最后一行代码，将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，命令示例如下。
      - 修改前

        ```
        FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
        ```

- 修改后：

```
FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
```

2. **执行多机启动命令**（可选）

   多台机器执行训练启动命令如下。

```
多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
#第一台节点
sh demo.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh demo.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh demo.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh demo.sh  xx.xx.xx.xx  4  3
```

   以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

3. **执行单机启动命令**（可选）

   一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

   进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh demo.sh localhost 1 0
```

   单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

```
ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
```

   其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务。

训练成功标志

“***** train metrics *****”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1. 如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：训练常见问题**解决。
2. 训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：训练常见问题**问题3小节。
3. 大模型参数如（qwen2-72B、llama2-70B）等sft训练完成后多线程退出时报“torch.distributed.DistStoreError: Socket Timeout”时请参考**问题4：Error waiting on exit barrier错误**
4. 需要开启profiling功能进行性能数据采集和解析请参考**录制Profiling**
5. 训练过程中报"ModuleNotFoundError: No module named 'tyro'"可参考**依赖包tyro错误:"ModuleNotFoundError..."**

4.12.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-202 打印训练日志

```
  0%|          | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|▏         | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▎         | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍         | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌         | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋         | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊         | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉         | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█         | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏        | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▎        | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▎        | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▍        | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▋        | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▊        | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|█▉        | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表4-105**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件。

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

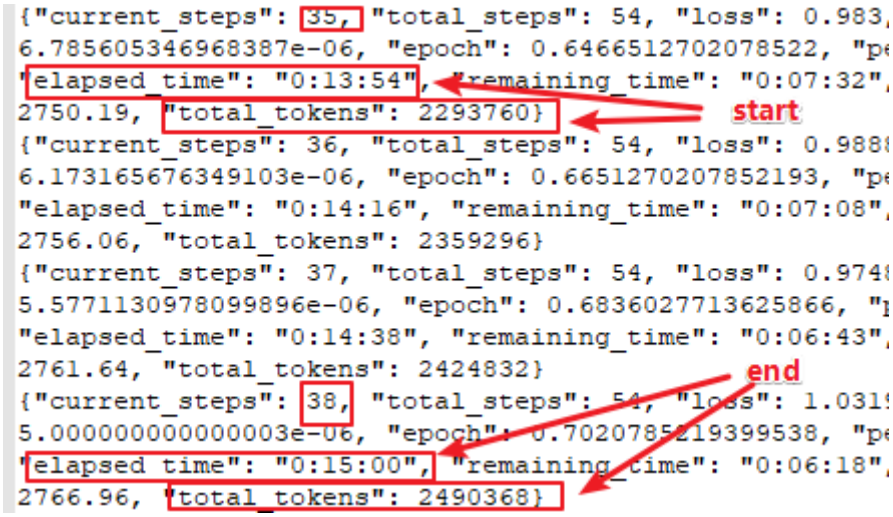
- 吞吐量（tokens/s/p）：可通过**表4-105**表格中**output_dir**参数值路径下的**trainer_log.jsonl**计算性能。取中间过程多steps平均值吞吐计算公式为：

  delta_tokens = end_total_tokens-start_ total_tokens

  delta_time = end_elapsed_time - start_elapsed_time

  吞吐值(tps) = delta_tokens / delta_time / 训练卡数

  如图所示：

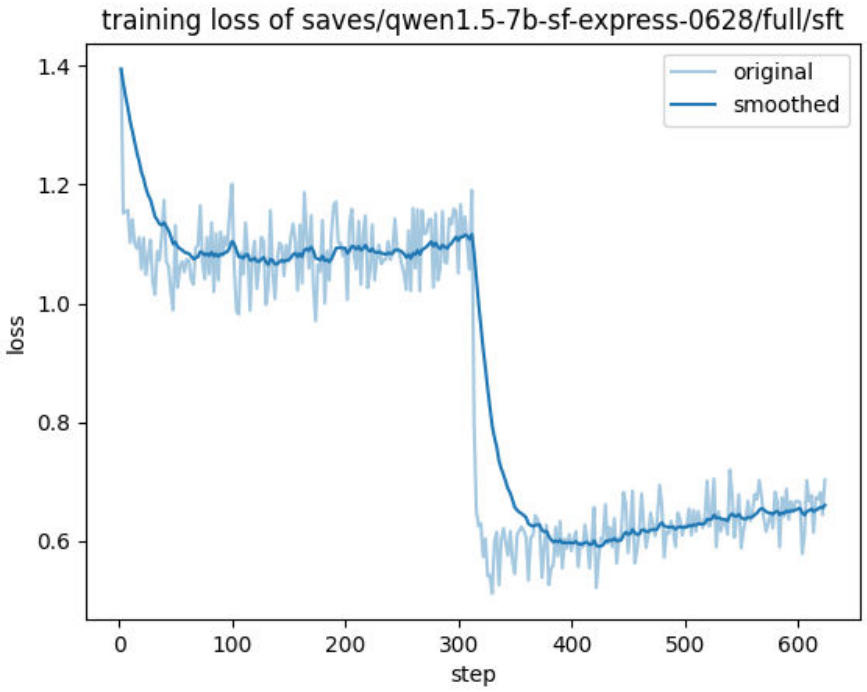
  

- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-105**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，将**trainer_log.jsonl**文件长传至可视化工具页面，如**图4-203**所示。

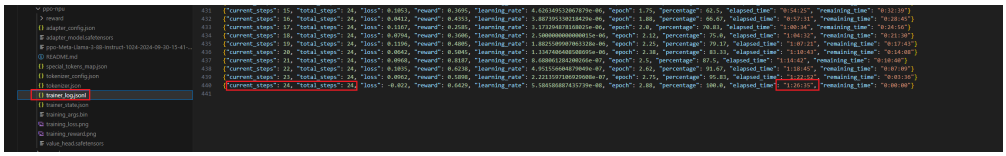
  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

图 4-203 Loss 收敛情况（示意图）



ppo训练结束不会打印性能。建议根据保存路径下的trainer_log.jsonl文件的最后一行总的训练steps和时间来判断性能。



4.12.5 训练 benchmark 工具

4.12.5.1 工具介绍及准备工作

本章节主要介绍针对LLaMAFactory开发的测试工具benchmark，支持训练、性能对比、下游任务评测、loss和下游任务对比能力。对比结果以excel文件呈现。方便用户验证发布模型的质量。所有配置都通过yaml文件设置，用户查看默认yaml文件即可知道最优性能的配置。

说明

目前仅支持SFT指令监督微调训练阶段。

代码目录

benchmark工具脚本存放在**代码包**AscendCloud-LLM-*xxx*.zip的LLM/LLaMAFactory/benchmark目录下，包含训练性能测试和训练精度测试脚本。

代码目录如下:

```
benchmark
├── config      #  默认的配置，使用前根据实际情况修改数据集路径dataset_dir、权重路径
model_name_or_path
  ├── deepspeed                  # zero配置文件目录
  ├── accuracy_cfgs.yaml           # 精度测试yaml配置文件
  ├── performance_cfgs.yaml        # 性能测试yaml配置文件
  ├── llama_factory_performance_baseline.yaml # LlamaFactory基于GP-Ant8训练性能基线值
  ├── llama_factory_accuracy_baseline.yaml  # LlamaFactory基于GP-Ant8训练精度基线值
  ......
  ├── ......
├── src          # 工具代码目录
  ├── accuracy.py        #精度测试脚本
  ├── common_utils.py    #获取训练日志工具
  ├── performance.py     #性能测试脚本
  ├── trainer.py         #训练启动脚本
├── data.tgz             # 样例数据
├── setup.py             # 构建工具包
```

约束限制

目前仅支持以下模型：

qwen2.5-7b

qwen2-7b

qwen1.5-7b

llama3.2-3b

llama3.1-8b

llama3-8b

llama2-7b

yi-6b

准备工作

1. 完成**准备工作**内容，生成benchmark-cli工具。
2. 解压版本包data.tgz：测试样例数据；比如工作目录为：/homa/ma-user/LLaMAFactory
   ```
   # 将默认数据解压config同级目录
   tar -zxvf ./benchmark/data.tgz ./benchmark/
   ```
3. 创建test-benchmark目录，该目录存放训练生成的权重文件及训练日志。
   ```
   # 任意目录创建
   mkdir test-benchmark
   ```
4. 修改yaml文件参数中model_name_or_path、dataset_dir和dataset或eval_dataset参数配置，修改**代码目录**下accuracy_cfgs.yaml或performance_cfgs.yaml文件内容，参数详解可参考**表4-105**。
   ```
   # 默认参数；根据自己实际要求修改
   ## accuracy_cfgs.yaml、performance_cfgs.yaml
   dataset_dir: /xxxx/benchmark/data/dataset
   dataset: gsm8k_train_alpaca
   model_name_or_path: /data/wulan1/model/qwen2.5-7b
   ## accuracy_cfgs.yaml
   eval_dataset: gsm8k_test
   ```
   样例yaml配置文件结构分为
   - base块：基础配置块
   - ModelName块：该模型所需配置的参数，如qwen2.5-7b块

样例截图如下：

```
.base: &base
  ### model
  ### method
  do_train: true
  stage: sft
  lora_target: all
  ### dataset
  dataset_dir: /data/wulan1/user/lmz/third-party-large-mode-labraries/LLM/L
  dataset: gsm8k_train_alpaca
  cutoff_len: 4096
  packing: true
  max_samples: 10240
  overwrite_cache: true
  preprocessing_num_workers: 16
  ### output
  logging_steps: 1
  save_steps: 5000
  plot_loss: true
  overwrite_output_dir: true
  include_tokens_per_second: true
  include_num_input_tokens_seen: true
  ### train
  per_device_train_batch_size: 1
  gradient_accumulation_steps: 8
  learning_rate: 1.0e-5
  num_train_epochs: 1.0
  lr_scheduler_type: cosine
  warmup_ratio: 0.1
  bf16: true
  ddp_timeout: 180000000
  flash_attn: sdpa

qwen2.5-7b:
  _base: &qwen2_5-7b
    <<: *base
    model_name_or_path: /data/wulan1/model/qwen2.5-7b
    template: qwen
  lora:
    <<: *qwen2_5-7b
    finetuning_type: lora
    output_dir: ./saves/qwen2.5-7b/sft_lora
    disable_gradient_checkpointing: true
  full:
```

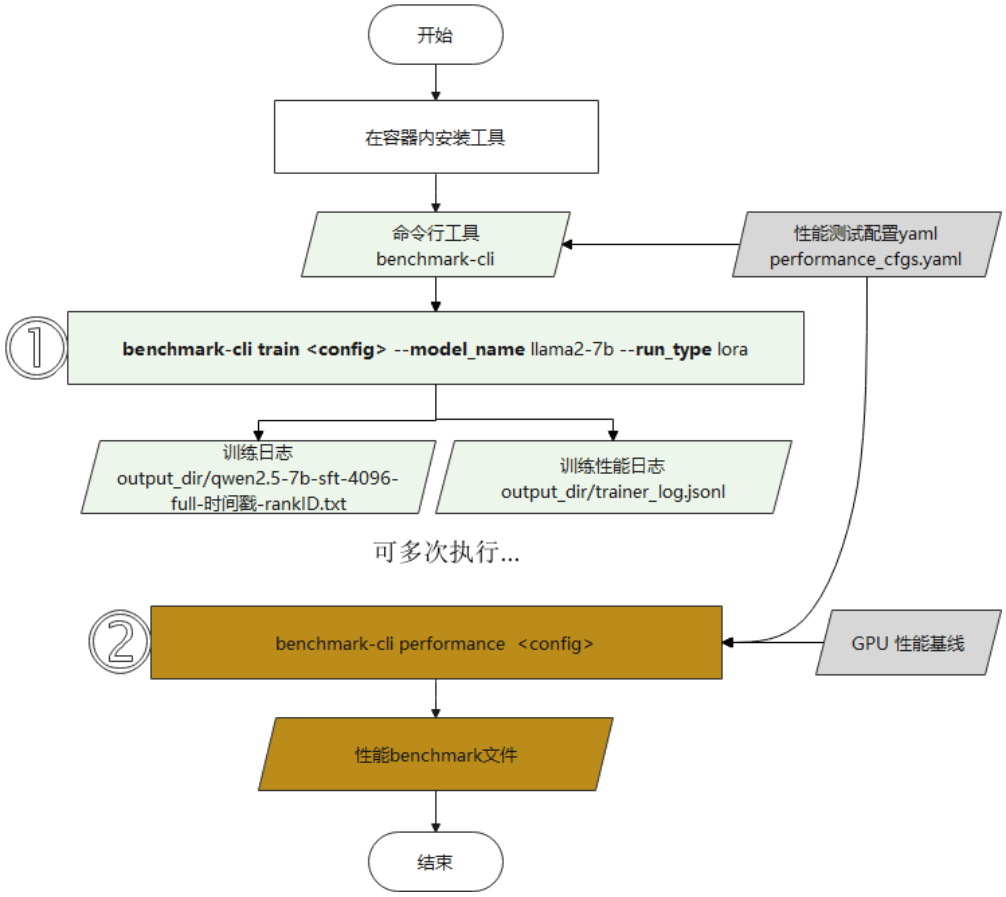
5. 开始训练测试，具体步骤参考**训练性能测试**或**训练精度测试**，根据实际情况决定。

4.12.5.2 训练性能测试

流程图

训练性能测试流程图如下图所示：

图 4-204 训练性能测试流程



执行训练任务

1. 进入**test-benchmark目录**执行训练命令，可以多次执行，卡数及其它配置参考**NPU卡数取值表**按自己实际情况决定。

   单机<可选>：

   ```
   # 默认8卡
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   # 指定设备卡数，如2卡
   ASCEND_RT_VISIBLE_DEVICES=0,1 benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   ```

   多机<可选>多机同时执行：

   ```
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>
   ```

   - <cfgs_yaml_file>：性能测试配置的yaml文件地址，如**代码目录**中performance_cfgs.yaml相对或绝对路径。
   - <model_name>：训练模型名，如qwen2-7b
   - <run_type>：训练策略类型及数据序列长度：【lora：4096-lora、full：4096-full、lora-8k：8192-lora、full-8k：8192-full】
   - --master_addr <master_addr>：主master节点IP，一般选rank0为主master。
   - --num_nodes <nodes>：训练节点总个数
   - --rank <rank>：节点ID

2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志，如qwen2.5-7b日志。

- qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt，打印吞吐值及训练参数

```
10/28/2024 16:54:54 - INFO - trainer - pkill -9 npu-smi
10/28/2024 16:54:58 - INFO - trainer - +++++++++ 性能: 3855.0588235294117 tokens/s/p +++++++++

10/28/2024 16:47:46 - INFO - trainer - {'do_train': True, 'stage': 'sft', 'lora_target':
'/data/wulan1/user/lmz/third-party-large-mode-labraries/LLM/LLaMAFactory/benchmark/data/
'overwrite_cache': True, 'preprocessing_num_workers': 16, 'logging_steps': 1, 'save_steps
'include_num_input_tokens_seen': True, 'per_device_train_batch_size': 1, 'gradient_accum
0.1, 'bf16': True, 'ddp_timeout': 180000000, 'flash_attn': 'sdpa', 'model_name_or_path':
'./saves/qwen2.5-7b/sft_lora', 'disable_gradient_checkpointing': True}
```

- qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt，打印训练过程中AICORE利用率

NpuID(Idx)	ChipId(Idx)	Pwr(W)	Temp(C)	AI Core(%)	AI Cpu(%)	Ctrl Cpu(%)	Memory(%)	Memory BW(%)
0	0	93.3	52	0	0	1	5	0
1	0	99.8	50	0	0	2	5	0
2	0	92.8	49	0	0	2	5	0
3	0	93.8	54	0	0	1	5	0
4	0	95.5	51	0	0	0	5	0

执行性能比较脚本

进入test-benchmark目录执行命令:

```
benchmark-cli performance <cfgs_yaml_file> --baseline <baseline> --o <output_dir>
```

- <cfgs_yaml_file>：性能测试配置的yaml文件地址，指**代码目录**中performance_cfgs.yaml相对或绝对路径，此配置文件为训练最优配置参数。
- --baseline <baseline>：<可选>GP-Ant8机器性能基线yaml文件路径，用户可自行修改，不填则使用工具自带基线配置，默认基线配置样例如下：

```
# 性能依次是 A800, ModelLink 313T, ModelLink 400T
qwen2.5-7b:
    full:    [2380, -1, -1]
    lora:    [3254, -1, -1]
    full-8k: [2394, -1, -1]
    lora-8k: [3199, -1, -1]
```

- --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径。

查看性能结果

任务完成之后会在test-benchmark目录下生成excel表格：

性能结果LLaMAFactory_train_performance_benchmark_<版本号>_<时间戳>.xlsx

表格样例如下：

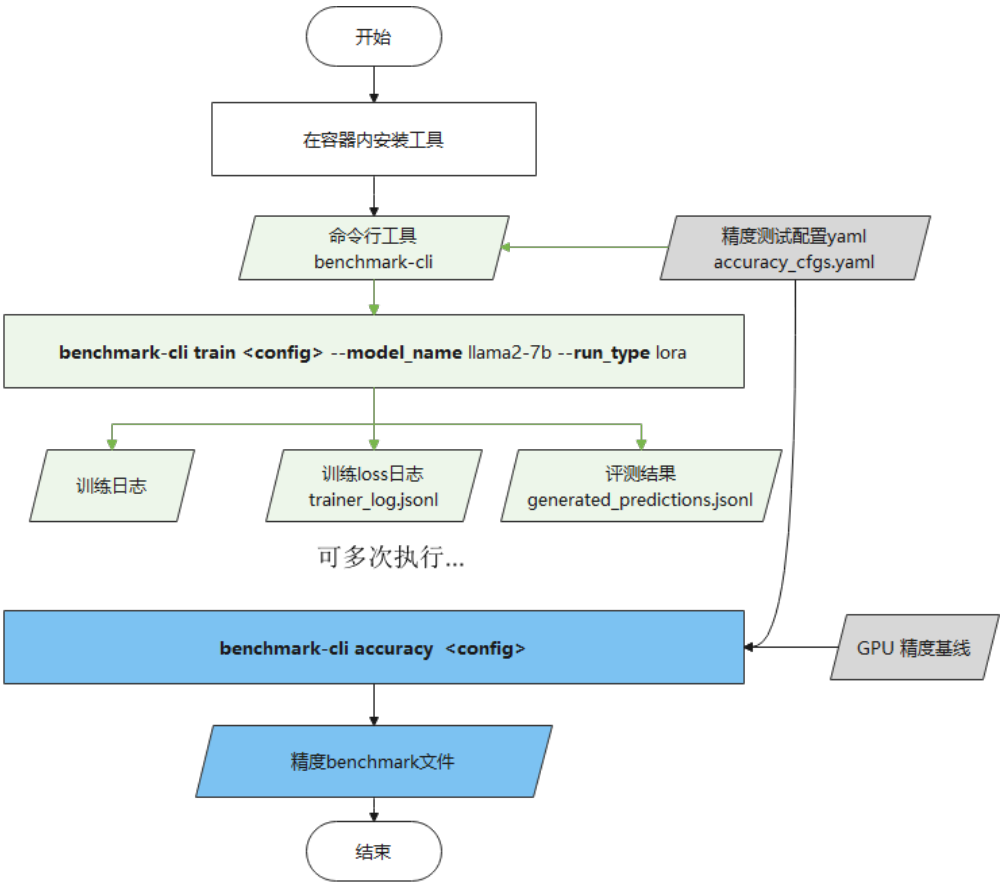
分类	训练框架	训练类型	序列长度	训练策略	ora_targ	加速框架	训练卡数	GBS	MBS	zero并行	FA	CORE 利用存	利用率	值(tok	性能值(t	tory VS	能基线(to	vs GP-Ant8
qwen2.5-7LLaMAFactSFT			4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	43	51	743.9821	-	-	-	-
qwen2-7b LLaMAFactSFT			4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	38.3	50	508.0854	-	-	2500	0.203234

4.12.5.3 训练精度测试

流程图

训练精度测试流程图如下图所示：

图 4-205 训练精度测试流程图



执行训练任务

1. 进入**test-benchmark目录**执行训练命令，可以多次执行，按自己实际情况。
   ```
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   ```
   - <cfgs_yaml_file>：精度测试配置的yaml文件地址，指**代码目录**中accuracy_cfgs.yaml相对或绝对路径
   - <model_name>：训练模型名，如qwen2.5-7b
   - <run_type>：训练策略类型及数据序列长度：【lora：4096-lora、full：4096-full】
2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志及权重文件，如qwen2.5-7b日志：
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt

执行精度比较脚本

进入test-benchmark目录执行命令:

```
benchmark-cli accuracy <cfgs_yaml_file> --o <output_dir> --baseline <baseline>
```

- <cfgs_yaml_file>：精度测试配置的yaml文件地址，如**代码目录**中accuracy_cfgs.yaml相对或绝对路径
- --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径

- --baseline <baseline>：<可选>GP-Ant8机器精度基线Yaml文件路径，不填则使用工具自带基线配置，默认基线配置样例如下：

```
qwen2-7b:
  full:
    loss: [0.4339,0.1433,0.2185,0.172,0.2376,0.2519,0.22
    score: 0.6815769522365428
  lora:
    loss: [0.4339,0.3791,0.3035,0.1714,0.1674,0.1443,0.1
    score: 0.7490523123578469
```

说明

客户使用工具自带精度基线Yaml则需使用accuracy_cfgs.yaml文件中默认配置，权重使用**表1 模型权重**中指定的Huggingface地址，数据指定data.tgz里面提供的gsm8k数据。

查看精度结果

任务完成之后会在test-benchmark目录下生成excel表格：

精度结果 LLaMAFactory_train_accuracy_benchmark_<版本号>_<时间戳>.xlsx

样例截图：

benchmark示例：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	warmup	学习率衰减策	数据集	MBS	GBS	平均相对	平均绝对	loss绝	推理评分差	NPU推理评分	GP-A推理评分
qwen2-7b	lora	233	adamw	0.0002	0	0	cosine	gsm8k_tra	1	32	0.418063	0.016463	0.0162	0.025018	0.728582	0.7536

具体精度benchmark包含如下字段：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	学习率warmup ratio	学习率衰减策略	数据集	MBS	GBS	loss平均相对误差	loss平均绝对误差	step1 loss绝对误差	推理评分差	NPU推理评分	GP-A推理评分

4.12.6 训练脚本说明

4.12.6.1 Yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-106 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。

参数	示例值	参数说明
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示当前的训练阶段。可选择值：sft、rm、ppo、dpo。 • sft代表指令监督微调； • rm代表奖励模型训练； • ppo代表PPO训练； • dpo代表DPO训练。
finetuning_type	full	用于指定微调策略类型，可选择值full、lora。 如果设置为full，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
lora_target	all	采取lora策略方法的目标模块，默认为all
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表4-107**中的**template列**

参数	示例值	参数说明
max_samples	50000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。
per_device_train_batch_size	1	指定每个设备的训练批次大小。
gradient_accumulation_steps	8	**必须修改**，指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可参考**表4-107**
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
max_steps	5000	非必填。表示训练step迭代次数。会自动计算得出。
save_steps	5000	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务。 ● 当参数值>=max_steps时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<max_steps时，生成模型会每经过save_steps次，保存一次模型版本。 模型版本保存次数=max_steps//save_steps+1
save_total_limit	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=max_steps//save_steps+1 ● 当参数值>1时，保存模型版本次数与save_total_limit的值一致。

参数	示例值	参数说明
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	2.0e-5	指定学习率
disable_gradient_checkpointing	true	关闭重计算，用于禁用梯度检查点，**默认开启**梯度检查点;在深度学习模型训练中用于保存模型的状态，以便在需要时恢复。这种技术可以帮助减少内存使用，特别是在训练大型模型时，但同时影响性能。True表示关闭重计算功能。
include_tokens_per_second include_num_input_tokens_seen	true	用于在训练过程中包含每秒处理的tokens和已经看到的输入tokens，方便计算性能。
reward_mode	/home/ma-user/ws/saves/rm/llama3-8b/lora	**PPO强化必修改**；指定Reward奖励任务完成时output_dir目录，PPO强化训练前提为完成Reward奖励学习；请根据实际规划修改。
loraplus_lr_ratio	16.0	lora+策略算法独有参数；设置Lora+算法的lambda值为16.0

tune_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true

# 全参
finetuning_type: full

# lora
# finetuning_type: lora
# lora_target: all

deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
```

```
template: qwen
cutoff_len: 4096
packing: true
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/tune/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

dpo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: dpo
do_train: true

# lora
finetuning_type: lora
lora_target: all

pref_beta: 0.1
pref_loss: sigmoid
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: dpo_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/dpo/llama3-8b/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 5.0e-6
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ppo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
reward_model: /home/ma-user/ws/saves/rm/llama3-8b/lora
### method
stage: ppo
do_train: true

# 全参
# finetuning_type: full
# reward_model_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: identity,alpaca_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/ppo/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0
bf16: true
ddp_timeout: 180000000
flash_attn: sdpa
include_tokens_per_second: true
include_num_input_tokens_seen: true
### generate
max_new_tokens: 512
top_k: 0
top_p: 0.9
```

rm_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
### method
stage: rm
do_train: true

# 全参
# finetuning_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: dpo_en_demo
template: llama3
cutoff_len: 4096
```

```
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/rm/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0
bf16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
  }
}
```

4.12.6.2 模型 NPU 卡数、梯度累积值取值表

不同模型推荐的训练参数和计算规格要求如表4-107所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推。

表 4-107 NPU 卡数、加速框架、梯度配置取值表

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
llama2	llama2	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		13B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3	llama3	70B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full		gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
llama3.1	llama3	8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3.2	llama3	1B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		3B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 2*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
Qwen2	qwen	72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		0.5/1.5B	lora/full	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
Qwen2_vl	qwen2_vl	2B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 2*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
				8192	gradient_accumulation_steps: 8	ZeRO-2-Offload	1*节点 & 8*Ascend
Qwen1.5	qwen	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full	8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
Qwen2.5	qwen	0.5B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
falcon2	falcon	11B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
GLM4	glm4	9B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
Yi	yi	6B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
		34B	full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
			full	8192	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

说明

以上参数为开启NPU FlashAttention融合算子，上述参数值仅供参考，请根据自己实际要求合理配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器、NPU节点数及其他配置。

具体优化工具使用说明可参考**如何选择最佳性能的zero-stage和-offloads**。

4.12.6.3 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

glm4-9b 模型

在训练开始前，需要修改glm4-9b模型中的tokenizer文件modeling_chatglm.py内容，具体步骤如下：

- 进入到tokenizer目录下{work_dir}/tokenizers/glm4-9B/，命令如下：

```
cd /home/ma-user/ws/tokenizers/glm4-9B
```

- 修改modeling_chatglm.py文件内容：

```
vim modeling_chatglm.py
# 注释掉以下两行内容
# if attention_mask is not None
    # attention_mask = ~attention_mask
```

样例图：

```
268 class SdpaAttention(CoreAttention):
269     def forward(self, query_layer, key_layer, value_layer, atte
270         if attention_mask is None and query_layer.shape[2] == k
271             context_layer = torch.nn.functional.scaled_dot_prod
272 
273 
274         else:
275             # if attention_mask is not None:
276                 # attention_mask = ~attention_mask
```

4.12.6.4 NPU_Flash_Attn 融合算子约束

1. query、key、value都需要梯度。默认开启重计算，则前向时qkv没有梯度，如果需要关闭重计算，可以在yaml配置 `disable_gradient_checkpointing: true` 关闭，但显存占用会直线上升。
2. attn_mask只支持布尔（bool）数据类型，或者为None。
3. query的shape仅支持 [B, N1, S1, D]，其中N1≤ 2048，D≤ 512并且dim== 4。
4. 对于GQA，key的shape是 [B, N2, S2, D]，其中N2 ≤ 2048，并且N1是N2的正整数倍。

不满足以上场景，则不能实现NPU_Flash_Attn功能。

4.12.6.5 BF16 和 FP16 说明

在大模型训练中，BF16（Brain Floating Point）和FP16（Float16）都是使用的半精度浮点数格式，但它们在结构和适用性上有一些重要的区别。

BF16：具有8个指数位和7个小数位。在处理大模型时有优势，能够避免在训练过程中数值的上溢或下溢，从而提供更好的稳定性和可靠性，在大模型训练和推理以及权重存储方面更受欢迎。

FP16：用于深度学习训练和推理过程中，可以加速计算并减少内存的占用，对模型准确性的影响在大多数情况下较小。与BF16相比在处理非常大或非常小的数值时遇到困难，导致数值的精度损失。

综上所述，BF16因其与FP32相似的数值范围和稳定性，在大模型训练中提供了优势。而FP16则在计算效率和内存使用方面有其独特的优点，但可能在数值范围和稳定性方面略逊一筹。因此，选择哪种格式取决于具体的应用场景和训练需求。

4.12.6.6 录制 Profiling

Ascend PyTorch Profiler是针对PyTorch框架开发的性能数据采集和解析工具，通过在PyTorch训练脚本中插入Ascend PyTorch Profiler接口，执行训练的同时采集性能数据，完成训练后直接输出可视化的性能数据文件，提升了性能分析效率。

Ascend PyTorch Profiler接口可全面采集PyTorch训练场景下的性能数据，主要包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等，可以全方位分析PyTorch训练时的性能状态。

录制命令如下：

在启动训练脚本基础：**步骤三：启动训练脚本** 新加DO_PROFILER=1和PROF_SAVE_PATH=/save_path参数，单机启动举例说明：

```
DO_PROFILER=1 PROF_SAVE_PATH=/save_path sh demo.sh localhost 1 0
```

- PROF_SAVE_PATH：Profiling录制结果存放路径
- DO_PROFILER：是否开启Profiling录制功能

4.12.7 附录：训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

1. 容器内执行以下命令，指定NPU内存分配策略的环境变量，开启动态内存分配，即在需要时动态分配内存，可以提高内存利用率，减少OOM错误的发生。
   ```
   export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True
   ```
2. 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。
3. 替换深度学习训练加速的工具或增加zero等级，可参考**模型NPU卡数、梯度累积值取值表**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。
   a. - ZeRO-0 数据分布到不同的NPU
   b. - ZeRO-1 Optimizer States分布到不同的NPU
   c. - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
   d. - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU
4. 增加卡数重新训练，未解决找相关人员定位。

问题 2：访问容器目录时提示 Permission denied

解决方法：

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: XXX not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

问题 4：Error waiting on exit barrier 错误

错误截图：

```
[ERROR] Error waiting on exit barrier. Elapsed: 300.105763912200 seconds
[ERROR] Traceback (most recent call last):
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py

[ERROR]     store_util.barrier(
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     synchronize(store, data, rank, world_size, key_prefix, barrier_timeout)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li

[ERROR]     agent_data = get_all(store, rank, key_prefix, world_size)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     store_get(f"{prefix}{node_rank}_FIN")
[ERROR] torch.distributed.DistStoreError: Socket Timeout
```

报错原因：多线程退出各个节点间超时时间默认为300s，时间设置过短。

解决措施：

修改容器内torch/distributed/elastic/agent/server/api.py文件参数：

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/
agent/server/api.py
```

修改def _exit_barrier(self)方法中的barrier_timeout参数，修改后如图4-206所示。

```
#修改前
barrier_timeout=self._exit_barrier_timeout
#修改后
barrier_timeout=3000
```

图 4-206 修改后的 barrier_timeout 参数

```
913     def _exit_barrier(self):
914         """
915         Define a barrier that keeps the agent process alive until all workers finish.
916
917         Wait for ``exit_barrier_timeout`` seconds for all agents to finish
918         executing their local workers (either successfully or not). This
919         acts as a safety guard against user scripts that terminate at different
920         times.
921         """
922         log.info(
923             "Local worker group finished (%s). "
924             "Waiting %s seconds for other agents to finish",
925             self._worker_group.state, self._exit_barrier_timeout
926         )
927         start = time.time()
928         try:
929             store_util.barrier(
930                 self._store,
931                 self._worker_group.group_rank,
932                 self._worker_group.group_world_size,
933                 key_prefix= TERMINAL_STATE_SYNC_ID,
934                 barrier_timeout=3000,
935             )
936             log.info(
937                 "Done waiting for other agents. Elapsed: %s seconds", time.time() - start
938             )
939         except SignalException as e:
940             log.warning("Got termination signal: %s", e.sigval)
941             raise
942         except Exception:
943             log.exception(
944                 "Error waiting on exit barrier. Elapsed: %s seconds",
945                 time.time() - start
946             )
```

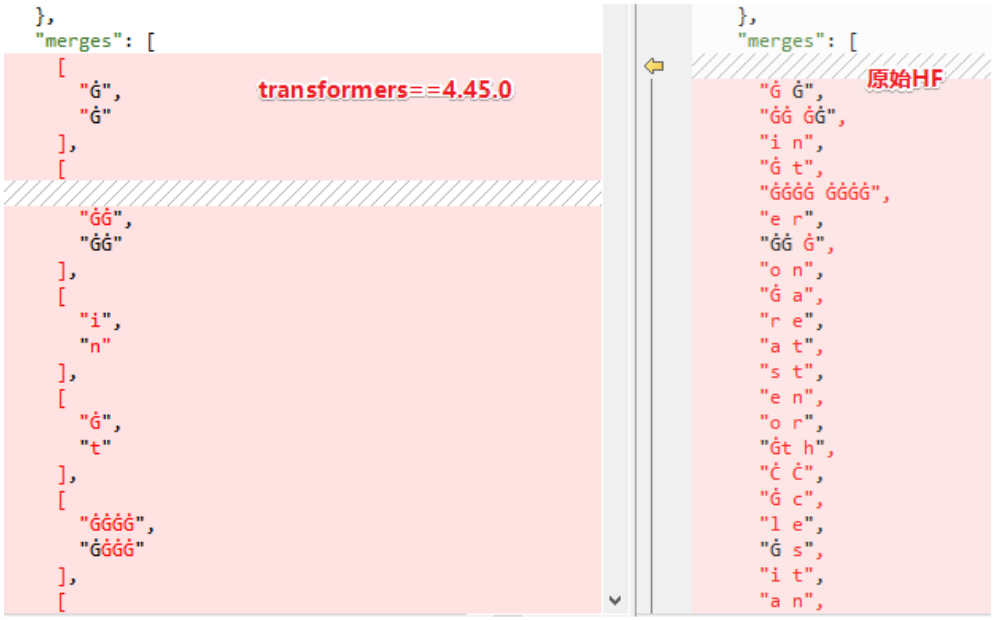
问题 5：训练完成使用 vllm0.6.0 框架推理失败：

错误截图：

```
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/tokenization_u
tils_fast.py", line 115, in __init__
    fast_tokenizer = TokenizerFast.from_file(fast_tokenizer_file)
Exception: data did not match any variant of untagged enum ModelWrapper at line 757272 column 1
```

报错原因：

训练时transformers版本要求为4.45.0，训练完成后保存的tokenizer.json文件中的“merges”时保存的是拆开的列表不是字符串，导致推理异常



解决措施，以下两种方法任选其一：

1. 更新transformes和tokenizers版本
   - GLM4-9B模型，容器内执行以下步骤：
     ```
     pip install transformers==4.43.2
     ```
   - 其它模型，容器内执行以下步骤：
     ```
     pip install transformers==4.45.0
     pip install tokenizers==0.20.0
     ```
2. 使用原始hf权重的tokenizer.json覆盖保存的tokenizer.json即可，如llama3-8b_lora具体过程如下：
   ```
   # 进入模型tokenizer目录
   cd /home/ma-user/ws/tokenizers/llama3-8b/
   # 替换tokenizer.json文件
   cp -f tokenizer.json /home/ma-user/ws/saves/rm/llama3-8b/lora/tokenizer.json
   ```

问题 6：训练过程中报依赖包 tyro 错误:"ModuleNotFoundError: No module named 'tyro'"

错误截图：

```
ModuleNotFoundError: No module named 'tyro'
```

报错原因：未指定tyro依赖包版本，导致安装依赖为最新0.9.0版本导致与其他依赖冲突

解决措施：任务前容器内更新'tyro'版本为0.8.14或以下版本

```
pip install tyro==0.8.14
```

4.13 主流开源大模型基于 Standard+OBS 适配 ModelLink PyTorch NPU 训练指导（6.3.911）

4.13.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.911版本相对于6.3.910版本新增如下内容：

- 文档中新增在数据预处理时，支持LLama-Factory格式的模板：
  - 支持Alpaca格式的数据，DATA_TYPE 环境变量需设置为AlpacaStyleInstructionHandler
  - 支持Sharegpt格式的数据，DATA_TYPE 环境变量需设置为SharegptStyleInstructionHandler
  - 已支持的系列模型模板：
    - qwen2.5系列
    - qwen2系列
    - qwen1.5系列
    - llama3.2系列
    - llama3.1系列
    - llama3系列
    - llama2系列
    - glm4-9b
    - mixtral-8x7b
    - baichuan2-13b

支持的模型列表

本方案支持以下模型的训练，如表4-108所示。

表 4-108 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce

序号	支持模型	支持模型参数量	权重文件获取地址
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-207 操作流程图

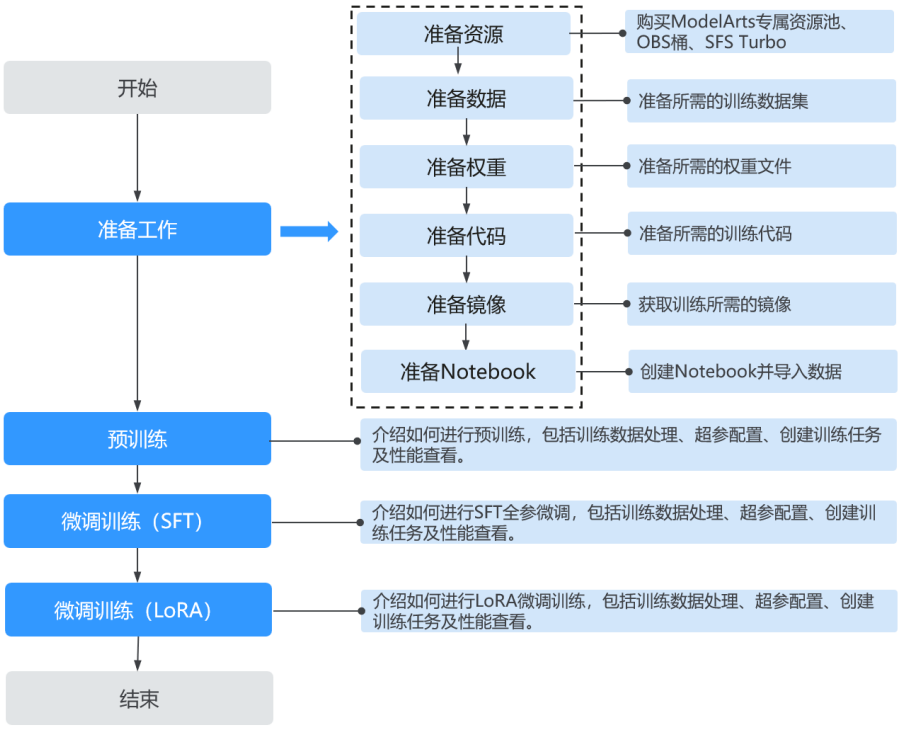


表 4-109 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.13.2 准备工作

4.13.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-115**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.13.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
     'id': '1',
     'url': 'https://simple.wikipedia.org/wiki/April',
     'title': 'April',
     'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

  如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：
  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst

  - MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
    - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
    - Human：数据集中每条数据的输入。
    - assistant：数据集中每条数据的输出。
  - 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

    - user_id: 用户的唯一不重复的ID值，必选。
    - excel_addr: 待处理的excel文件的地址，必选。
    - dataset_name: 处理后的数据集名称，必选。
    - proportion: 测试集所占份数，范围[1,9]，可选。
    - test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
    - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：

- conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
- from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
- value：具体的对话内容。
- system：系统提示词，用来为整个对话设定场景或提供指导原则。
- tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.13.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-108**。

   权重文件下载有如下几种方式，但不仅限于以下方式：

   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

     ```
     huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
     ```

     如果要下载指定版本的模型文件，则命令如下：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.13.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-110**所示。

表 4-110 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.911代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
              |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |——scripts/             # 训练需要的启动脚本
                   |——llama2          # llama2系列模型执行脚本的文件夹
                   |——llama3          # llama3系列模型执行脚本的文件夹
                   |——qwen            # Qwen系列模型执行脚本的文件夹
                   |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   |—— ...
                   |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   |—— install.sh       # 环境部署脚本
              |———src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference               # 推理代码包
|——llm_tools                   # 推理工具
```

代码上传至 OBS

将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|——llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
     |—— AscendSpeed              # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|—— training_data                                    #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
     |—— train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
     |—— alpaca_gpt4_data.json                          #微调数据文件
|—— tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
     |—— llama2-13b-hf
|—— models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
     |—— llama2-13b-hf
```

4.13.2.5 准备镜像

4.13.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-111 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行install.sh文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案**：在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

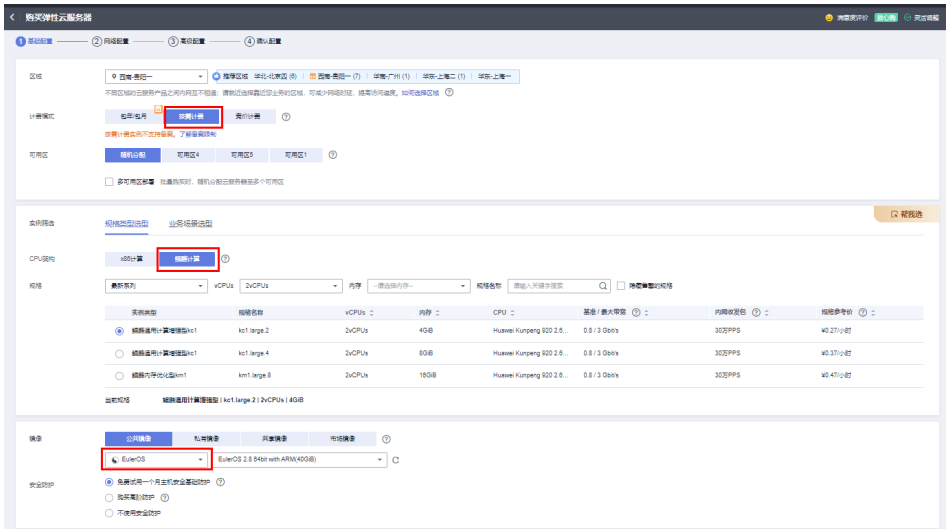
4.13.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

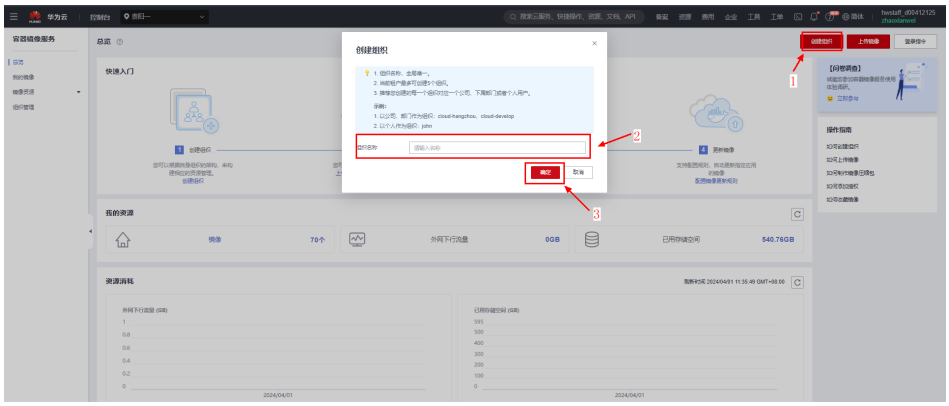
图 4-208 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-209 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

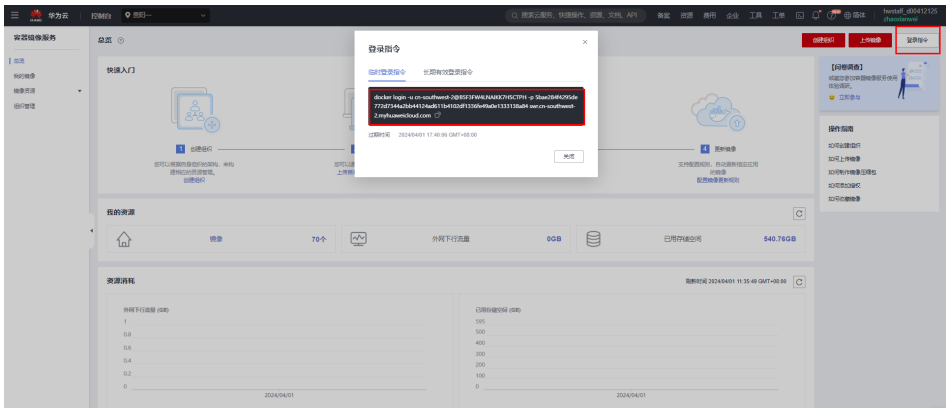
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-111**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-210 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.13.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-211**中都需要执行install.sh文件，来安装依赖以及下载完整代码。命令如下：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

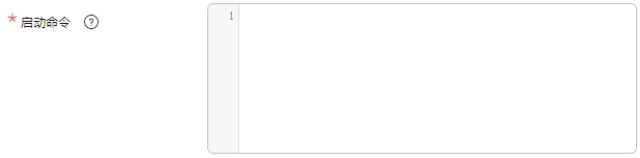
⚠ 注意

- 使用基础镜像的方法，需要确认训练作业的资源池是否联通公网，否则执行install.sh 文件时下载代码会失败。因此可以选择配置网络或使用**ECS中构建新镜像**的方法。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 中的transformers 的版本。

  由默认 transformers==4.45.0 修改为：transformers==4.44.2

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-211 训练作业启动命令



4.13.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-110**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.911-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面

   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```

2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。

   ```
   FROM {image_url}
   ```

3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。

   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```

   ⚠ 注意

   若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的transformers 的版本。

   由默认 transformers==4.45.0 修改为：transformers==4.44.2

4. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网

   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```

如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

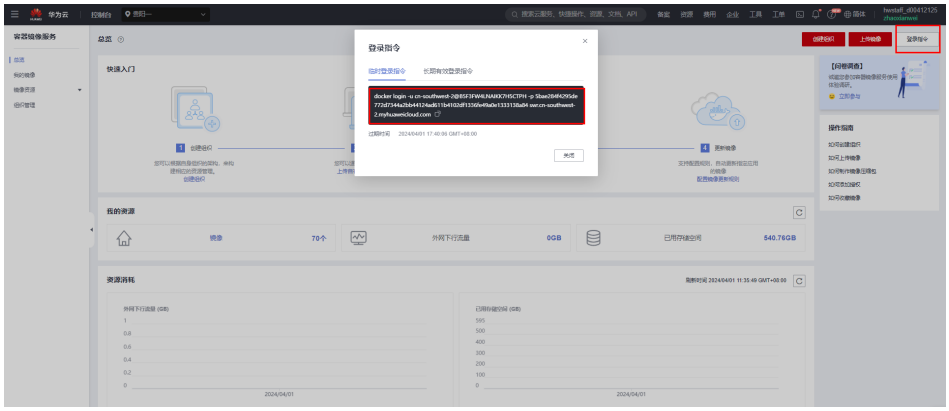
```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-212 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.13.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中，如果用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-213 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，如果需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-214 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
```

```
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

4.13.3 预训练

前提条件

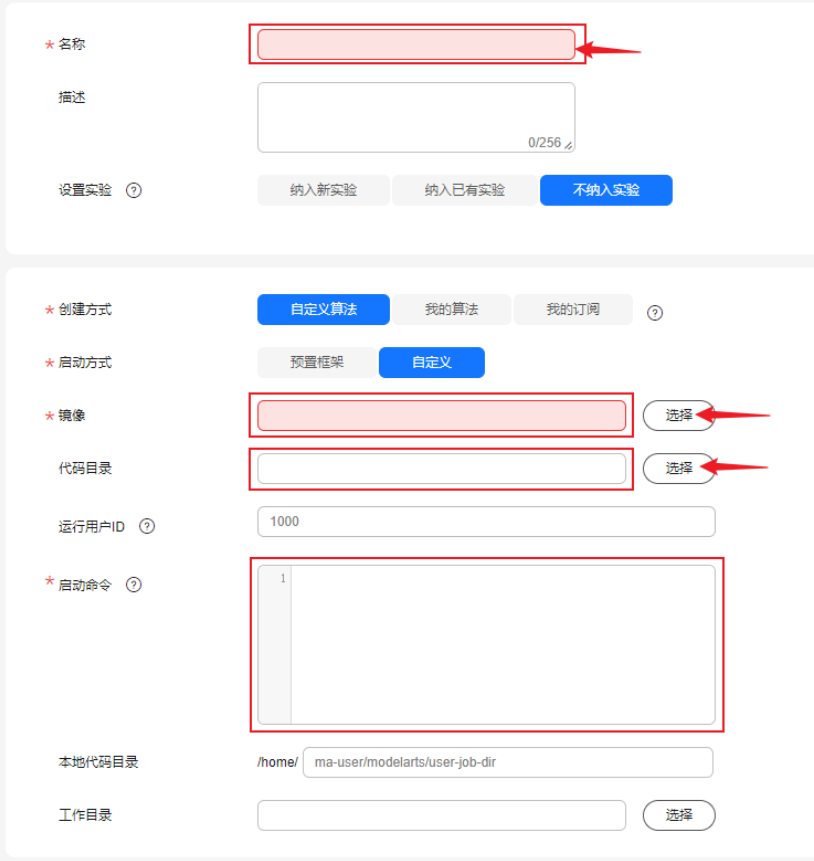
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-215 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-112表格中的配置进行填写。

表 4-112 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。

环境变量	示例值	参数说明
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

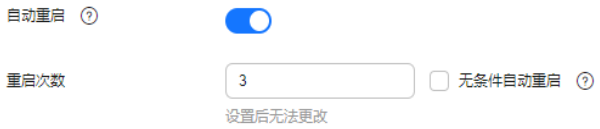
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-216 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-115**进行配置。

图 4-217 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.13.4 SFT 全参微调训练

前提条件

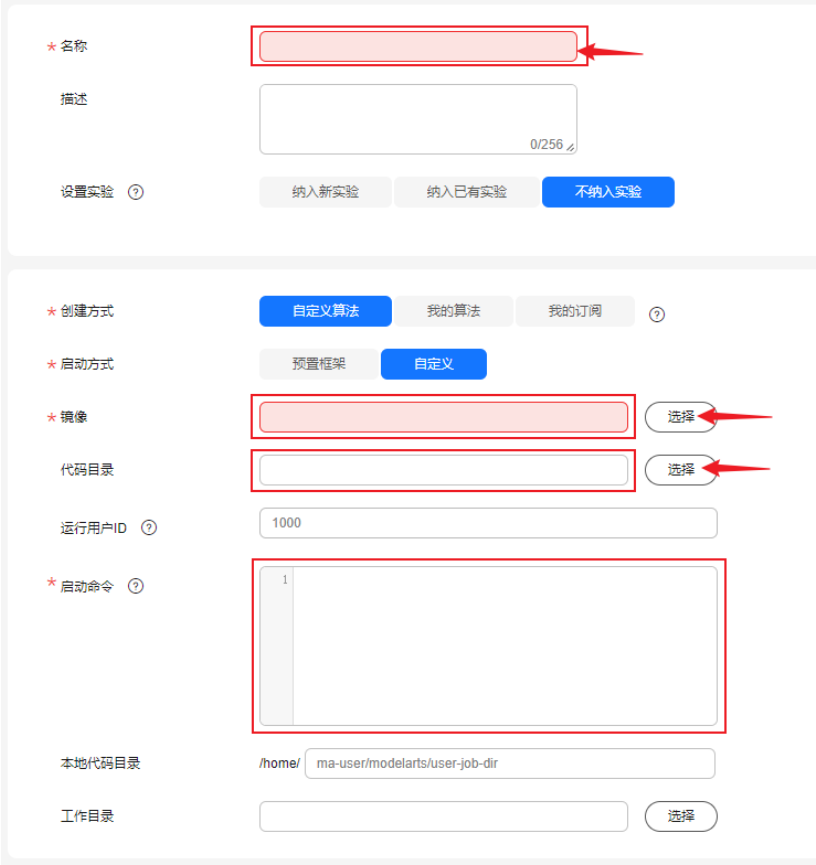
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-218 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-112表格中的配置进行填写。

图 4-219 环境变量

表 4-113 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。

环境变量	示例值	参数说明
SAVE_INTERVAL	1000	用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

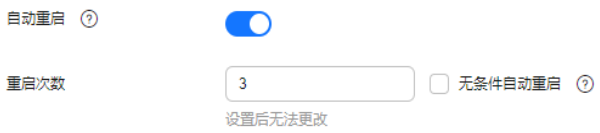
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-220 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-115**进行配置。

图 4-221 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.13.5 LoRA 微调训练

前提条件

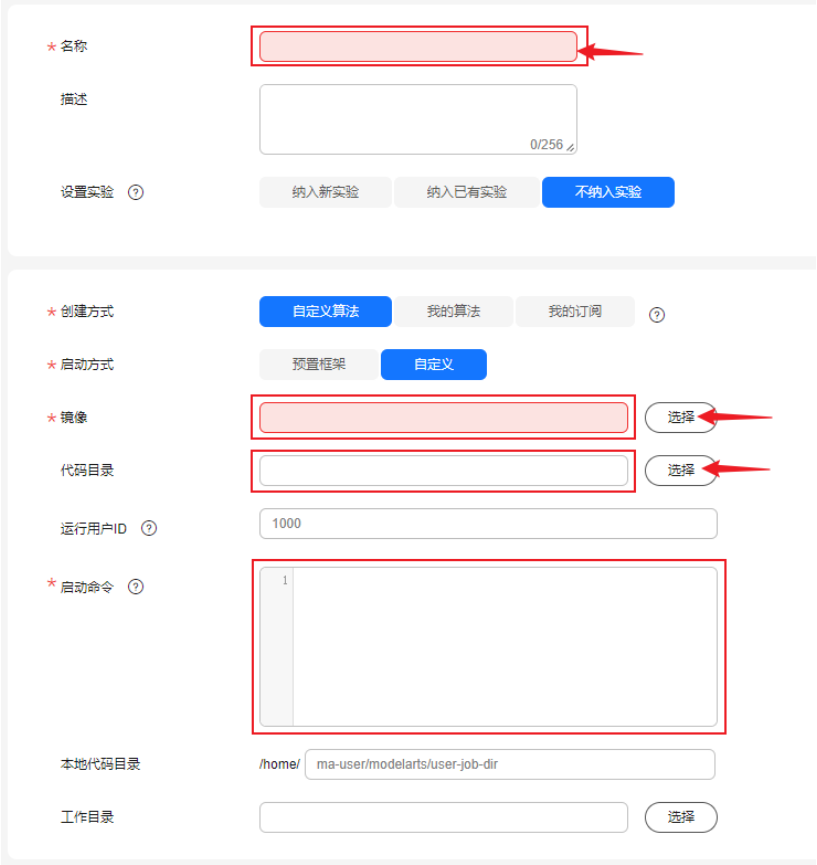
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-222 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-112表格中的配置进行填写。

表 4-114 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。

环境变量	示例值	参数说明
SAVE_INTERVAL	1000	用于模型中间版本地保存。 • 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 • 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 • 当参数不设置或<=0时，不会触发效果。 • 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 • 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

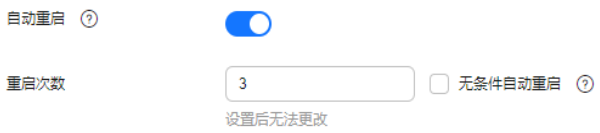
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-223 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考表4-115进行配置。

图 4-224 选择资源池规格

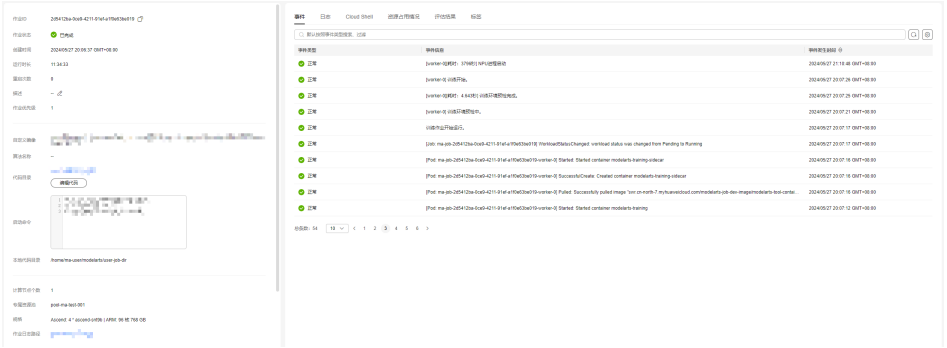
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考查看日志和性能章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看模型开发简介。

4.13.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

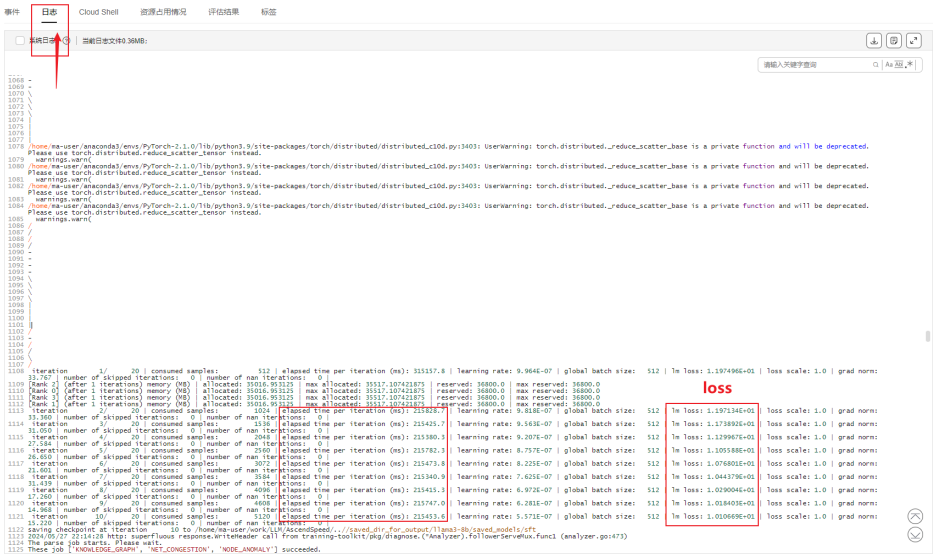
图 4-225 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-226 查看日志和性能



4.13.7 训练脚本说明

4.13.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-115所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-115 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.13.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。如果未进行数据集预处理，则会自动执行**scripts/llama2/1_preprocess_data.sh**。

ModelLink 预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

ModelLink 微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

LLama-Factory 微调数据集预处理参数说明

ModelLink开源仓已经支持LLama-Factory格式的数据预处理，目前仅支持sft全参微调，lora微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。

- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。可选项有['AlpacaStyleInstructionHandler SharegptStyleInstructionHandler']。
  - AlpacaStyleInstructionHandler ：用于处理Alpaca风格的数据集。
  - SharegptStyleInstructionHandler：用于处理sharegpt风格的数据集。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。
- --prompt-type：需要指定使用模型的template。已支持的系列模型可查看：**文档更新内容**。

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
```

```
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

  ```
  [
    {"text": "document"},
    {"other keys": "optional content"}
  ]
  ```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

  ```
  doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
  ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

  ```
  def _filter(self, sample):
      messages = self._format_msg(sample)
      full_prompt = self.prompter.generate_training_prompt(messages)
      tokenized_full_prompt = self._tokenize(full_prompt)
      if self.args.append_eod:
          tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
          tokenized_full_prompt["attention_mask"].append(1)
          tokenized_full_prompt["labels"].append(self.tokenizer.eod)
      if not self.train_on_inputs:
          user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
              self.prompter.template.assistant_token + "\n"
          tokenized_user_prompt = self._tokenize(user_prompt)
          user_prompt_len = len(tokenized_user_prompt["input_ids"])
          tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
      for key in self.args.json_keys:
          tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
      return tokenized_full_prompt
  ```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

    ```
    [
    {
        "instruction": "指令（必填）",
        "input": "输入（选填）",
        "output": "模型回答（必填）",
    }
    ]
    ```

  - 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

    ```
    "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."
    ```

```
"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

– 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行
  b. 循环处理其中的单轮对话
  c. 在单轮对话中
     i. 对user和assistant的文本进行清洗
     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
     iii. input_ids是user_ids和assistant_ids的拼接
     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
  d. attention_mask是和input_ids等长的全1序列
  e. 返回input_ids\attention_mask\labels的字典
  f. 处理完单一sample

  注：labels中用-100填充的地方，表示会被loss_mask给mask掉

  – 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼

接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **LlamaFactoryInstructionHandler解析**

  LlamaFactoryInstructionHandler是处理数据集的一个基类，继承自BaseDatasetHandler，实现对Llama-Factory格式数据集的处理。

  注意：仅支持微调场景，如：sft全参微调，lora微调。已支持的系列模型可查看：**文档更新内容**。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    tokenized_full_prompt = self._tokenize_prompt(messages, self.llama_factory_template,
self.tokenizer.tokenizer)

    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)

    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

继承LlamaFactoryInstructionHandler的类都会复用 _filter 函数。根据 self.llama_factory_template 来获取模型的模板，随后通过 self._tokenize_prompt 函数将数据集中的关键内容进行拼接，并用于训练。若想详细了解 self._tokenize_prompt ，可查看具体代码。

- **AlpacaStyleInstructionHandler解析**

  AlpacaStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为Alpaca格式，格式如下：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **SharegptStyleInstructionHandler解析**

  SharegptStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为**Sharegpt**格式，格式如下：

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-116 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.13.7.3 训练的权重转换说明

以**llama2-13b**举例，使用训练作业运行**obs_pipeline.sh**脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

如果用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。如果用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-117 权重转换脚本中的环境变量

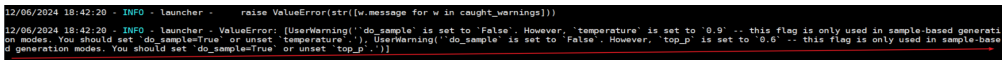
参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.13.7.4 训练 tokenizer 文件说明

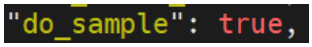
在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（ transformers==4.45.0 ），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。



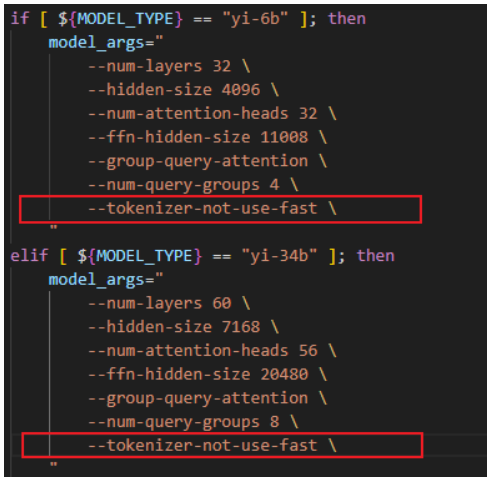
Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-227所示。

图 4-227 修改 Yi 模型 3_training.sh 文件



ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-228所示。

图 4-228 修改 ChatGLMv3-6B tokenizer 文件

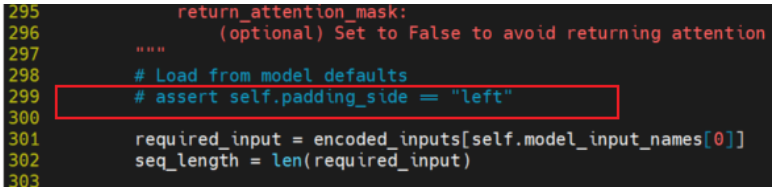


图 4-229 修改 ChatGLMv3-6B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference

        return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-230所示。

图 4-230 修改 ChatGLMv4-9B tokenizer 文件

```
        # Load from model defaults
        #assert self.padding_side == "left"
```

图 4-231 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-232所示。

图 4-232 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.13.8 常见错误原因和解决方法

4.13.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法：

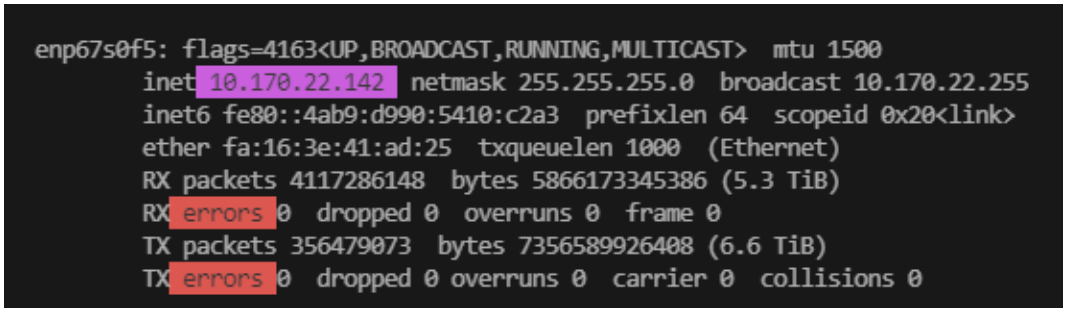
- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表4-115进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.13.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。




```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.14 主流开源大模型基于 Standard+OBS+SFS 适配 ModelLink PyTorch NPU 训练指导（6.3.911）

4.14.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.911版本相对于6.3.910版本新增如下内容：

- 文档中新增在数据预处理时，支持LLama-Factory格式的模板：
  - 支持Alpaca格式的数据，DATA_TYPE 环境变量需设置为AlpacaStyleInstructionHandler
  - 支持Sharegpt格式的数据，DATA_TYPE 环境变量需设置为SharegptStyleInstructionHandler
  - 已支持的系列模型模板：
    - qwen2.5系列
    - qwen2系列
    - qwen1.5系列
    - llama3.2系列
    - llama3.1系列
    - llama3系列
    - llama2系列
    - glm4-9b

- mixtral-8x7b
- baichuan2-13b

支持的模型列表

本方案支持以下模型的训练，如表4-118所示。

表 4-118 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-233 操作流程图

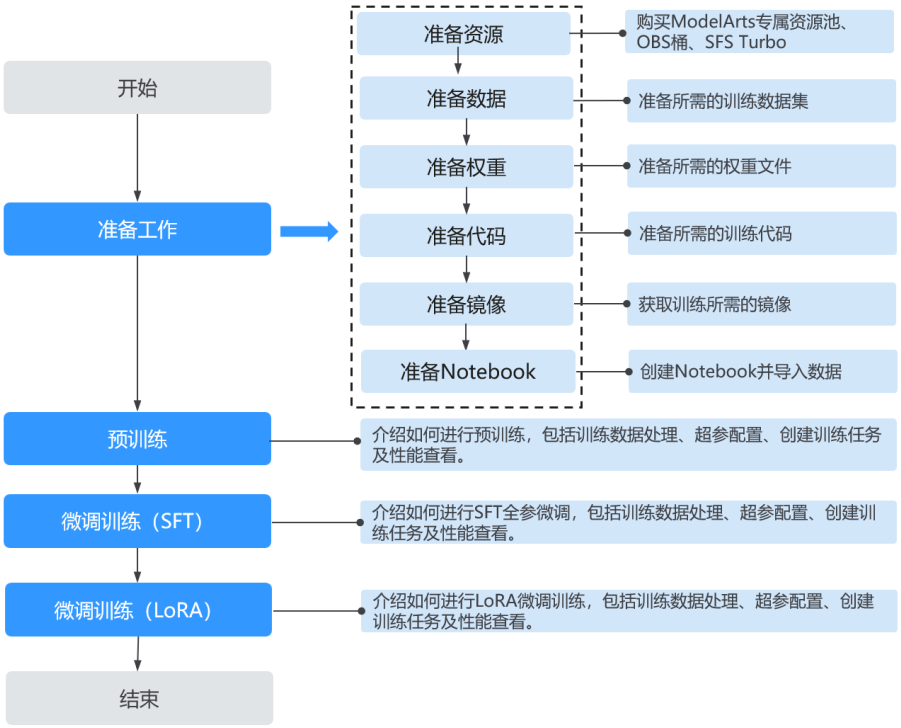


表 4-119 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。

阶段	任务	说明
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.14.2 准备工作

4.14.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-126**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。具体过程请参考**创建OBS桶**，例如桶名：standard-llama2-13b。

由于ModelArts创建训练作业时，需要将作业日志输出至OBS桶中，因此创建OBS桶为必选项。用户可通过**OBS Browser+**、**obsutil**等工具访问和管理OBS桶，将代码、模型文件、数据集等数据上传或下载进行备份。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

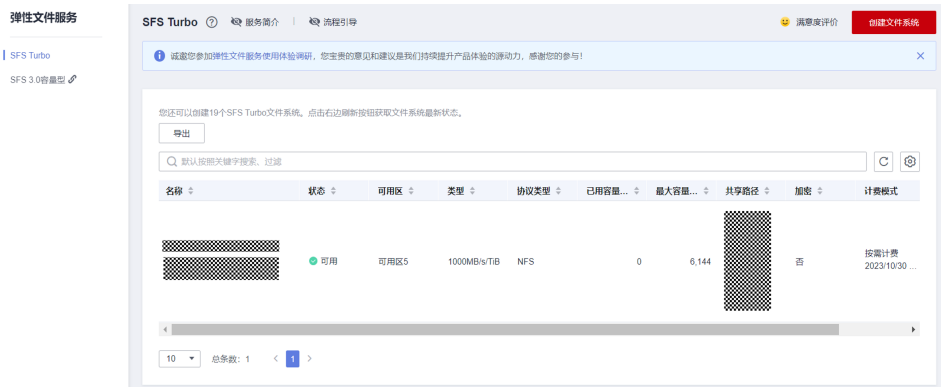
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-234 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

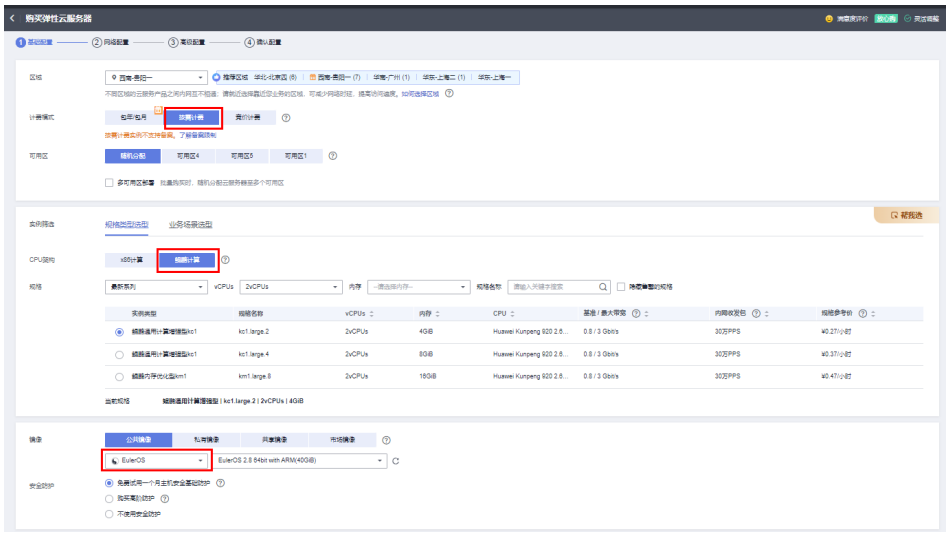
图 4-235 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 4-236 购买 ECS

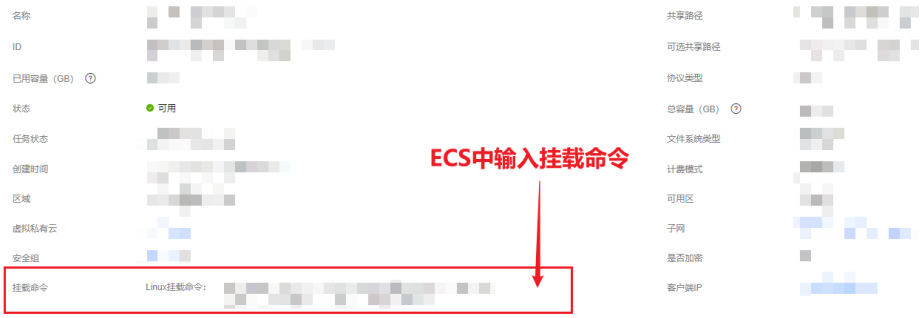


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看基本信息图4-237，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 4-237 SFS Turbo 基本信息



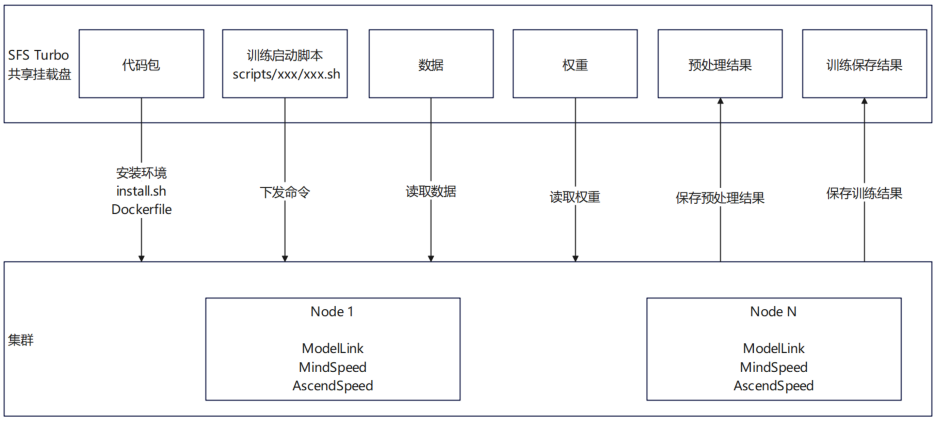
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-238 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表4-121**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.14.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

如果用户希望将MOSS数据集的Excel格式转换为.json格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据集至 SFS Turbo

准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。可通过两种方式，将数据集上传至SFS Turbo中。

方式一：将下载的原始数据通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/training_data目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具。数据存放参考目录：

```
/mnt/sfs_turbo/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
2. 利用**OBS Browser+工具**将下载的数据集上传至创建的文件夹目录下。得到OBS下数据集结构：


```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.14.2.3 准备权重

获取对应模型的权重文件，获取链接参考**表4-118**。权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。文件会直接下载用户本地，需要再上传至SFS Turbo中。
- **方法二：huggingface-cli**：**huggingface-cli**是Hugging Face官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

随后可通过以下两种方式，将下载到本地的模型文件上传至SFS Turbo中。

本地上传权重文件至 SFS Turbo

通过以下两种方式将下载到本地的模型文件上传至SFS Turbo中。方式一操作简单，但是数据传输速度比较慢，费时间。方式二操作相对方式一复杂一些，但是数据传输速度较快。

方式一：将已下载的模型文件通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/model目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放模型，例如在桶standard-llama2-13b中创建文件夹model/llama-2-13b-hf。
2. 利用**OBS Browser+工具**将下载的模型文件上传至创建的文件夹目录下。
3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.14.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-120**所示。

表 4-120 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.911代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
├──llm_train                # 模型训练代码包
        ├──AscendSpeed          # 基于AscendSpeed的训练代码
            ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            ├──scripts/             # 训练需要的启动脚本
                ├──llama2           # llama2系列模型执行脚本的文件夹
                ├──llama3           # llama3系列模型执行脚本的文件夹
                ├──qwen             # Qwen系列模型执行脚本的文件夹
                ├──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                ├── ...
                ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                ├── install.sh       # 环境部署脚本
            ├──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
├──llm_inference              # 推理代码包
├──llm_tools                  # 推理工具
```

代码上传至 SFS Turbo

将AscendSpeed代码包AscendCloud-LLM-xxx.zip直接上传至ECS服务器中的SFS Turbo中，例如存放在/mnt/sfs_turbo/AscendCloud-LLM-xxx.zip目录下并解压缩。

```
unzip AscendCloud-*.zip
```

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至SFS Turbo后，目录结构如下。

```
/mnt/sfs_turbo/
├──llm_train                  # 解压代码包后自动生成的代码目录，无需用户创建
    ├── AscendSpeed             # 代码目录
```

```
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          # 目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                          #微调数据文件
|── tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
|── models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.14.2.5 准备镜像

4.14.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-121 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**可二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。

- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

使用以上方案时，都会下载Megatron-LM、MindSpeed、ModelLink源码至AscendSpeed文件夹中。下载后的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/             # 训练需要的启动脚本
  |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/           # MindSpeed昇腾大模型加速库
  |——ModelLink/           # ModelLink端到端的大语言模型方案
      |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

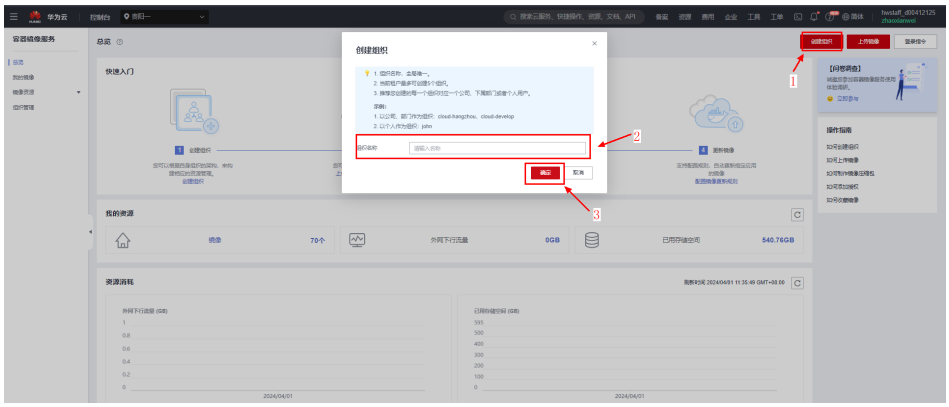
训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

4.14.2.5.2 ECS 获取和上传基础镜像

Step1 创建镜像组织

在SWR服务页面创建镜像组织。

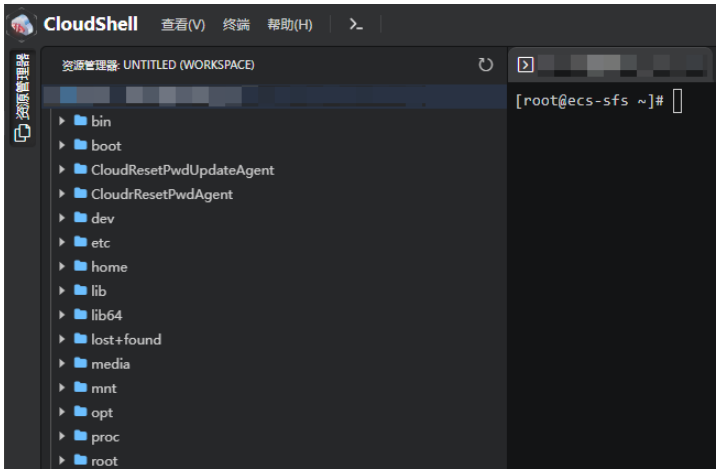
图 4-239 创建镜像组织



Step2 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录如图所示。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

图 4-240 CloudShell 远程登录界面



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

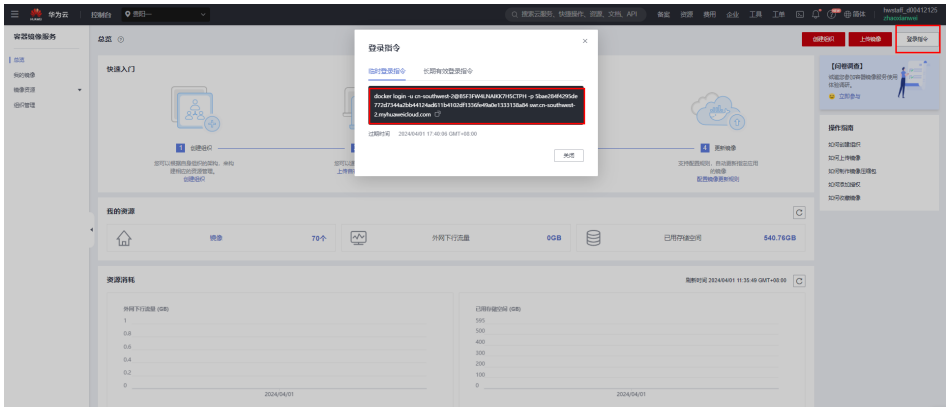
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-121**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-241 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.14.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-242**中都需要执行 install.sh文件，来安装依赖以及下载完整代码。

⚠ 注意

- 使用基础镜像的方法，需要确认训练作业的资源池是否联通公网，否则执行 install.sh 文件时下载代码会失败。因此可以选择配置网络或使用**ECS中构建新镜像**的方法。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 中的 transformers 的版本。

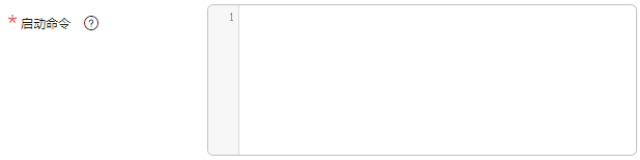
  由默认 transformers==4.45.0 修改为：transformers==4.44.2

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-242 训练作业启动命令



4.14.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-120**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   cd ./llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```

   注意

   若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

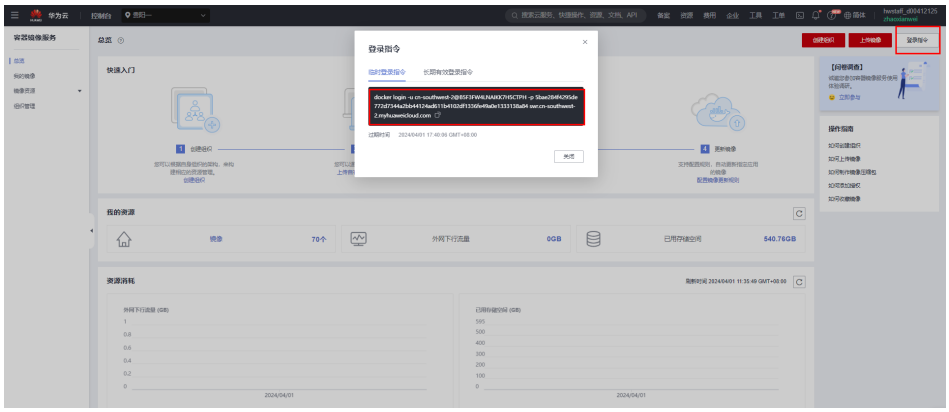
   由默认 transformers==4.45.0 修改为：transformers==4.44.2

4. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://
   xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-243 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.14.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-122**所示。其他超参均有默认值，可以参考**表4-125**按照实际需求修改。

表 4-122 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	**可添加**。该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。

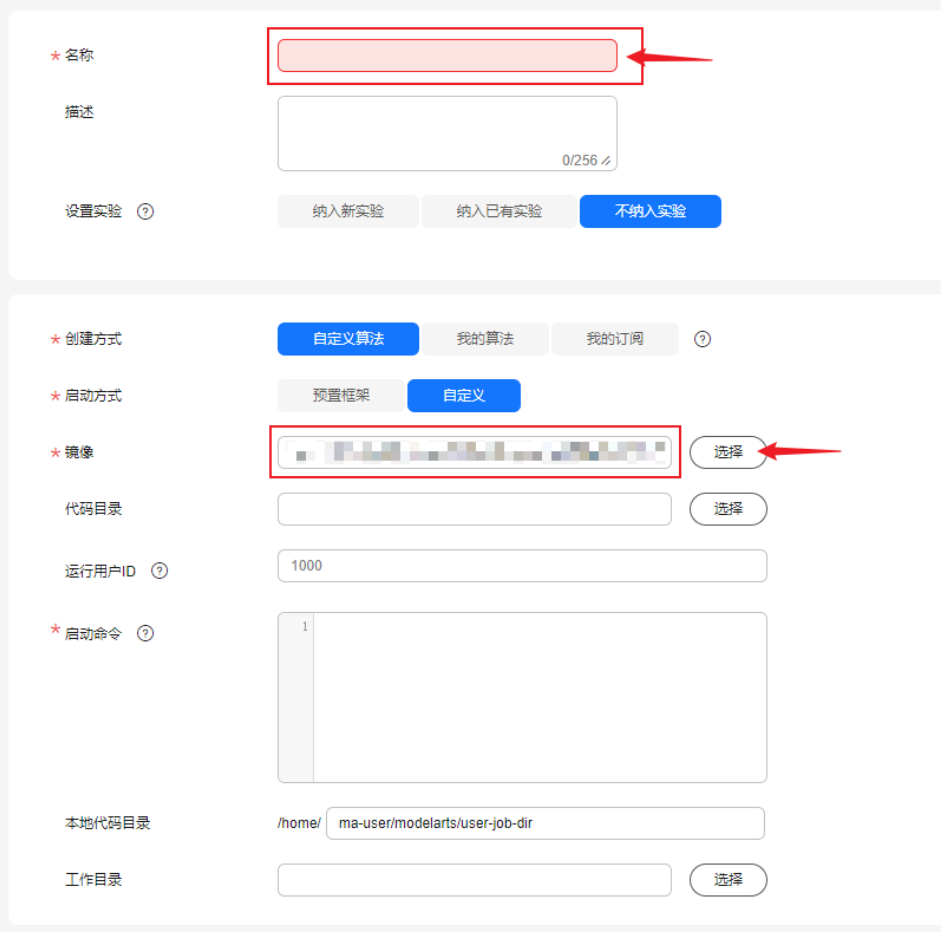
参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-244 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

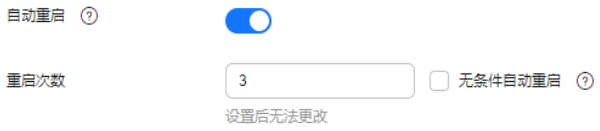
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-245 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

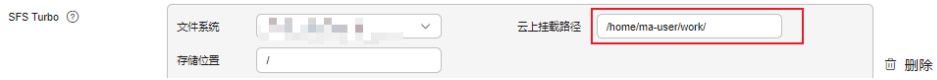
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-126**进行配置。

图 4-246 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-247 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.14.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-123**所示。其他超参均有默认值，可以参考**表4-125**按照实际需求修改。

表 4-123 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

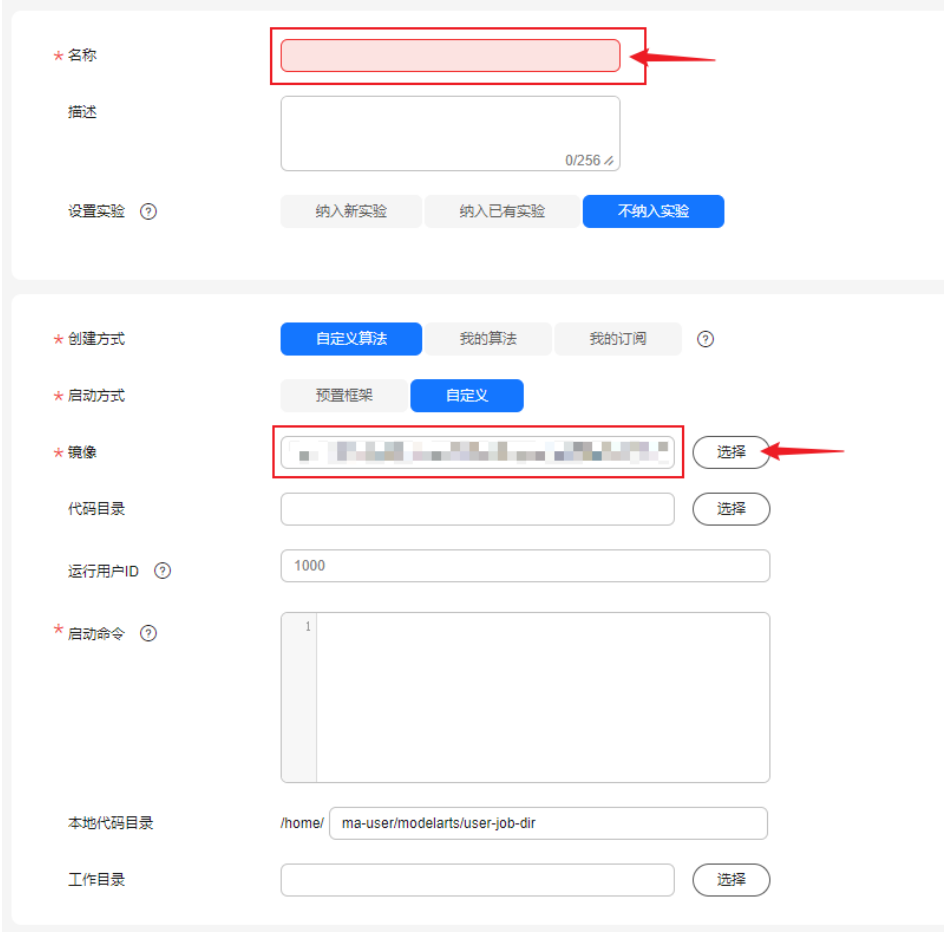
参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-248 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

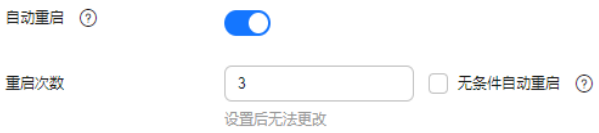
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-249 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

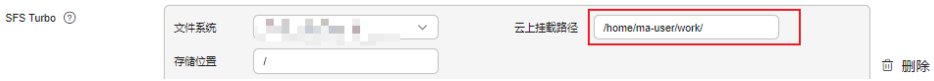
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-126**进行配置。

图 4-250 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-251 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.14.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-124**所示。其他超参均有默认值，可以参考**表4-125**按照实际需求修改。

表 4-124 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。

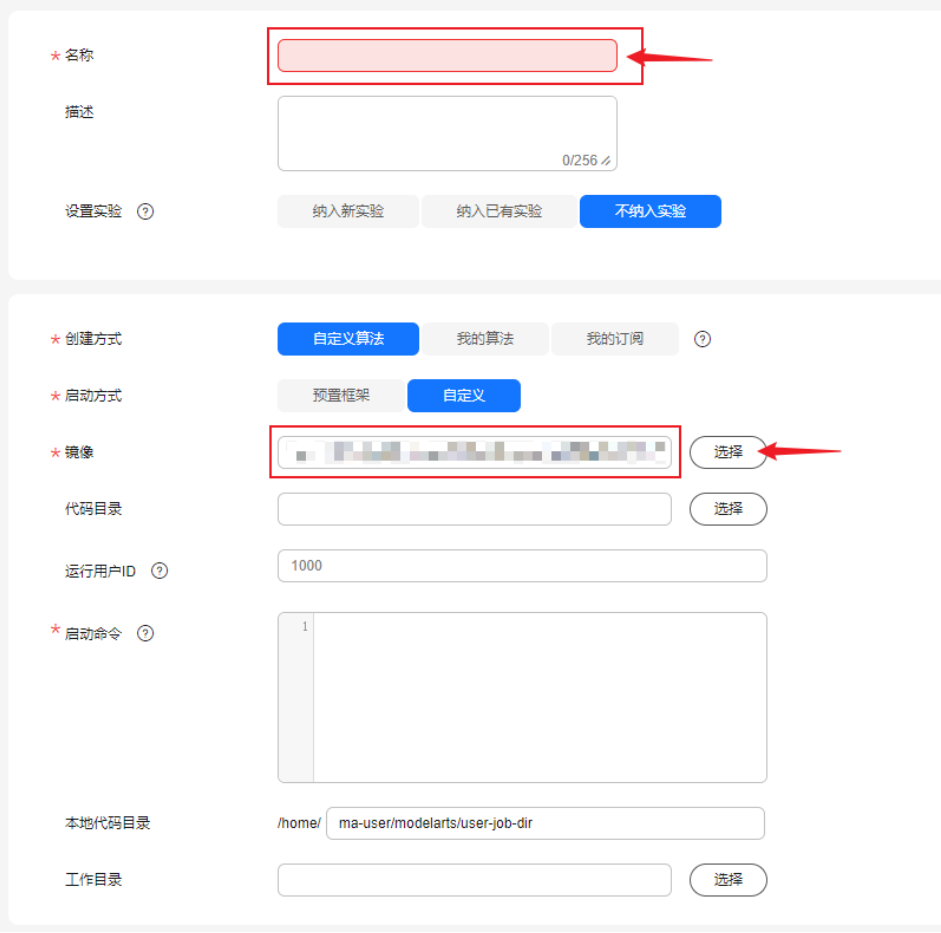
参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-252 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

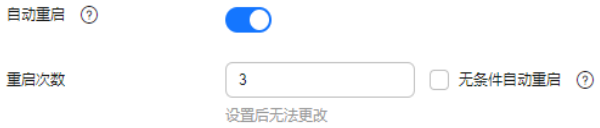
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-253 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

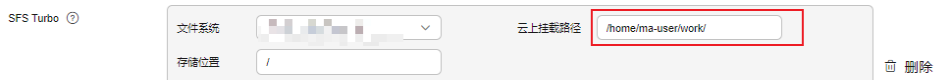
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-126**进行配置。

图 4-254 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-255 选择 SFS Turbo



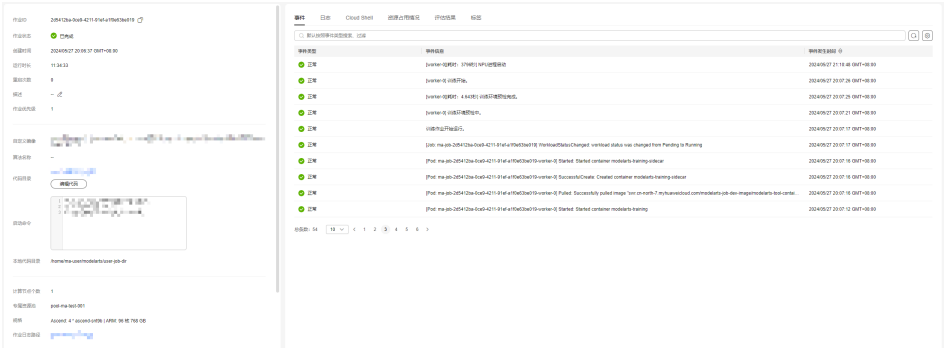
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.14.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

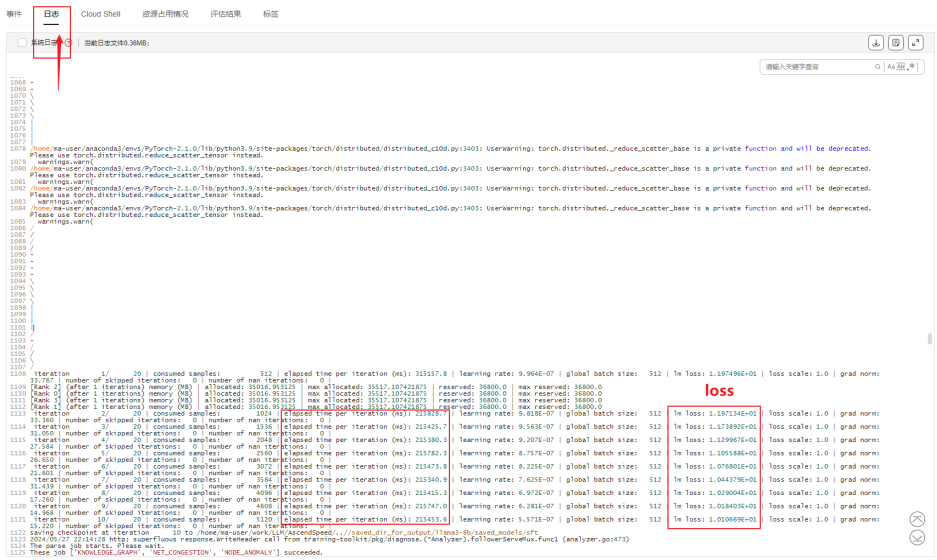
图 4-256 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-257 查看日志和性能



4.14.7 训练脚本说明

4.14.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-125 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-126所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-126 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.14.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**0_pl_pretrain_13b.sh**训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

ModelLink 预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

ModelLink 微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。支持 .parquet \ .csv \ .json \ .jsonl \ .txt \ .arrow 格式。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

LLama-Factory 微调数据集预处理参数说明

ModelLink开源仓已经支持LLama-Factory格式的数据预处理，目前仅支持sft全参微调，lora微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。可选项有['AlpacaStyleInstructionHandler ShareggptStyleInstructionHandler']。
  - AlpacaStyleInstructionHandler ：用于处理Alpaca风格的数据集。
  - ShareggptStyleInstructionHandler：用于处理sharegpt风格的数据集。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。
- --prompt-type：需要指定使用模型的template。已支持的系列模型可查看：**文档更新内容**。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample

- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。

  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) + self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

c. 在单轮对话中
   i. 对user和assistant的文本进行清洗
   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
   iii. input_ids是user_ids和assistant_ids的拼接
   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
d. attention_mask是和input_ids等长的全1序列
e. 返回input_ids\attention_mask\labels的字典
f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}
```

```
"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **LlamaFactoryInstructionHandler解析**

  LlamaFactoryInstructionHandler是处理数据集的一个基类，继承自BaseDatasetHandler，实现对Llama-Factory格式数据集的处理。

  注意：仅支持微调场景，如：sft全参微调，lora微调。已支持的系列模型可查看：**文档更新内容**。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    tokenized_full_prompt = self._tokenize_prompt(messages, self.llama_factory_template, self.tokenizer.tokenizer)

    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)

    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  继承LlamaFactoryInstructionHandler的类都会复用 _filter 函数。根据 self.llama_factory_template 来获取模型的模板，随后通过 self._tokenize_prompt 函数将数据集中的关键内容进行拼接，并用于训练。若想详细了解 self._tokenize_prompt ，可查看具体代码。

- **AlpacaStyleInstructionHandler解析**

  AlpacaStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为Alpaca格式，格式如下：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **SharegptStyleInstructionHandler解析**

  SharegptStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为**Sharegpt**格式，格式如下：

```
[
  {
```

```
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以llama2为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-127 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.14.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-128 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径

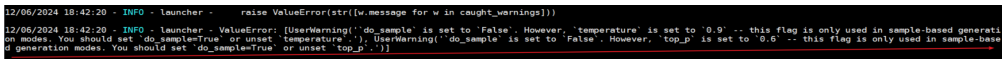
参数	示例	参数说明
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.14.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.45.0），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-258所示。

图 4-258 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-259 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-260 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-261 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-262 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-263 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.14.8 常见错误原因和解决方法

4.14.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表4-126进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.14.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.15 主流开源大模型基于 Lite Cluster 适配 ModelLink PyTorch NPU 训练指导（6.3.911）

4.15.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Cluster上的训练方案。训练框架使用的是ModelLink。

本方案目前仅适用于企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**表4-131**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Cluster。
- 本文档中的**CCE集群版本选择v1.27~1.28**。版本使用的容器引擎为**Containerd**。
- 镜像适配的Cann版本是cann_8.0.rc3，驱动版本是23.0.6。

- 确保集群可以访问公网。

文档更新内容

6.3.911版本相对于6.3.910版本新增如下内容：

- 文档中新增在数据预处理时，支持LLama-Factory格式的模板：
  - 支持Alpaca格式的数据，DATA_TYPE 环境变量需设置为AlpacaStyleInstructionHandler
  - 支持Sharegpt格式的数据，DATA_TYPE 环境变量需设置为SharegptStyleInstructionHandler
  - 已支持的系列模型模板：
    - qwen2.5系列
    - qwen2系列
    - qwen1.5系列
    - llama3.2系列
    - llama3.1系列
    - llama3系列
    - llama2系列
    - glm4-9b
    - mixtral-8x7b
    - baichuan2-13b

训练支持的模型列表

本方案支持以下模型的训练，如表4-129所示。

表 4-129 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)

序号	支持模型	支持模型参数量	权重文件获取地址
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**
24	llama3.1	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct**
25		llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct**
26	Qwen2.5	qwen2.5-0.5b	**https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct**
27		qwen2.5-7b	**https://huggingface.co/Qwen/Qwen2.5-7B-Instruct**
28		qwen2.5-14b	**https://huggingface.co/Qwen/Qwen2.5-14B-Instruct**
29		qwen2.5-32b	**https://huggingface.co/Qwen/Qwen2.5-32B-Instruct**
30		qwen2.5-72b	**https://huggingface.co/Qwen/Qwen2.5-72B-Instruct**
31	llama3.2	llama3.2-1b	**https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct**
32		llama3.2-3b	**https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct**

操作流程

图 4-264 操作流程图

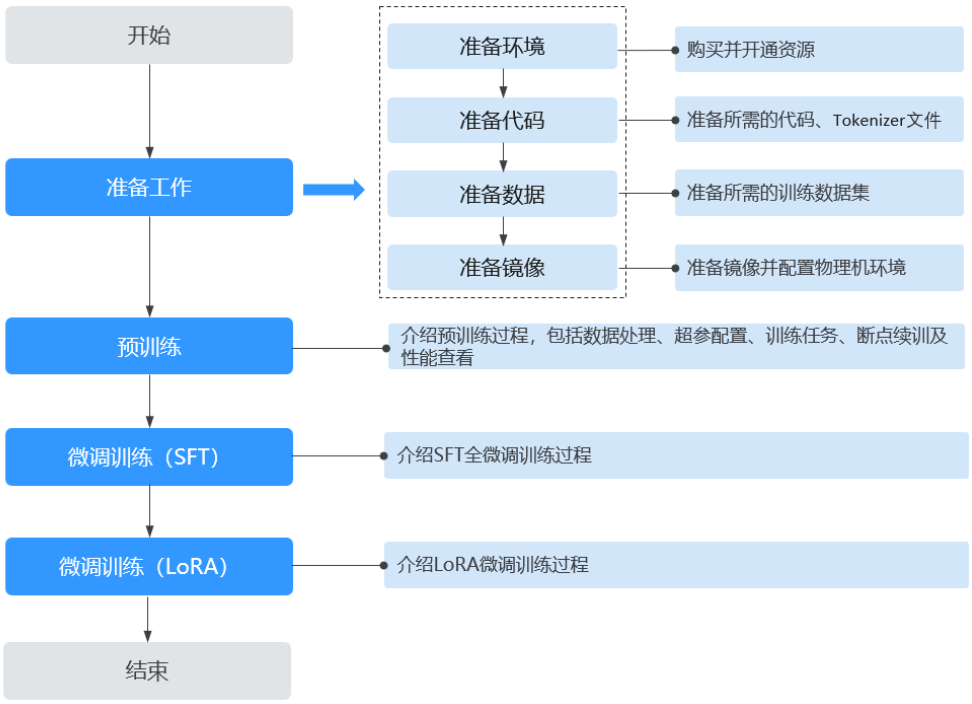


表 4-130 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite k8s Cluster运行的，需要购买并开通k8s Cluster资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.15.2 准备工作

4.15.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Cluster。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-138**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**k8s Cluster资源购买**，熟悉集群资源开通流程，再开始操作购买Cluster资源。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

k8s Cluster 资源配置

若已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

其中k8s Cluster的容器中挂载存储支持OBS、SFS Turbo等方案进行挂载。例如OBS支持静态挂载和动态挂载，而SFS Turbo仅支持静态挂载，详细的挂载操作流程可阅读**通过静态存储卷使用已有极速文件存储**和**通过动态存储卷使用对象存储**。

kubectl 访问集群配置

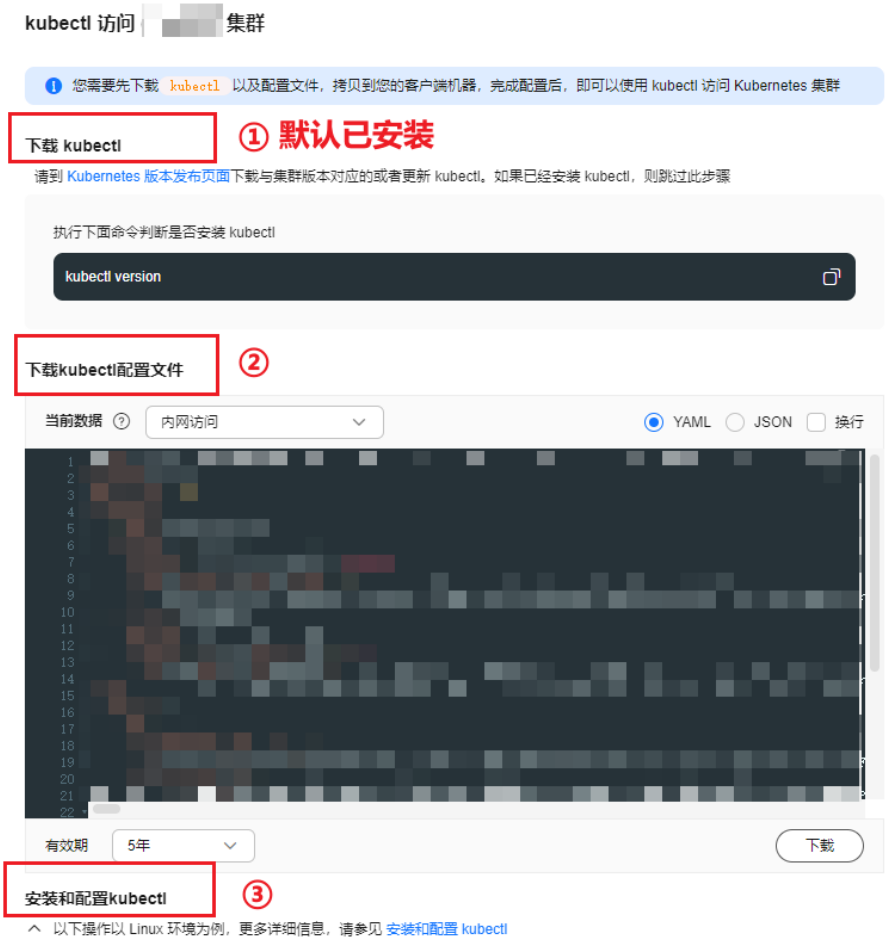
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的 CCE 集群控制版面中。根据**图4-265**的步骤进行操作，单击kubectl配置时，会弹出**图4-266**步骤页面。

图 4-265 配置中心

2. 根据**图4-266**，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-266 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。若显示如图图4-267的内容，则配置成功。

```
kubectl cluster-info
```

图 4-267 查看 Kubernetes 集群信息正确弹出内容

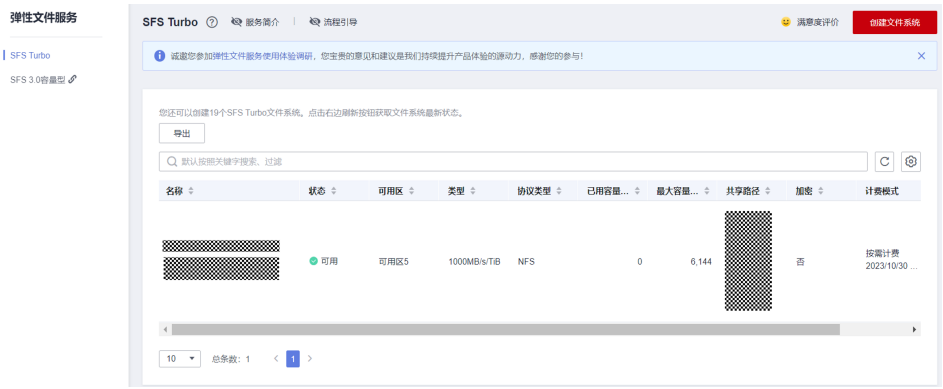
创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。

图 4-268 创建 SFS Turbo



2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

   详细操作指导请参考**创建SFS Turbo文件系统**。

   其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-269 SFS 类型和容量选择

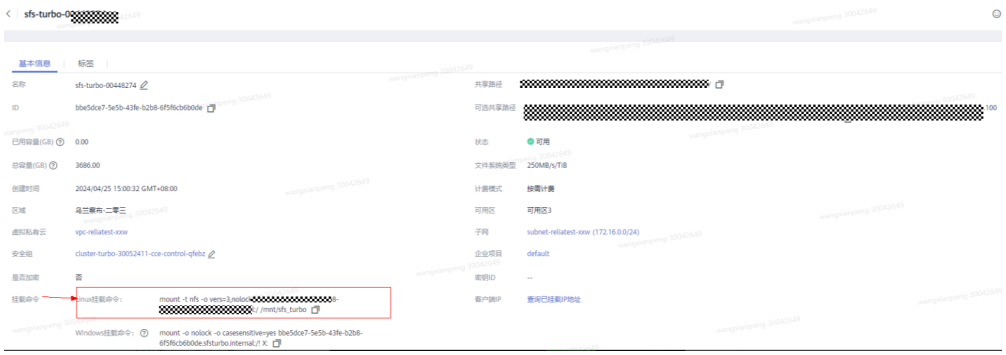
CCE 集群关联 SFS Turbo

进入已购买创建的CCE集群，选择存储，随后单击“创建存储卷声明PVC”。

- 选择“极速文件存储”，随后输入PVC名称。
- 选择“新建存储卷PV”，并单击“选择极速文件存储”。
- 进入选择页面，选择已经创建好的SFS Turbo，最后输入PV名称。

接下来需要通过访问集群节点，挂载SFS Turbo。

- 可通过ssh登录CCE集群中的某个节点（ssh使用的是eip地址）。
- 创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo
- SFS Turbo存储手动挂载到安装节点中，挂载命令如下截图：
- 挂载完成后，可通过以下步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。



4.15.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-131**所示，模型列表、对应的开源权重获取地址如**表4-129**所示。

表 4-131 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.911-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

获取对应模型的权重文件，获取链接参考**表4-129**。

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  若要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.911中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
      |——AscendSpeed           # 基于AscendSpeed的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——scripts/             # 训练需要的启动脚本
            |——llama2            # llama2系列模型执行脚本的文件夹
            |——llama3            # llama3系列模型执行脚本的文件夹
            |——qwen              # Qwen系列模型执行脚本的文件夹
            |——qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
            |—— ...
            |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
            |—— install.sh       # 环境部署脚本
         |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
```

```
|──llm_inference                  # 推理代码包
|──llm_tools                     # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}（例如使用SFS Turbo的路径：/mnt/sfs_turbo/）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data               # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizers                   #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                       #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录服务器。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如SFS Turbo的路径：/mnt/sfs_turbo目录下，以下都以/mnt/sfs_turbo为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/mnt/sfs_turbo/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/mnt/sfs_turbo，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /mnt/sfs_turbo
   mkdir -p tokenizers/Llama2-70B
   ```

4.15.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注他们自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

若用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，若同时输入，则优先使用test_count，若都未输入，则返回处理失败 False。

- **LLama-Factory Alpaca 指令微调数据**：数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - history： 一个列表，包含之前轮次的对话记录，每一对都是用户消息和模型回复。这有助于保持对话的一致性和连贯性。

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **LLama-Factory ShareGPT 指令微调数据**：ShareGPT 格式来源于通过记录 ChatGPT 与用户对话的数据集，主要用于对话系统的训练。它更侧重于多轮对话数据的收集和组织，模拟用户与 AI 之间的交互。数据集包含有以下字段：
  - conversations：包含一系列对话对象，每个对象都由发言者(from)和发言内容(value)组成。
  - from：表示对话的角色，可以是"human"（人类）或"gpt"（机器），表示是谁说的这句话。
  - value：具体的对话内容。
  - system：系统提示词，用来为整个对话设定场景或提供指导原则。
  - tools：描述可用的外部工具或功能的信息，这些工具可能被模型用来执行某些任务或获取更多信息。

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

上传数据到指定目录

将下载的原始数据存放在/mnt/sfs_turbo/training_data目录下。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

```
mkdir training_data
```

数据存放参考目录结构如下：

```
${workdir}
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.15.2.4 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-132 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b

表 4-133 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v  # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择containerd作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

   buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

   buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

   a. 下载并解压buildkit程序。
      ```
      # 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
      wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

      # 创建解压的目录
      ```

```
mkdir /usr/local/buildkit

# 解压到指定的目录
tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit

# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

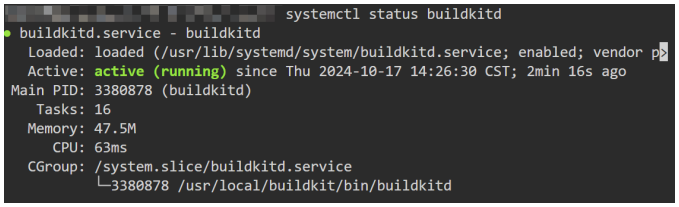
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

containerd 容器引擎有命名空间的概念。Kubernetes 下使用的 containerd 默认命名空间是 k8s.io。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。

  ```
  ctr -n k8s.io pull {image_url}
  ```

- 使用nerdctl工具拉取镜像。

  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

⚠ 注意

集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

步骤三 构建 ModelArts Lite 训练镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-131**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.908-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面

   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```

2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。

   ```
   FROM {image_url}
   ```

3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。

   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```

   注意

   若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

   由默认 transformers==4.45.0 修改为：transformers==4.44.2

4. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。

   ```
   nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称>  .
   ```

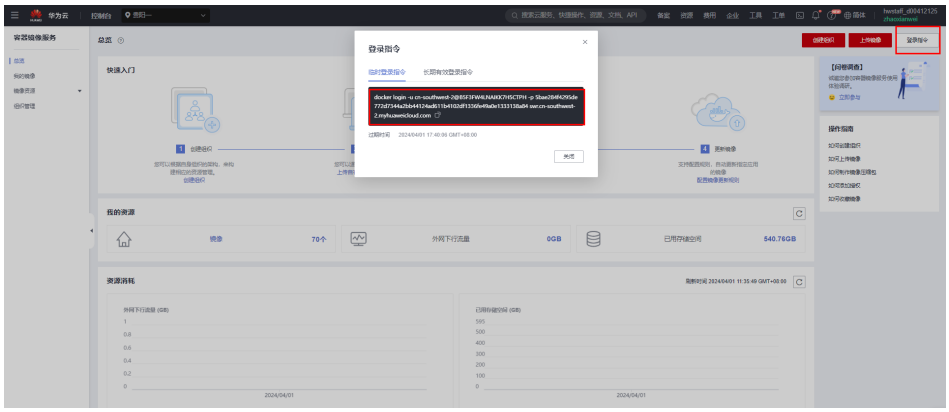
   nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以 nerdctl pull 拉取测试以下镜像是否能拉取成功。

   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤四 在节点机器中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。

图 4-270 复制登录指令



由于使用的容器引擎是containerd，不再是docker，因此需要改写复制的登录指令，将docker进行替换，使用nerdctl工具。

```
# docker login 替换为：
nerdctl login
```

步骤五 修改并上传镜像

1. 在机器中输入Step4登录指令后，使用下列示例命令将镜像上传至SWR：

```
nerdctl --namespace k8s.io tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在**步骤三 构建ModelArts Lite训练镜像**中使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

nerdctl --namespace k8s.io tag ${dockerfile_image_name} swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
nerdctl --namespace k8s.io push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

nerdctl --namespace k8s.io push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

步骤六 编写 Config.yaml 文件

k8s有两种方式来管理对象：

- 命令式，即通过Kubectl指令直接操作对象。
- 声明式，通过定义资源YAML格式的文件来操作对象。

首先给出单个节点训练的config.yaml文件模板，用于配置pod。而在训练中，需要按照参数说明修改**${}**中的参数值。该模板使用SFS Turbo挂载方案。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob              # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                     # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980     # 保持不动
data:                                    # data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                            # job名字，需要和configmap中名字保持联系
  namespace: default                     # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano                 # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}           # 镜像地址
          imagePullPolicy: IfNotPresent  # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                       # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
          env:
          - name: name
            valueFrom:
```

```
        fieldRef:
          fieldPath: metadata.name
    - name: ip
      valueFrom:
        fieldRef:
          fieldPath: status.hostIP
    - name: framework
      value: "PyTorch"
    command: ["/bin/sh", "-c"]
    args:
    - ${command}
    resources:
      requests:
        huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
        memory: ${requests_memory}                 # 容器请求的最小内存
        cpu: ${requests_cpu}                     # 容器请求的最小 CPU
      limits:
        huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
        memory: ${limits_memory}                  # 容器可使用的最大内存
        cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
    volumeMounts:                             # 容器内部映射路径
    - name: shared-memory-volume
      mountPath: /dev/shm
    - name: ascend-driver                     # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons                    # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                              # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                           # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                    # 物理机外部路径
  - name: shared-memory-volume                    # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
```

```
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
```

双个节点训练的config.yaml文件模板，用于实现双机分布式训练。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob     # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                  # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
data:                    #data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                         # job名字，需要和configmap中名字保持联系
  namespace: default                  # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano               # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}         # 镜像地址
          imagePullPolicy: IfNotPresent      # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：
每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                       # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
```

```
    env:
    - name: name
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: ip
      valueFrom:
        fieldRef:
          fieldPath: status.hostIP
    - name: framework
      value: "PyTorch"
    command: ["/bin/sh", "-c"]
    args:
    - ${command}
    resources:
      requests:
        huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
        memory: ${requests_memory}                # 容器请求的最小内存
        cpu: ${requests_cpu}                      # 容器请求的最小 CPU
      limits:
        huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
        memory: ${limits_memory}                  # 容器可使用的最大内存
        cpu: ${limits_cpu}                        # 容器可使用的最大 CPU
    volumeMounts:                                 # 容器内部映射路径
    - name: shared-memory-volume
      mountPath: /dev/shm
    - name: ascend-driver                         # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons                        # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                                  # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                               # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                        # 物理机外部路径
  - name: shared-memory-volume                    # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
```

```
        path: /usr/slog
    - name: sfs-volume
      persistentVolumeClaim:
        claimName: ${pvc_name}    #已创建的PVC名称
    restartPolicy: OnFailure
- name: work
  replicas: 1
  template:
    metadata:
      name: training
      labels:
        app: ascendspeed
        ring-controller.cce: ascend-1980  # 保持不动
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: volcano.sh/job-name
                    operator: In
                    values:
                      - vcjob
              topologyKey: kubernetes.io/hostname
      hostNetwork: true                    # 采用宿主机网络模式
      containers:
      - image: ${image_name}         # 镜像地址
        imagePullPolicy: IfNotPresent       # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
        name: ${container_name}
        securityContext:                        # 容器内 root 权限
          allowPrivilegeEscalation: false
          runAsUser: 0
        env:
        - name: name
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ip
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: framework
          value: "PyTorch"
        command: ["/bin/sh", "-c"]
        args:
        - ${command}
        resources:
          requests:
            huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
            memory: ${requests_memory}                # 容器请求的最小内存
            cpu: ${requests_cpu}                    # 容器请求的最小 CPU
          limits:
            huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
            memory: ${limits_memory}                 # 容器可使用的最大内存
            cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
        volumeMounts:                           # 容器内部映射路径
        - name: shared-memory-volume
          mountPath: /dev/shm
        - name: ascend-driver                   # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons                  # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: localtime
          mountPath: /etc/localtime
        - name: hccn                            # 驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: npu-smi                         # npu-smi
          mountPath: /usr/local/sbin/npu-smi
```

```
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                # 物理机外部路径
  - name: shared-memory-volume                    # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
      claimName: ${pvc_name}    #已创建的PVC名称
  restartPolicy: OnFailure
```

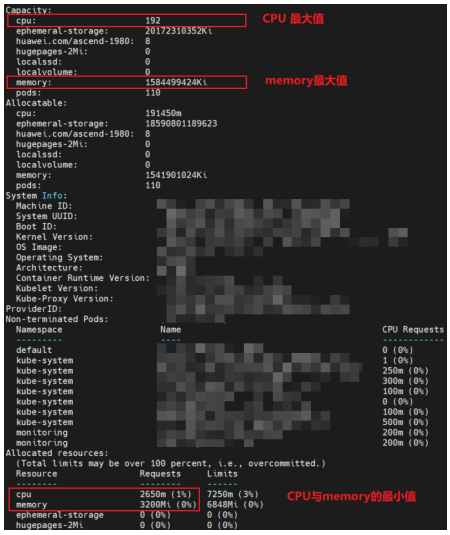
参数说明：

- ${container_name} 容器名称，此处可以自己定义一个容器名称，例如ascendspeed。
- ${image_name} 为**步骤五 修改并上传镜像**中，上传至SWR上的镜像链接。
- ${command} 使用config.yaml文件创建pod后，在容器内自动运行的命令。在进行训练任务中会给出替换命令。
- /mnt/sfs_turbo 为宿主机中默认挂载SFS Turbo的工作目录，目录下存放着训练所需代码、数据等文件。
  - 同样，/mnt/sfs_turbo 也可以映射至容器中，作为容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。为方便访问两个地址可以相同。
- ${pvc_name} 为在**CCE集群关联SFS Turbo**步骤中创建的PVC名称。
- 在设置容器中需要的CPU与内存大小时，可通过运行以下命令查看申请的节点机器中具体的CPU与内存信息。

  ```
  kubectl describe node
  ```

  - ${requests_cpu} 指在容器中请求的最小CPU核心数量，可使用Requests中的值，例如2650m。
  - ${requests_memory} 指在容器中请求的最小内存空间大小，可使用Requests中的值，例如3200Mi。
  - ${limits_cpu} 指在容器中可使用的最大CPU核心数量，例如192。

- ${limits_memory} 指在容器中可使用的最大内存空间大小，例如换算成1500Gi。



4.15.3 预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-134**所示。其他超参均有默认值，可以参考**表4-137**按照实际需求修改。

表 4-134 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-suser/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 修改 config.yaml 中的${command}

请根据**步骤二 修改训练超参配置**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
  - name: work1
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
  - name: work2
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
  - name: work3
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g               2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk               2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                        1/1     Running   0             22m
default       vcjob-work-0                        1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                  1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                  1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw            1/1     Running   0             37h
```

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-271 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:     32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.15.4 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-135**所示。其他超参均有默认值，可以参考**表4-137**按照实际需求修改。

表 4-135 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**表4-135**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
```

```
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0; # 单机训练执行命令
```

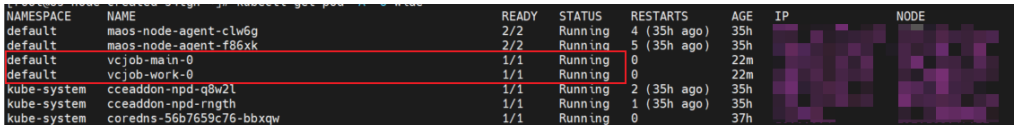
步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```



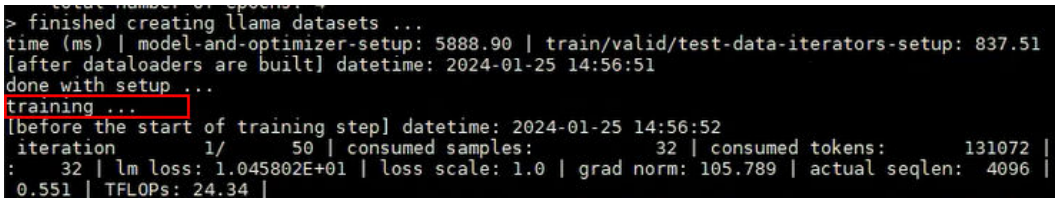
若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-272 等待模型载入



训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看sft微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.15.5 LoRA 微调训练

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-136**所示。其他超参均有默认值，可以参考**表4-137**按照实际需求修改。

表 4-136 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。

参数	示例值	参数说明
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**表4-136**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                  READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g                 2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk                 2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                          1/1     Running   0             22m
default       vcjob-work-0                          1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                    1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                    1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw              1/1     Running   0             37h
```

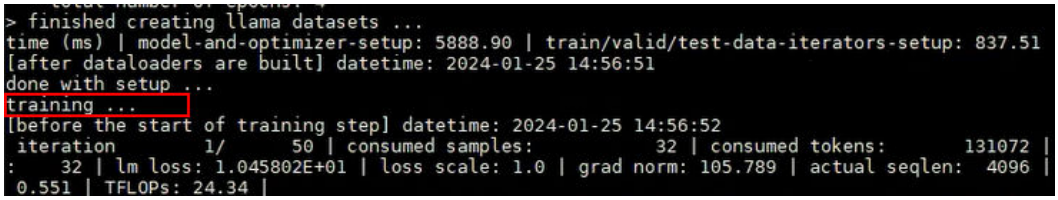
若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-273 等待模型载入



训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看lora微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.15.6 查看日志和性能

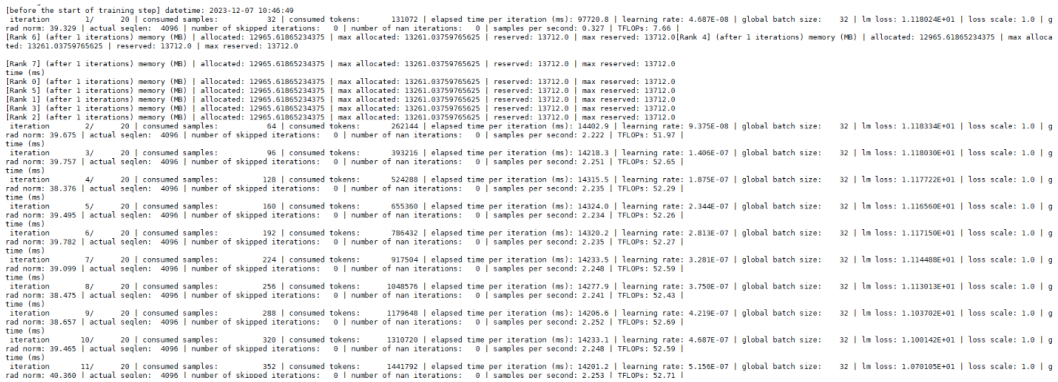
查看日志

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

训练过程中，训练日志会在最后的Rank节点打印。

图 4-274 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

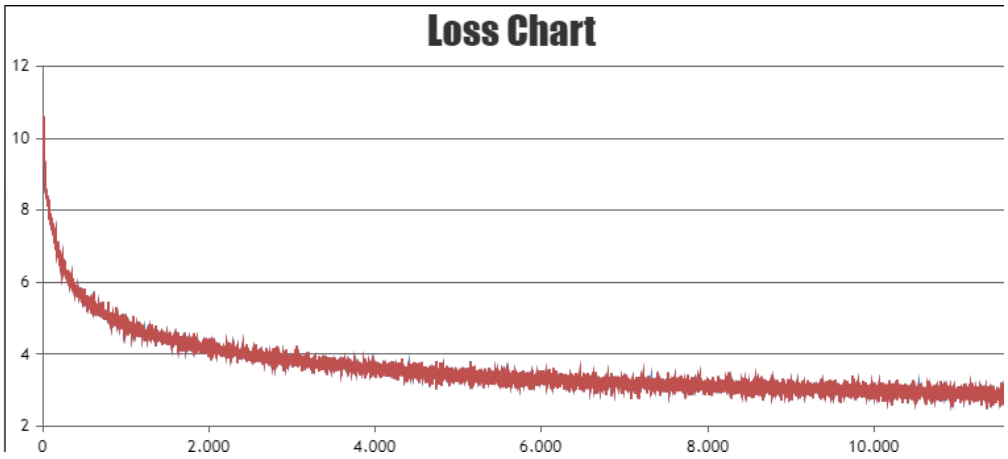
- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-137**。

- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具TrainingLogParser查看loss收敛情况，如图4-275所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 4-275 Loss 收敛情况（示意图）

4.15.7 训练脚本说明

4.15.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 4-137 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。

参数	示例值	参数说明
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler, AlpacaStyleInstructionHandler, SharegptStyleInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。 ● AlpacaStyleInstructionHandler：使用LLama-Factory模板Alpaca数据集 ● SharegptStyleInstructionHandler：使用LLama-Factory模板Sharegpt数据集
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。若训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。

参数	示例值	参数说明
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	1000	用于模型中间版本地保存。 ● 当参数值>=TRAIN_ITERS时，生成模型仅保存经过TRAIN_ITERS次训练后的最后一个版本。 ● 当参数值<TRAIN_ITERS时，生成模型会每经过SAVE_INTERVAL次，保存一次模型版本。 模型版本保存次数=TRAIN_ITERS//SAVE_INTERVAL+1
SAVE_TOTAL_LIMIT	0	用于控制权重版本保存次数。 ● 当参数不设置或<=0时，不会触发效果。 ● 参数值需<=TRAIN_ITERS//SAVE_INTERVAL+1 ● 当参数值>1时，保存模型版本次数与SAVE_TOTAL_LIMIT的值一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-138**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-138 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.15.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

ModelLink 预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

ModelLink 微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

LLama-Factory 微调数据集预处理参数说明

ModelLink开源仓已经支持LLama-Factory格式的数据预处理，目前仅支持sft全参微调，lora微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。可选项有['AlpacaStyleInstructionHandler ShareggptStyleInstructionHandler']。
  - AlpacaStyleInstructionHandler ：用于处理Alpaca风格的数据集。
  - ShareggptStyleInstructionHandler：用于处理sharegpt风格的数据集。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。
- --prompt-type：需要指定使用模型的template。已支持的系列模型可查看：**文档更新内容**。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample

- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。

- input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
- output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

– 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

– 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) + self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

c. 在单轮对话中
   i. 对user和assistant的文本进行清洗
   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
   iii. input_ids是user_ids和assistant_ids的拼接
   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
d. attention_mask是和input_ids等长的全1序列
e. 返回input_ids\attention_mask\labels的字典
f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}
```

```
"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **LlamaFactoryInstructionHandler解析**

  LlamaFactoryInstructionHandler是处理数据集的一个基类，继承自BaseDatasetHandler，实现对Llama-Factory格式数据集的处理。

  注意：仅支持微调场景，如：sft全参微调，lora微调。已支持的系列模型可查看：**文档更新内容**。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    tokenized_full_prompt = self._tokenize_prompt(messages, self.llama_factory_template, self.tokenizer.tokenizer)

    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)

    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  继承LlamaFactoryInstructionHandler的类都会复用 _filter 函数。根据 self.llama_factory_template 来获取模型的模板，随后通过 self._tokenize_prompt 函数将数据集中的关键内容进行拼接，并用于训练。若想详细了解 self._tokenize_prompt ，可查看具体代码。

- **AlpacaStyleInstructionHandler解析**

  AlpacaStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为Alpaca格式，格式如下：

```
[
  {
    "instruction": "人类指令（必填）",
    "input": "人类输入（选填）",
    "output": "模型回答（必填）",
    "system": "系统提示词（选填）",
    "history": [
      ["第一轮指令（选填）", "第一轮回答（选填）"],
      ["第二轮指令（选填）", "第二轮回答（选填）"]
    ]
  }
]
```

- **SharegptStyleInstructionHandler解析**

  SharegptStyleInstructionHandler是处理Llama-Factory格式数据集的一个类，继承LlamaFactoryInstructionHandler。要求的训练数据格式为**Sharegpt**格式，格式如下：

```
[
  {
```

```
    "conversations": [
      {
        "from": "human",
        "value": "人类指令"
      },
      {
        "from": "function_call",
        "value": "工具参数"
      },
      {
        "from": "observation",
        "value": "工具结果"
      },
      {
        "from": "gpt",
        "value": "模型回答"
      }
      ],
    "system": "系统提示词（选填）",
    "tools": "工具描述（选填）"
  }
]
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/1_preprocess_data.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-139 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀

环境变量	示例	参数说明
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.15.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。

- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-140 权重转换脚本中的环境变量

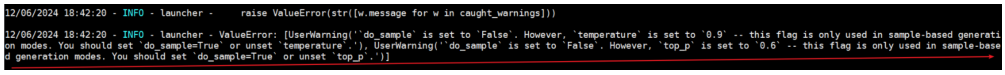
参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.15.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

LLama2 模型

在当前的软件版本中，由于transformers的版本过高（transformers==4.45.0），导致llama2系列模型与transformers不兼容导致报错，报错如图所示。



解决：在训练开始前，针对llama2模型中的tokenizer，需要修在generation_config.json中加入"do_sample": true，具体如图所示。

```
"do_sample": true,
```

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-276所示。

图 4-276 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-277所示。

图 4-277 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-278 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-279所示。

图 4-279 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-280 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下代码信息进行查找，修改后如图4-281所示。

图 4-281 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.15.8 常见错误原因和解决方法

4.15.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表4-138进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.15.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

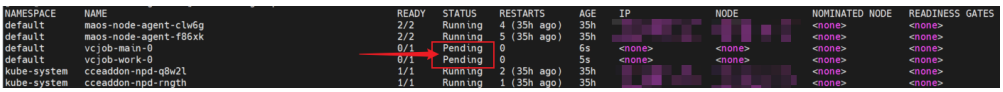
```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.15.8.3 工作负载 Pod 异常

Pod 状态为 Pending

当Pod状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。具体参考链接为**工作负载状态异常定位方法**。

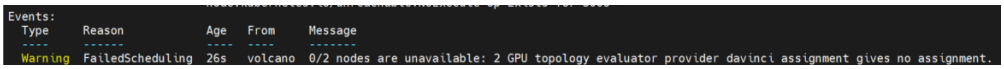


通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

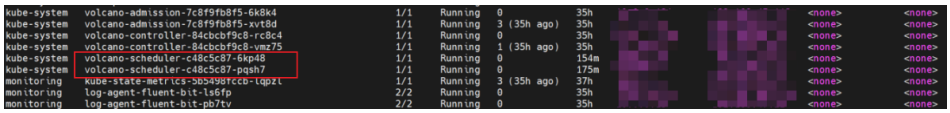
volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。



解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```
3. 若重启后，还是会Pending，建议多重复重启几次。



其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

如何删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.16 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.910）

4.16.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc3。
- Server驱动版本要求23.0.6。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-141 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表4-142所示。

表 4-142 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如表4-143所示。

表 4-143 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
57	internvl2-8B	√	x	x	x	x	OpenGVLab/InternVL2-8B at main (huggingface.co)
58	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)
59	internvl2-40B	√	x	x	x	x	OpenGVLab/InternVL2-40B at main (huggingface.co)
60	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
61	deepseek-v2-236b	x	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2**
62	deepseek-v2-lite-16b	√	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite**
63	qwen2-vl-7B	√	x	x	x	x	**Qwen/Qwen2-VL-7B-Instruct at main (huggingface.co)** 注意：Qwen2-VL 开源vllm依赖特定transformers版本， 请手动安装： pip install git+https://github.com/huggingface/transformers.git@21fac7abba2a37fae86106f87fcf9974fd1e3830
64	qwen-vl	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL**
65	qwen-vl-chat	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL-Chat**

66	MiniCPM-v2	√	x	x	x	x	**https://huggingface.co/HwwwH/MiniCPM-V-2**
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
       ├── build.sh               # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile              # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
   ├──AutoSmoothQuant  # W8A8量化工具
       ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
       ├── autosmoothquant_ascend          # 量化代码
       ├── build.sh                        # 安装量化模块的脚本
   ├──AutoAWQ             # W4A16量化工具
       ├──convert_awq_to_npu.py   # awq权重转换脚本
       ├──quantize.py   # 昇腾适配的量化转换脚本
       ├──build.sh      # 安装量化模块的脚本
   ├──llm_evaluation    # 推理评测代码包
     ├──benchmark_tools    #性能评测
       ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
       ├── benchmark_parallel.py   # 评测静态性能脚本
       ├── benchmark_serving.py    # 评测动态性能脚本
       ├── benchmark_utils.py      # 抽离的工具集
       ├── generate_datasets.py    # 生成自定义数据集的脚本
       ├── requirements.txt        # 第三方依赖
     ├──benchmark_eval #精度评测
       ├──opencompass.sh       #运行opencompass脚本
       ├──install.sh           #安装opencompass脚本
       ├──vllm_api.py          #启动vllm api服务器
       ├──vllm.py             #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型基于Lite Server适配PyTorch NPU训练指导**。

4.16.2 部署推理服务

4.16.2.1 非分离部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

什么是非分离部署

全量推理和增量推理在同一节点上进行。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-141**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.910-xxx.zip和算子包AscendCloud-OPP-6.3.910-xxx.zip到主机中，包获取路径请参见**表4-142**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-143**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下。
```
df -h
```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.910-xxx.zip和算子包AscendCloud-OPP-6.3.910-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

步骤五 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

步骤六 启动推理服务

1. 进入容器。
```
docker exec -it -u ma-user ${container-name} /bin/bash
```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
```
npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，若希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-282 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 92.5        51                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+====================================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

4. 配置环境变量。
```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
```

```
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

5. 若要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

   通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

   在MoE模型和小模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8；和Qwen2-1.5B、Qwen2-0.5B。当前MoE模型图模式启动不支持multi step。

   MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

   开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

6. 若要使用eagle投机，配置环境变量，使eagle投机对齐论文版本实现。目前默认开启此模式，若不开启，目前vllm0.6.0版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现，默认开启
export ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
```

   如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

   具体可参考6 eagle投机小模型训练 **步骤五：训练生成权重转换成可以支持vLLM推理的格式**

7. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理

8. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务【**推荐，在vllm-0.6.0之后的版本性能更好**】**

  在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

  （1）非多模态

  ```
  python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --host=${docker_ip} \
  --port=8080 \
  --gpu-memory-utilization=0.9 \
  --num-scheduler-steps=8 \
  --trust-remote-code
  ```

  （2）多模态

  当前支持Llava、InternVL2、MiniCPM、qwen2-vl模型，具体模型和权重地址参见**表4-143**，推荐显卡数量参见**表4-149**。

  ```
  export VLLM_IMAGE_FETCH_TIMEOUT=100
  export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
  # PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
  # 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
  export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

  python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --chat-template ${chat_template_path} \
  --dtype ${dtype} \
  --host=${docker_ip} \
  --port=${port} \
  --gpu-memory-utilization=0.9 \
  --trust-remote-code
  ```

  多模态推理服务启动模板参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
  - PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
  - --chat-template：对话构建模板，可选参数。如：

    （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **方式三：多机部署vLLM服务API接口启动服务**（可选）

  当单机显存无法放下模型权重时，可选用该种方式部署；该种部署方式，需要机器在同一个集群，NPU卡之间IP能够ping通方可，具体步骤如下：

  i. 查看卡IP。
  ```
  for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
  ```
  ii. 检查卡之间的网络是否通。
  ```
  # 在另一个节点上执行，29.81.3.172是上一步输出的ipaddr的值
  hccn_tool -i 0 -ping -g address 29.81.3.172
  ```
  iii. 启动Ray集群。
  ```
  # 指定通信网卡，使用ifconfig查看，找到和主机IP一致的网卡名
  export GLOO_SOCKET_IFNAME=enp67s0f5
  export TP_SOCKET_IFNAME=enp67s0f5
  # 指定可使用的卡
  export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
  # 将其中一个节点设为头节点
  ray start --head --num-gpus=8
  # 在其他节点执行
  ray start --address='10.170.22.18:6379' --num-gpus=8
  ```
  - --num-gpus：要跟ASCEND_RT_VISIBLE_DEVICES指定的可用卡数一致。
  - --address：头节点IP+端口号，头节点创建成功后，会有打印。

  iv. 正常启服务即可。

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。若使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数，否则会导致投机推理启动报错。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

```
export USE_PREFIX_HIGH_PRECISION_MODE=1
```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
--max-loras=32 \
--max-cpu-loras=32
```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。

- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:
Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on
http://0.0.0.0:8080 (Press CTRL+C to quit)
```

步骤七 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-144**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
```

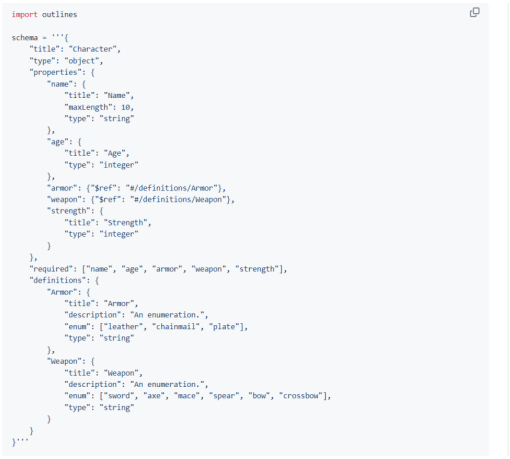
```
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-144 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-283** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-145 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-146 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

4.16.2.2 分离部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

什么是分离部署

大模型推理是自回归的过程，有以下两阶段：

- Prefill阶段（全量推理）

  将用户请求的prompt传入大模型，进行计算，中间结果写入KVCache并推出第1个token，属于计算密集型。

- Decode阶段（增量推理）

  将请求的前1个token传入大模型，从显存读取前文产生的KVCache再进行计算，属于访存密集型。

分离部署场景下，全量推理和增量推理在不同的容器上进行，用于提高资源利用效率。

分离部署的实例类型启动分为以下三个阶段：

1. **步骤六 启动全量推理实例**：必须为NPU实例，用于启动全量推理服务，负责输入的全量推理。全量推理占用至少1个容器。
2. **步骤七 启动增量推理实例**：必须为NPU实例，用于启动增量推理服务，负责输入的增量推理。增量推理占用至少1个容器。
3. **步骤八 启动scheduler实例**：可为CPU实例，用于启动api-server服务，负责接收推理请求，向全量或增量推理实例分发请求，收集推理结果并向客户端返回推理结果。服务调度实例不占用显卡资源，建议增加1个容器，也可以在全量推理或增量推理的容器上启动。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-141**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.910-xxx.zip和算子包AscendCloud-OPP-6.3.910-xxx.zip到主机中，包获取路径请参见**表4-142**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-143**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

   3.权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：

```
df -h
```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.910-xxx.zip和算子包AscendCloud-OPP-6.3.910-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

如果推理需要使用npu加速图片预处理，需要安装torchvision_npu，可放到镜像制作脚本里面。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

步骤五 生成 ranktable

介绍如何生成ranktable，以1p1d-tp2分离部署模式为例。当前1p1d分离部署模式，全量节点和增量节点分别占用2张卡，一共使用4张卡。

- 配置tools工具根目录环境变量

  使用AscendCloud-LLM发布版本进行推理，基于AscendCloud-LLM包的解压路径配置tool工具根目录环境变量：

  ```
  export LLM_TOOLS_PATH=${root_path_of_AscendCloud-LLM}/llm_tools
  ```

  其中，`${root_path_of_AscendCloud-LLM}`为AscendCloud-LLM包解压后的根路径。

  当使用昇腾云的官方指导文档制作推理镜像时，可直接基于该固定路径配置环境变量：

  ```
  export LLM_TOOLS_PATH=/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools
  ```

- 获取每台机器的rank_table

  在每个机器生成global rank_table信息与local rank_table信息。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 4,5 --decode-server-list 6,7 --api-server --save-dir ./save_dir
  ```

  执行后，会生成一个global_ranktable.json文件和使用实例个数的local_ranktable.json文件；如果指定了`--api-server`，还会生成一个local_ranktable_host.json文件用于确定服务入口实例。

  ./save_dir 生成ranktable文件如下（假设本地主机ip为10.**.**.18）。

  ```
  global_ranktable_10.**.**.18.json      # global rank_table
  local_ranktable_10.**.**.18_45.json    # 全量节点local rank_table
  ```

```
local_ranktable_10.**.**.18_67.json    # 增量节点local rank_table
local_ranktable_10.**.**.18_host.json  # api-server
```

- 合并不同机器的global rank_table(可选)

  如果分离部署在多台机器，获取每台机器的rank_table后，合并各个机器的global rank_table得到完整的global rank_table。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode merge --global-ranktable-list ./ranktable/global_ranktable_0.0,0,0.json ./ranktable/global_ranktable_1.1.1.1.json --save-dir ./save_dir
  ```

  pd_ranktable_tools.py的入参说明如下。

  - --mode：脚本的处理模式，可选值为gen或者merge。gen模式表示生成rank_table文件，merge模式表示合并global rank_table文件。
  - --save-dir：保存生成的rank_table文件的根目录，默认为当前目录。
  - --api-server：仅在`gen`模式有效，可选输入，当存在该输入时，表示分离部署的服务入口在该机器。注意，在多台机器启动分离部署时，只能有一台机器存在服务入口。当存在该输入时，会生成local_ranktable_xx_host.json文件，用于在启动推理服务时确定服务入口实例。
  - --prefill-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm全量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应全量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定全量实例。
  - --decode-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm增量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应增量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定增量实例。
  - --global-ranktable-list：仅在`merge`模式有效，必选输入，后续入参表示需要合并的global rank_table，使用空格分隔开。

  执行后，会生成完成合并的global_ranktable_merge.json文件。

- global_rank_table.json格式说明

  server_group_list的长度必须为3，第一个元素(group_id="0")代表Scheduler实例的ip信息，只能有一个实例。

  第二个元素(group_id="1")代表全量实例信息，长度即为全量实例个数。其中需要配置每个全量实例的ip信息以及使用的device信息。rank_id为逻辑卡号，必然从0开始计算，device_id为物理卡号，device_ip则通过上面的hccn_tool获取。

  第三个元素(group_id="2")代表增量实例信息，长度即为增量实例个数。其余信息和全量类似。

  global_rank_table.json具体示例如下：

  ```
  {
     "version": "1.0",
     "status": "completed",
     "server_group_list": [
        {
           "group_id": "0",
           "server_count": "1",
           "server_list": [
              {
                 "server_id": "localhost",
                 "server_ip": "localhost"
              }
           ]
        },
        {
  ```

```
        "group_id": "1",
        "server_count": "1",
        "server_list": [
          {
            "server_id": "localhost",
            "server_ip": "localhost",
            "device": [
              {
                "device_id": "4",
                "device_ip": "10.**.**.22",
                "rank_id": "0"
              },
              {
                "device_id": "5",
                "device_ip": "10.**.**.23",
                "rank_id": "1"
              }
            ]
          }
        ]
      },
      {
        "group_id": "2",
        "server_count": "1",
        "server_list": [
          {
            "server_id": "localhost",
            "server_ip": "localhost",
            "device": [
              {
                "device_id": "6",
                "device_ip": "29.**.**.56",
                "rank_id": "0"
              },
              {
                "device_id": "7",
                "device_ip": "29.**.**.72",
                "rank_id": "1"
              }
            ]
          }
        ]
      }
    ]
}
```

- local_rank_table.json格式说明

  每个全量/增量实例都需要local_rank_table.json。下面以某一个增量实例为例，需要和global_rank_table.json中的增量信息完全对应，group_id为0。

```
{
   "version": "1.0",
   "status": "completed",
   "group_id": "0",
   "server_count": "1",
   "server_list": [
      {
         "server_id": "localhost",
         "server_ip": "localhost",
         "device": [
            {
               "device_id": "6",
               "device_ip": "29.**.**.56",
               "rank_id": "0"
            },
            {
               "device_id": "7",
               "device_ip": "29.**.**.72",
```

```
            "rank_id": "1"
          }
        ]
      }
    ]
}
```

步骤六 启动全量推理实例

以下介绍如何启动全量推理实例。

1. 启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci4、davinci5。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

     说明

     - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
     - driver及npu-smi需同时挂载至容器。
     - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
   - --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
   - {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动全量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_45.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
```

```
--model=${model} \
--tensor-parallel-size=2 \
--max-model-len=4096 \
--max-num-seqs=256 \
--max-num-batched-tokens=4096 \
--host=0.0.0.0 \
--port=8088 \
--served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量或增量推理实例启动的--port参数相关。--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(`--port`)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

参数定义和使用方式与vLLM0.5.0版本一致，此处介绍关键参数。详细参数解释请参见**https://github.com/vllm-project/vllm/blob/main/vllm/engine/arg_utils.py**。

步骤七 启动增量推理实例

1. 启动增量推理容器

   启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

   ```
   docker run -itd \
   --device=/dev/davinci6 \
   ```

```
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci6、davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动增量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_67.json

export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=8089 \
    --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前

实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。

- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用,（英文逗号）分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP地址
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

步骤八 启动 scheduler 实例

建议在PD服务（即全量推理和增量推理服务）启动后，再启动scheduler服务。

1. 启动scheduler容器。启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了0张卡。

- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动scheduler实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=9000 \
   --served-model-name ${served-model-name}

# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。当前端口9000是对外服务端口，而8088、8089则为scheduler调度推理服务端口。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下，

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号。分离部署对外服务使用的是scheduler实例端口，在后续推理性能测试和精度测试时，服务端口需要和scheduler实例端口保持一致。
- --model：HuggingFace下载的官方权重

- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量USE_OPENAI=1时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

说明

- 全量和增量节点的local rank table必须一一对应。
- 全量和增量节点不能使用同一个端口。
- scheduler实例中NODE_PORTS=8088,8089；端口设置顺序必须与global rank table文件中各全量和增量节点顺序一致，否则会报错。

步骤九 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-147**。

通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

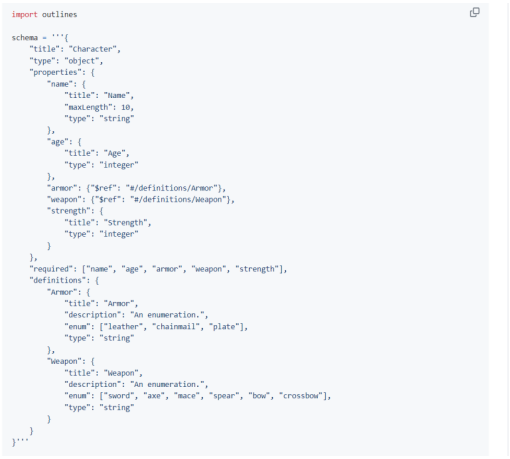
```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-147 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-284** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.16.3 推理性能测试

4.16.3.1 语言模型推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
   |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。

   ```
   cd benchmark_tools
   ```

语言模型脚本相对路径是tools/llm_evaluation/benchmark_tools/benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_parallel.py --backend openai --host ${docker_ip} --port ${port} --tokenizer /path/to/tokenizer  --epochs 5 --num-scheduler-steps 8 \
--parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等后端。本文档使用的推理接口是openai。
- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
- --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。
- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。不输入默认为random。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。
- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-285 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。

- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

```
cd benchmark_tools
conda activate python-3.9.10
```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_serving.py --backend openai --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --num-scheduler-steps 8 --benchmark-csv
benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-286 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

单条请求性能测试

针对openai的/v1/completions以及/v1/chat/completions两个非流式接口，请求体中可以添加可选参数"return_latency"，默认为false，若指定该参数为true，则会在相应请求的返回体中返回字段"latency"，返回内容如下：

1. prefill_latency（首token时延）：请求从到达服务开始到生成首token的耗时
2. model_prefill_latency（模型计算首token时延）：服务从开始计算首token到生成首token的耗时
3. avg_decode_latency（平均增量token时延）：服务计算增量token的平均耗时
4. time_in_queue（请求排队时间）：请求从到达服务开始到开始被调度的耗时
5. request_latency（请求总时延）：请求从到达服务开始到结束的耗时

以上指标单位均是ms，保留2位小数。

4.16.3.2 多模态模型推理性能测试

benchmark 方法介绍

静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
   |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。
   ```
   cd benchmark_tools
   ```
3. 多模态模型脚本相对路径是llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python modal_benchmark_parallel.py \
   --host ${docker_ip} \
   --port ${port} \
   --tokenizer /path/to/tokenizer  \
   --epochs 5 \
   --parallel-num 1 4 8 16 32 \
   --prompt-tokens 1024 2048 \
   --output-tokens 128 256 \
   ```

```
--height ${height} \
--width ${width} \
--benchmark-csv benchmark_parallel.csv
```

参数说明

- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --height: 图片长度（分辨率相关参数）。
- --width: 图片宽度（分辨率相关参数）。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

备注：当前版本仅支持语言+图片多模态性能测试。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中。

4.16.4 推理精度测试

本章节介绍两个精度测评工具。如何使用opencompass工具开展语言模型的推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen；以及使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等，该工具为离线测评，不需要启动推理服务，目前支持大语言模型。

约束限制

- 确保容器可以访问公网。
- 使用opencompass工具需用vllm接口启动在线服务。
- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

使用 Opencompass 精度测评工具

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh          #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。
```
conda activate python-3.9.10
bash install.sh
```
3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。
```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
pip install huggingface-hub==0.25.1
```
4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**5**进行评测。
```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```
5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。
```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```
参数说明:
   - vllm_path：构造vllm评测配置脚本名字，默认为vllm。
   - host：与起服务的host保持一致，比如起服务为0.0.0.0，host设置也为0.0.0.0。
   - service_port：服务端口，与启动服务时的端口保持，比如8080。
   - max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
   - eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
   - model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
   - benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

   参考命令：
```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen    model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size，推理时传入的prompts数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用ppl时必须为True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

   执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

   output_path 指定保存结果的路径。

   参考模型llama3系列模型，数据集mmlu为例，配置如下：

   表 4-148 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu
```

   执行脚本如下

```
# for llama3_8b
python run.py --datasets mmlu_ppl \
--hf-type base --hf-path {hf-path} \
--max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
-w {output_path} --debug
```

参数说明如下：

- --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
- --hf-type：HuggingFace模型权重类型(base,chat)，默认为chat，依据实际的模型选择。
- --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
- --max-seq-len：模型的最大序列长度。
- --max-out-len：模型的最大输出长度。
- --hf-num-gpus：需要使用的卡数。
- --batch-size：推理每次处理的输入数目。
- -w：存放输出结果的目录。

9. 查看精度测试结果。

   默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。

   总体打分结果参考txt和csv文件的最后一行，举例如下：

   npu：

   mmlu：46.6

   gpu：

   mmlu：47

   NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远（小于10）认为分数有效。

使用 Lm-eval 精度测评工具

使用lm-eval工具暂不支持qwen-7b、qwen-14b、qwen-72b、chatglm2-6b、chatglm3-6b模型。

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。

   ```
   rm -rf lm-evaluation-harness/
   git clone https://github.com/EleutherAI/lm-evaluation-harness.git
   cd lm-evaluation-harness
   git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
   pip install -e .
   ```

2. 执行如下精度测试命令，可以根据参数说明修改参数。

   ```
   lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization},distributed_executor_backend='ray' \
   --tasks ${task}  --batch_size ${batch_size}   --log_samples  --cache_requests true  --trust_remote_code  --output_path ${output_path}
   ```

   参数说明:

- model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
  - vllm_path是模型权重路径；
  - max_model_len 是最大模型长度，默认设置为4096；
  - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
  - tensor_parallel_size是使用的卡数；
  - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
  - distributed_executor_backend是开启多进程服务方式，选择ray开启。
- model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
- tasks：评测数据集任务，比如openllm。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
- output_path：结果保存路径。

使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：

```
lm_eval --model vllm  --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,distributed_executor_backend='ray' \
--tasks openllm  --batch_size auto  --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：

```
lm_eval --model vllm  --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant",distributed_executor_backend='ray' \
--tasks openllm  --batch_size auto  --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

4.16.5 推理模型量化

4.16.5.1 使用 AWQ 量化

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-143**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

步骤一 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

AutoAWQ量化工具的适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/AutoAWQ目录下。

1、在容器中使用ma-user用户, vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
cd llm_tools/AutoAWQ
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

步骤二 权重格式离线转换（可选）

在GPU上AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.16.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-143**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
   - --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
   - --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**步骤六 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.16.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化、per-tensor+per-head静态量化以及per-token，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-143**。

per-tensor 静态量化场景

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数。

步骤1 使用tensorRT量化工具进行模型量化。

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

步骤2 抽取kv-cache量化系数。

该步骤的目的是将步骤**步骤1**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

步骤3 启动kv-cache-int8-per-tensor量化服务。

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8_pertensor  #只支持int8，表示kvint8 per-tensor量化
--quantization-param-path kv_cache_scales.json  #输入2. 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

----结束

per-tensor+per-head 静态量化场景

如需使用该场景量化方法，请自行准备kv-cache量化系数，格式和per-tensor静态量化所需的2. 抽取kv-cache量化系数生成的json文件一致，只需把每一层的量化系数修改为列表，列表的长度为kv的头数，列表中每一个值代表每一个kv头使用的量化系数。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": [0.09965550899505615, 0.20214074850082397, 0.16
                "1": [0.07757135480642319, 0.15182086825370789, 0.13
```


在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-tensor+per-head量化服务。

```
--kv-cache-dtype int8_pertensor_perhead  #只支持int8，表示kvint8 per-tensor+per-head量化
--quantization-param-path kv_cache_scales.json  #输入生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

per-token 动态量化场景

如需使用该场景量化方法，推理前向会自动计算kv-cache量化系数，并进行kv的量化。

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-token量化服务。

```
--kv-cache-dtype int8_pertoken  #只支持int8，表示kvint8 per-token量化
```

4.16.5.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**表4-143**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**步骤六 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
          max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.16.5.5 使用 llm-compressor 工具量化

当前版本使用llm-compressor工具量化仅支持Deepseek-v2系列模型的W8A8量化。

本章节介绍如何在GPU的机器上使用开源量化工具**llm-compressor**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   git clone https://github.com/vllm-project/llm-compressor.git
   cd llm-compressor
   pip install -e .
   ```
2. 修改examples/quantizing_moe/deepseek_moe_w8a8_int8.py中的代码：

   1）若本地已有权重，请将MODEL_ID修改为权重路径；
   ```
   MODEL_ID = "deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct"
   ```
   2）若量化Deepseek-V2-236B模型，请将num_gpus改为8；
   ```
   device_map = calculate_offload_device_map(
       MODEL_ID,
       reserve_for_hessians=True,
       num_gpus=8,
       torch_dtype=torch.bfloat16,
       trust_remote_code=True,
   )
   ```
   3）为减少量化时间，建议将以下参数设置为512；
   ```
   NUM_CALIBRATION_SAMPLES = 512
   ```
3. 执行权重量化：
   ```
   python deepseek_moe_w8a8_int8.py
   ```

说明

1、执行权重量化过程中，请保证使用的GPU卡上没有其他进程，否则可能出现OOM；

2、若量化Deepseek-v2-236b模型，大致需要10+小时。

使用量化模型

使用量化模型需要在NPU的机器上运行。

启动vLLM前，请开启图模式（参考**步骤六 启动推理服务**中的配置环境变量），启动服务的命令和启动非量化模型一致。

4.16.6 eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据进行训练eagle小模型，并使用自行训练的小模型进行eagle推理。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装 EAGLE。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的

{

"prefix": "AAA"

"input": "BBB",

"output": "CCC"

}

格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

其中：

input_file_path：预训练json文件地址。

out_file_name：输出的sharegpt格式文件地址。

prefix_name：预训练json文件的前缀 字段名称 （可设置为None，此时预训练数据集只有 input output 两段）输入前缀，（例如：您是一个xxx专家,您需要回答下面问题）

input_name：预训练json文件的指令输入 字段名称（例如：请问苹果是什么颜色）

output_name output：预训练json文件的output字段名称，例如：苹果是红色的。

code_type：预训练json文件编码 默认utf-8

当转换为share gpt格式时，prefix和 input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

若使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

否则使用第二步生成的开源数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--used_npus "0,1,2,3,4,5,6,7" \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048 \
--dtype bfloat16
```

其中

outdir：生成的训练data 地址

end_num：生成的data总条数

used_npus：使用哪些NPU

model_type：使用模型类型 目前支持 qwen2 llama1 llama2 及 llama3，其中llama1、2及chat都填写llama

model_name：模型地址

data_path：预训练数据集地址 即一中生成的文件地址

seed：生成训练data所使用的seed（ 此处42为开源训练设定参数 ）

max_length：模型的max_length

dtype：为模型dtype 默认为bfloat16

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

tmpdir：即为步骤三中的outdir，训练data地址

cpdir：为训练生成权重的地址

configpath：为模型config文件的地址

basepath：为大模型权重地址

bs：为batch大小

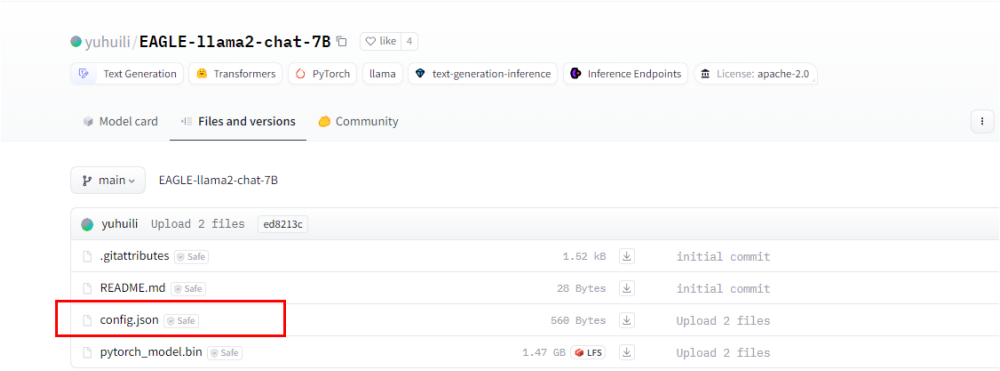
其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-287 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。

图 4-288 eagle config 文件



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

然后在llm_tools/spec_decode/EAGLE文件夹，执行

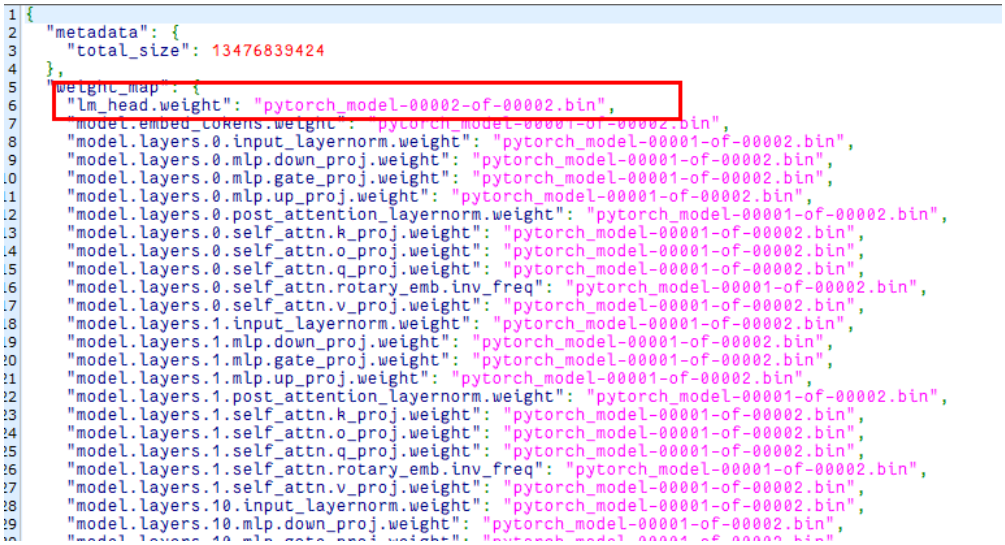
```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

--base-path：为大模型权重地址，例如 ./llama2-7b-chat

--draft-path：小模型权重地址，即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat

--base-weight-name：为大模型包含lm_head的权重文件名，可以在base-path目录下的model.safetensors.index.json文件获取，例如llama2-7b-chat的权重名为pytorch_model-00001-of-00002.bin

图 4-289 权重文件名



--draft-weight-name为小模型权重文件名，即刚才移动的.bin文件或者.safetensors 文件。

4.16.7 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-149 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32

2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128
12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16

22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32
32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64

42	deepseek-llm-7b	1	16	1	32
43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32
51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	2	8	1	16
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	MiniCPM-v2.6	2	4	1	32

59	llama-3.1-405B-AWQ	-	-	8	32
60	qwen2-57b-a14b	-	-	2	16
61	deepseek-v2-lite-16b	2	4	1	4
62	deepseek-v2-236b	-	-	8	4
63	qwen2-vl-7B	2	64	1	64
64	qwen-vl	1	64	1	64
65	qwen-vl-chat	1	64	1	64
66	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.16.8 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`，

  解决方法：该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()
- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查**步骤六中4. 配置环境变量**章节中，高精度模式的环境变量是否开启。
- 问题8：使用autoAWQ进行qwen-7b模型量化时报错TypeError: 'NoneType' object is not subscriptable

  解决办法：修改qwen-7b权重路径下modeling_qwen.py第39行为SUPPORT_FP16 = True

4.17 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.910）

4.17.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-150**所示。

表 4-150 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
57	internvl2-8B	√	x	x	x	x	OpenGVLab/InternVL2-8B at main (huggingface.co)
58	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)
59	internvl2-40B	√	x	x	x	x	OpenGVLab/InternVL2-40B at main (huggingface.co)
60	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
61	deepseek-v2-236b	x	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2**
62	deepseek-v2-lite-16b	√	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite**
63	qwen2-vl-7B	√	x	x	x	x	**Qwen/Qwen2-VL-7B-Instruct at main (huggingface.co)** 注意：Qwen2-VL 开源vllm依赖特定transformers版本， 请手动安装： pip install git+https://github.com/huggingface/transformers.git@21fac7abba2a37fae86106f87fcf9974fd1e3830
64	qwen-vl	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL**
65	qwen-vl-chat	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL-Chat**

66	MiniCPM-v2	√	x	x	x	x	**https://huggingface.co/HwwwH/MiniCPM-V-2**
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

操作流程

图 4-290 操作流程图

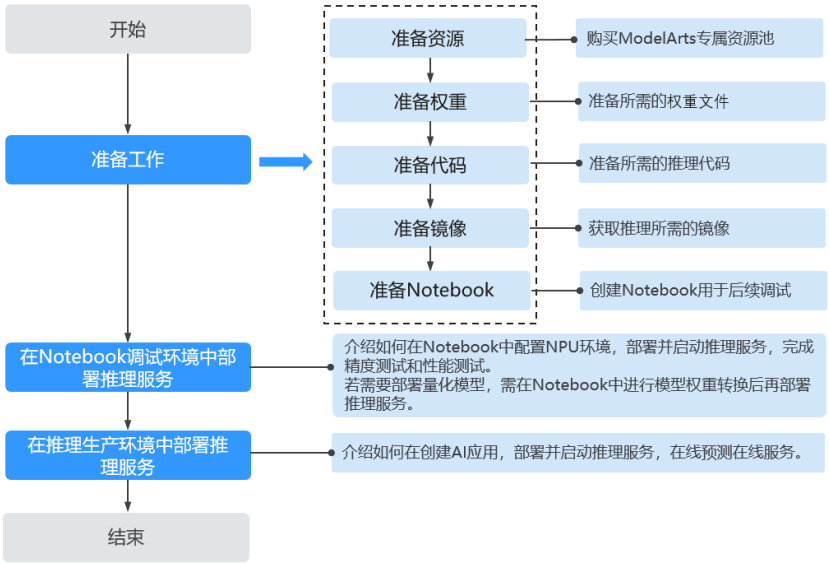


表 4-151 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.910-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 如果需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.17.2 准备工作

4.17.2.1 准备资源

创建专属资源池

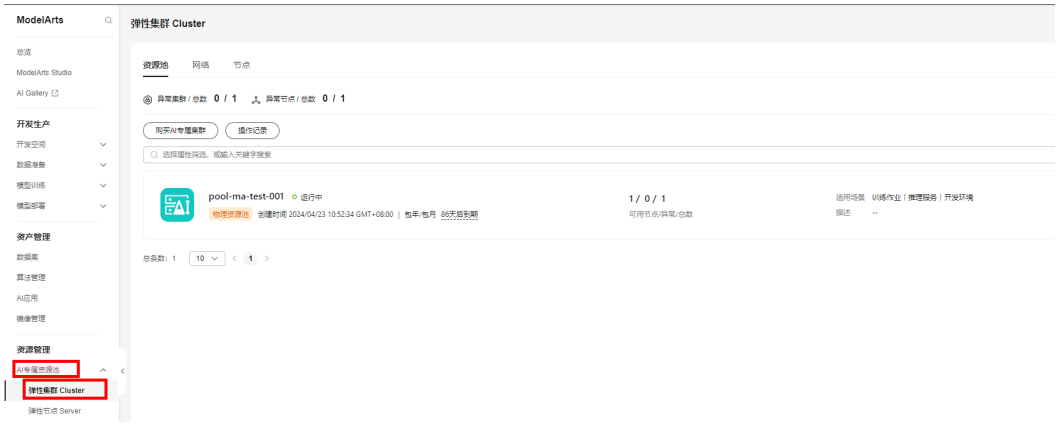
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 4-291 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 4-292 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.17.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.17.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-152**所示。

表 4-152 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
```

```
        ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
        ├── build.sh                # 推理构建脚本
        ├── vllm_install.patch      # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
  ├──AutoAWQ              # W4A16量化工具
     ├──convert_awq_to_npu.py   # awq权重转换脚本
     ├──quantize.py   # 昇腾适配的量化转换脚本
     ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

4.17.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-153 基础容器镜像地址

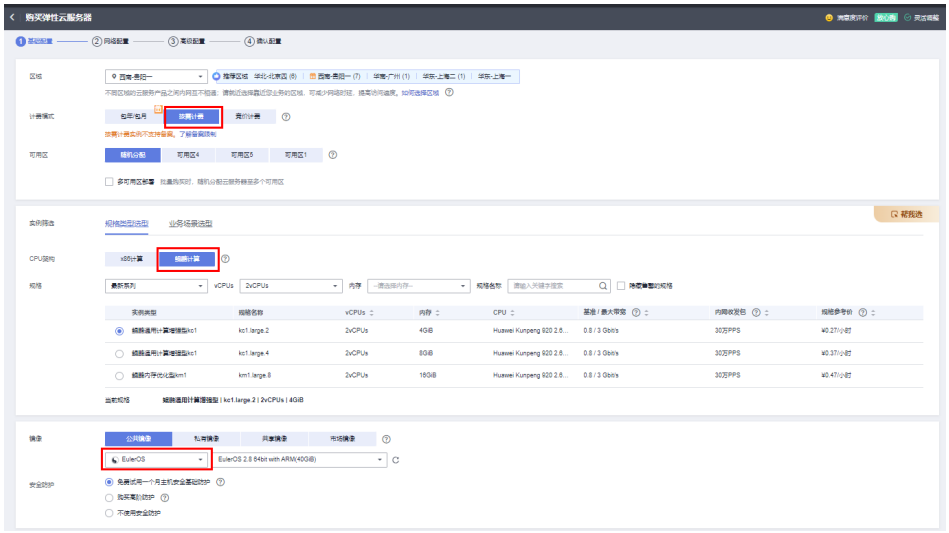
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-293 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-294 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-152**。

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.910-xxx.zip和算子包AscendCloud-OPP-6.3.910-xxx.zip。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```

修改build_image.sh内容，将'ENTRYPOINT ["/home/mind/model/run_vllm.sh"]'修改为'ENTRYPOINT sh /home/mind/model/run_vllm.sh'。

图 4-295 修改 build_image.sh

```
#!/bin/bash

OPTIONS=$(getopt -n "$0" -o i:e:n --long base-image:,specify-enrtypoint:,image-name: -- "$@")

if [ $? != 0 ]; then
    echo "args error"
    exit 1
fi

eval set -- "$OPTIONS"

while true; do
    case "$1" in
        -i|--base-image)
            base_image="$2"
            shift 2
            ;;
        -e|--specify-enrtypoint)
            specify_enrtypoint="$2"
            shift 2
            ;;
        -n|--image-name)
            image_name="$2"
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "unknown options"
            exit 1
            ;;
    esac
done

if [ -z "$base_image" ]; then
    echo "--base-image not specified"
    exit 1
fi

if [ -z "$image_name" ]; then
    echo "--image-name not specified"
    exit 1
fi

if [ -n "$specify_enrtypoint" ]; then
    echo 'ENTRYPOINT sh /home/mind/model/run_vllm.sh' >> Dockerfile
fi

cd ../../../../
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile ./
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/.dockerignore ./

docker build -t $image_name --build-arg BASE_IMAGE=$base_image .

rm -f Dockerfile
rm -f .dockerignore

```

执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

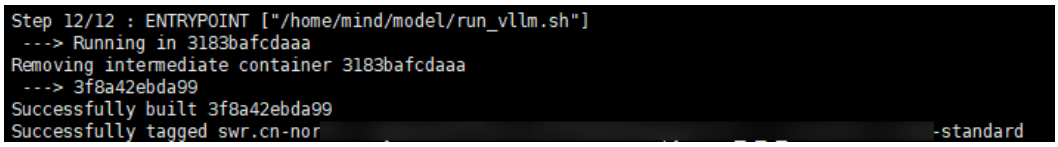
```
sh build_image.sh --base-image=${base_image} --image-name=${image_name} --specify-enrtypoint=True
```

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

图 4-296 成功构建镜像



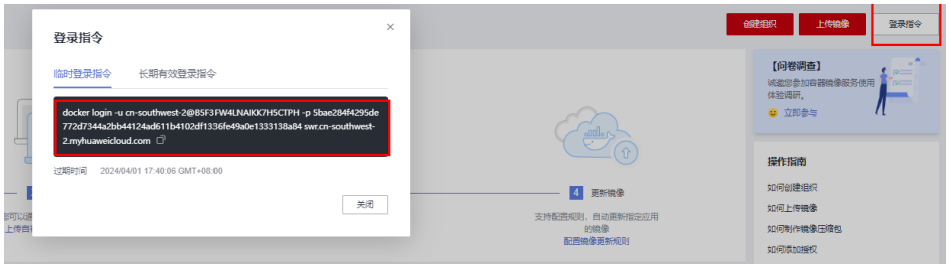
如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-297 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

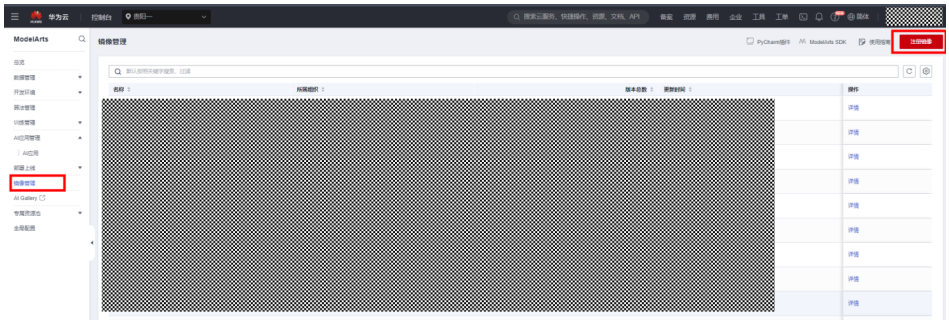
打印如下信息，表示上传镜像成功。

图 4-298 成功上传镜像

Step8 注册镜像

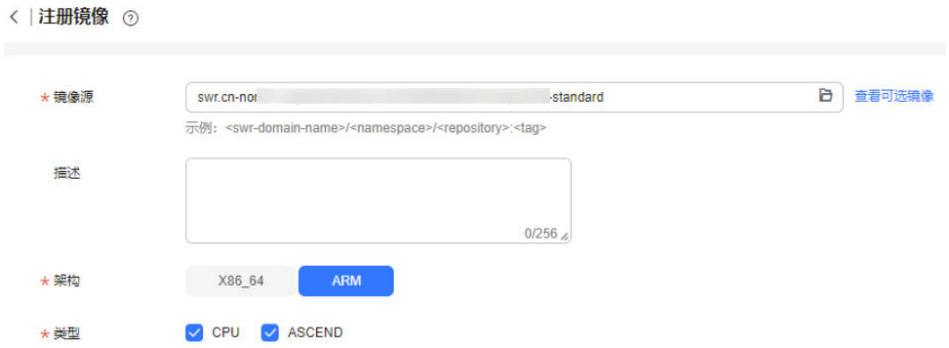
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中单击“注册镜像”。

图 4-299 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 4-300 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.17.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 4-301 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-302 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.17.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

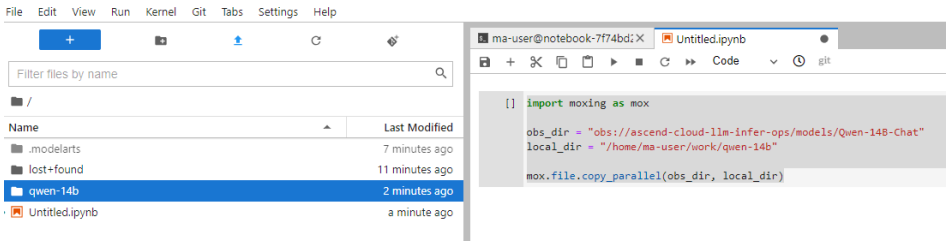
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-303 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-304 查询结果

   ```
   +------------------------------------------------------------------------------------------------+
   | npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
   +---------------------------+---------------+----------------------------------------------------+
   | NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
   | Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
   +===========================+===============+====================================================+
   | 4     910B2               | OK            | 91.4        50                0    / 0           |
   | 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
   +===========================+===============+====================================================+
   | 5     910B2               | OK            | 92.5        51                0    / 0           |
   | 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
   +===========================+===============+====================================================+
   +---------------------------+---------------+----------------------------------------------------+
   | NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
   +===========================+===============+====================================================+
   | 4       0                 | 10915         | python                   | 55400                   |
   +===========================+===============+====================================================+
   | 5       0                 | 21273         | python                   | 55388                   |
   +===========================+===============+====================================================+
   ```

2. 配置环境变量。

   ```
   export USE_PFA_HIGH_PRECISION_MODE=1
   # PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

   export USE_IFA_HIGH_PRECISION_MODE=1
   # IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

   export USE_PREFIX_HIGH_PRECISION_MODE=1
   # 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
   ```

3. 若要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

   ```
   export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
   export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
   export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
   ```

```
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500    # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV    #可选
```

设置动态分档位后，在PTA模式下不支持接收超过最大档的并发请求，超过后会导致推理服务终止。请将最大档（PTA_TORCHAIR_DECODE_GEAR_LIST参数的中设置的最大值）与模型启动时的max-num-seqs保持一致来进行规避。

在MoE模型和小模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8；和Qwen2-1.5B、Qwen2-0.5B。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

4. 若要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，若不开启，目前vllm0.6.0版本与实验室版本权重无法对齐，会导致小模型精度问题。

   ```
   export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
   xport ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
   ```

   如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

   ```
   python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
   ```

   具体可参考 8 eagle 投机小模型训练 **步骤五：训练生成权重转换成可以支持vLLM推理的格式**

5. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。

6. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**

     在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     （1）非多模态

     ```
     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     ```

```
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code
```

（2）多模态

当前支持Llava、InternVL、MiniCPM-v2.6模型，具体模型和权重地址参见**支持的模型列表和权重文件**，推荐显卡数量参见**表4-157**。

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：

  （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

– **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

– -model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。

- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

  如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

  ```
  export USE_PREFIX_HIGH_PRECISION_MODE=1
  ```
- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

  ```
  --enable-lora \
  --lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
  --max-lora-rank=16 \
  --max-loras=32 \
  --max-cpu-loras=32
  ```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-154**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
   "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

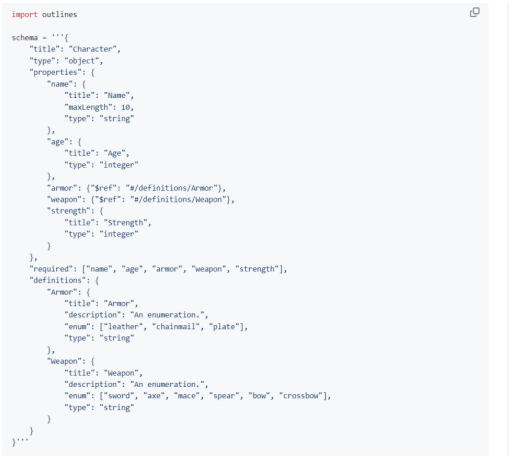
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-154 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。

参数	是否必选	默认值	参数类型	描述
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

参数	是否必选	默认值	参数类型	描述
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-305** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-155 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-156 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.17.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

说明

- 如果需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
- 权重文件夹不要以"model"命名，如果以"model"命名会导致后续创建AI应用报错。

- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-306 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
```

```
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务启动模板参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：

  （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。

- --tensor-parallel-size：并行卡数。此处举例为1，表示使用单卡启动服务。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- -max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。

- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

```
export USE_PREFIX_HIGH_PRECISION_MODE=1
```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
--max-loras=32 \
--max-cpu-loras=32
```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。

- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 准备模型文件和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

可在run_vllm.sh增加如下环境变量开启高阶配置：

a. 配置环境变量。

```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

b. 若要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

在MoE模型和小模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8；和Qwen2-1.5B、Qwen2-0.5B。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

c. 若要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，若不开启，目前vllm0.6.0版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
```

如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE 文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体可参考 8 eagle 投机小模型训练 **步骤五：训练生成权重转换成可以支持vLLM推理的格式**

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 4-307 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-308 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

   首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

图 4-309 创建完成

说明

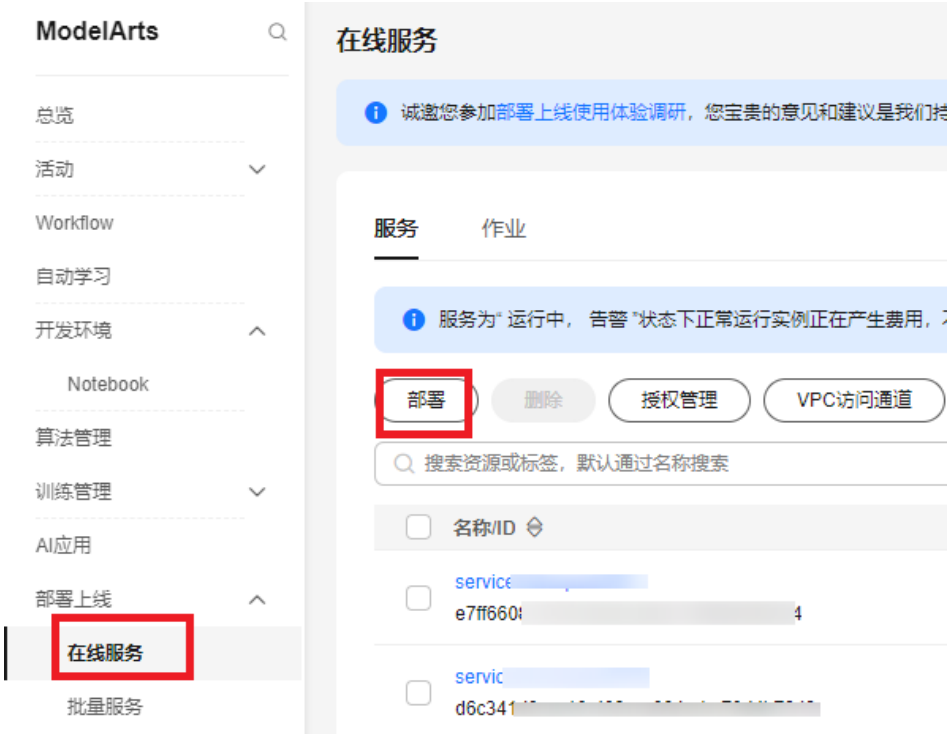
如果权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 4-310 部署在线服务

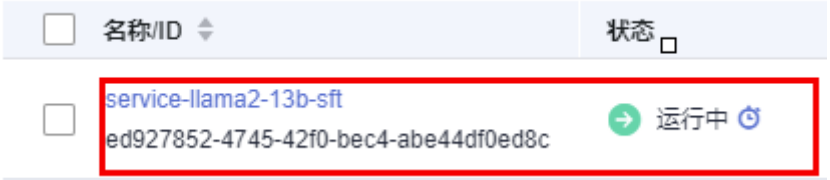


2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-311 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 4-312 服务部署完成

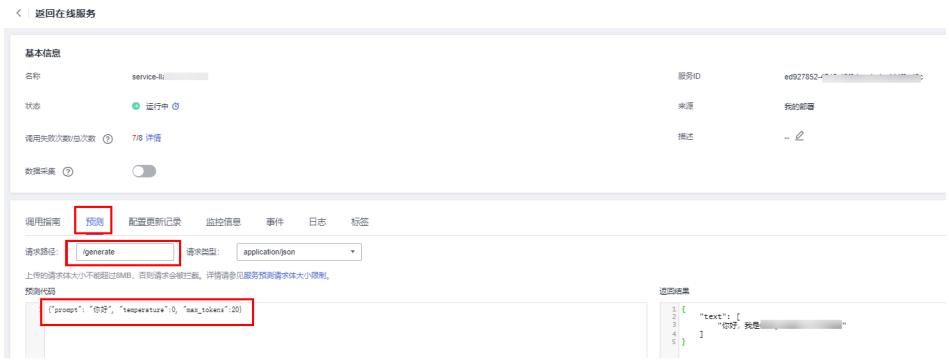


Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

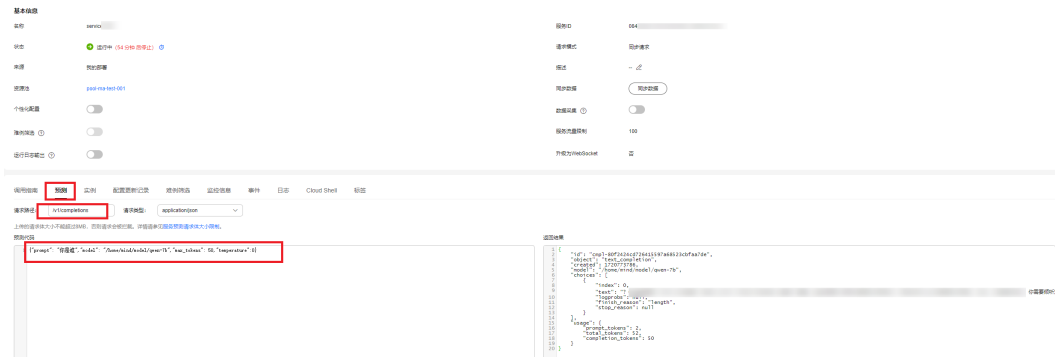
如果以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-313 预测-vllm



如果以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-314 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.17.5 推理精度测试

本章节介绍如何使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等。

约束限制

- 确保容器可以访问公网。

- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

步骤一：配置精度测试环境

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。
```
rm -rf lm-evaluation-harness/
git clone https://github.com/EleutherAI/lm-evaluation-harness.git
cd lm-evaluation-harness
git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
pip install -e .
```
2. 执行如下精度测试命令，可以根据参数说明修改参数。
```
lm_eval --model vllm   --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization} \
--tasks ${task}  --batch_size ${batch_size}   --log_samples  --cache_requests true  --trust_remote_code --output_path ${output_path}
```
参数说明:
   - model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
     - vllm_path是模型权重路径；
     - max_model_len 是最大模型长度，默认设置为4096；
     - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
     - tensor_parallel_size是使用的卡数；
     - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
   - model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
   - tasks：评测数据集任务，比如openllm。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
   - output_path：结果保存路径。

   使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：
```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096 \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```
   使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：
```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant" \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

4.17.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。如果需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，如果静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py    # 工具函数集
├── benchmark.py          # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

目前性能测试已经支持投机推理能力。

执行性能测试脚本前，需先安装相关依赖。

```
conda activate python-3.9.10
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend openai --host 127.0.0.1 --port 8080 --num-scheduler-steps 8 \
--tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

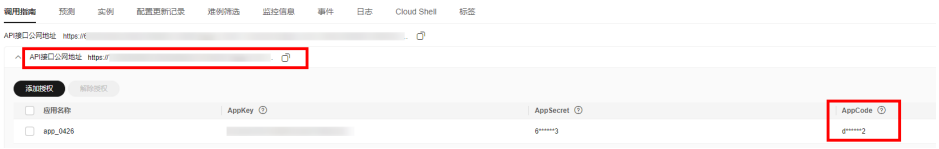
生产环境中进行测试：

```
python benchmark_parallel.py --backend openai --url xxx --app-code xxx --num-scheduler-steps 8 \
--tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-315 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
- --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。
- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。

- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-316 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_dataset.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。

- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

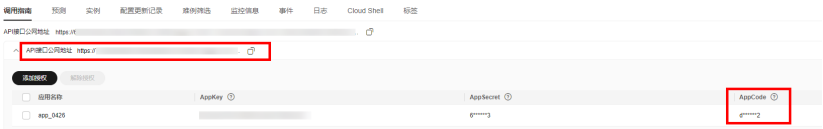
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend openai --host 127.0.0.1 --port 8080 --num-scheduler-steps 8 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend openai --url xxx --app-code xxx --num-scheduler-steps 8 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-317 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。

- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-318 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.17.7 推理模型量化

4.17.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel, W8A16 per-channel

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、在容器中使用ma-user用户， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.17.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.17.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-319 抽取 kv-cache 量化系数


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.17.7.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
```

```
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.17.8 eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据进行训练eagle小模型，并使用自行训练的小模型进行eagle推理。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装 EAGLE。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的

{

"prefix": "AAA"

"input": "BBB",

"output": "CCC"

}

格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

其中：

input_file_path：预训练json文件地址。

out_file_name：输出的sharegpt格式文件地址。

prefix_name：预训练json文件的前缀 字段名称（可设置为None，此时预训练数据集只有 input output 两段）输入前缀，（例如：您是一个xxx专家,您需要回答下面问题）

input_name：预训练json文件的指令输入 字段名称（例如：请问苹果是什么颜色）

output_name output：预训练json文件的output字段名称，例如：苹果是红色的。

code_type：预训练json文件编码 默认utf-8

当转换为share gpt格式时，prefix和 input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

若使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

否则使用第二步生成的开源数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--used_npus "0,1,2,3,4,5,6,7" \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048 \
--dtype bfloat16
```

其中

outdir：生成的训练data 地址

end_num：生成的data总条数

used_npus：使用哪些NPU

model_type：使用模型类型 目前支持 qwen2 llama1 llama2 及 llama3，其中llama1、2及chat都填写llama

model_name：模型地址

data_path：预训练数据集地址 即一中生成的文件地址

seed：生成训练data所使用的seed（此处42为开源训练设定参数）

max_length：模型的max_length

dtype：为模型dtype 默认为bfloat16

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

tmpdir：即为步骤三中的outdir，训练data地址

cpdir：为训练生成权重的地址

configpath：为模型config文件的地址

basepath：为大模型权重地址

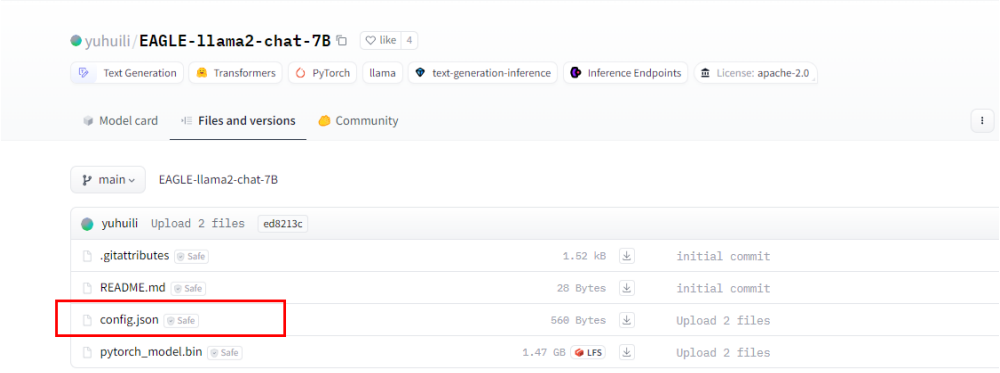
bs：为batch大小

其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-320 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

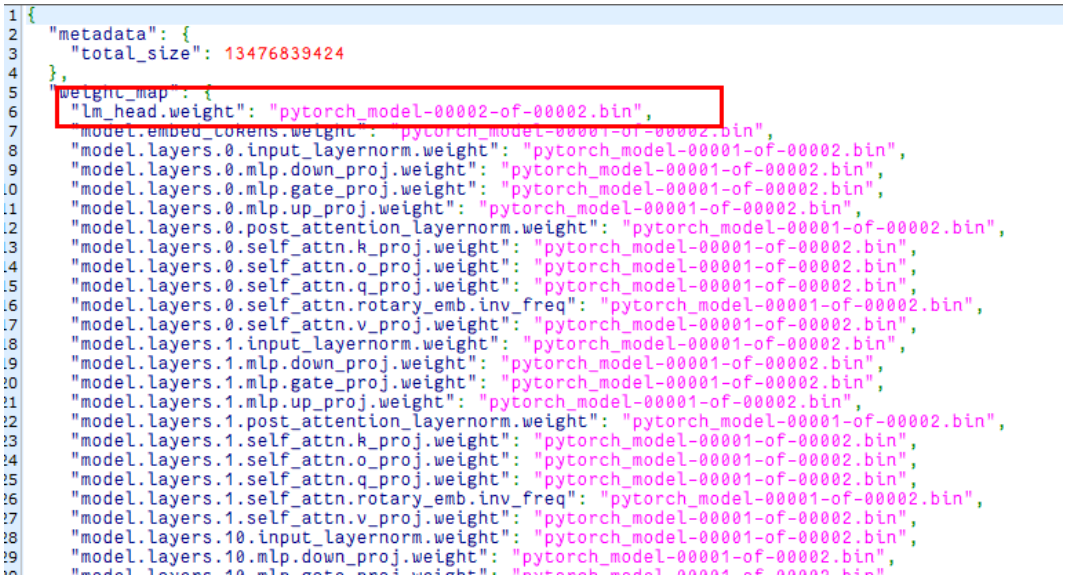
然后在llm_tools/spec_decode/EAGLE文件夹，执行

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

--base-path：为大模型权重地址，例如 ./llama2-7b-chat

--draft-path：小模型权重地址 即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat

--base-weight-name：为大模型包含lm_head的权重文件名，可以在 base-path 目录下的 model.safetensors.index.json 文件获取，例如llama2-7b-chat 的权重名为 pytorch_model-00001-of-00002.bin



```
{
  "metadata": {
    "total_size": 13476839424
  },
  "weight_map": {
    "lm_head.weight": "pytorch_model-00002-of-00002.bin",
    "model.embed_tokens.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.gate_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.up_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.post_attention_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.k_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.o_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.q_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.rotary_emb.inv_freq": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.v_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.gate_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.up_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.post_attention_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.k_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.o_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.q_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.rotary_emb.inv_freq": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.v_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.10.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.10.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
```


--draft-weight-name 为小模型权重文件名，即刚才移动的.bin文件或者.safetensors文件

4.17.9 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-157 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32

2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128
12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16

22	qwen1.5-110b	-	-	8	128
23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32
32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64

42	deepseek-llm-7b	1	16	1	32
43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32
51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	2	8	1	16
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	MiniCPM-v2.6	2	4	1	32

59	llama-3.1-405B-AWQ	-	-	8	32
60	qwen2-57b-a14b	-	-	2	16
61	deepseek-v2-lite-16b	2	4	1	4
62	deepseek-v2-236b	-	-	8	4
63	qwen2-vl-7B	2	64	1	64
64	qwen-vl	1	64	1	64
65	qwen-vl-chat	1	64	1	64
66	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.17.10 附录：Standard 大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

  config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`，

  解决方法：将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()
- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.18 主流开源大模型基于 Lite Cluster 适配 PyTorch NPU 推理指导（6.3.910）

4.18.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite k8s Cluster上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Lite k8s Cluster和昇腾Snt9B资源。
- 本文档中的CCE集群版本选择v1.27~1.28。版本使用的容器引擎为**Containerd。**
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 支持FP16和BF16数据类型推理。
- Lite k8s Cluster驱动版本推荐为23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-158**所示。

表 4-158 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	qwen2.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
30	qwen2.5-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
31	qwen2.5-3b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct
32	qwen2.5-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
33	qwen2.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
34	qwen2.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
35	qwen2.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
36	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
37	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
38	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
39	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
40	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
41	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
42	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
43	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
44	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
45	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
46	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
47	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
48	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
49	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
50	llama-3.2-1B	√	x	x	x	x	Llama-3.2-1B-Instruct · 模型库 (modelscope.cn)
51	llama-3.2-3B	√	x	x	x	x	Llama-3.2-3B-Instruct · 模型库 (modelscope.cn)
52	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
53	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
54	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
55	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
56	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
57	internvl2-8B	√	x	x	x	x	OpenGVLab/InternVL2-8B at main (huggingface.co)
58	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)
59	internvl2-40B	√	x	x	x	x	OpenGVLab/InternVL2-40B at main (huggingface.co)
60	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
61	deepseek-v2-236b	x	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2**
62	deepseek-v2-lite-16b	√	x	√	x	x	**https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite**
63	qwen2-vl-7B	√	x	x	x	x	**Qwen/Qwen2-VL-7B-Instruct at main (huggingface.co)** 注意：Qwen2-VL 开源vllm依赖特定transformers版本， 请手动安装： pip install git+https://github.com/huggingface/transformers.git@21fac7abba2a37fae86106f87fcf9974fd1e3830
64	qwen-vl	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL**
65	qwen-vl-chat	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen-VL-Chat**

66	MiniCPM-v2	√	x	x	x	x	**https://huggingface.co/HwwwH/MiniCPM-V-2**
							注意：需要修改源文件site-packages/timm/layers/pos_embed.py，在第46行上面新增一行代码，如下：
							posemb = posemb.contiguous() #新增
							posemb = F.interpolate(posemb, size=new_size, mode=interpolation,
							antialias=antialias)

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

4.18.2 准备工作

4.18.2.1 准备环境

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**Lite Cluster资源开通**，熟悉集群资源开通流程，再开始操作购买k8s Cluster资源。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

图 4-321 购买 Lite 专属池

k8s Cluster 资源配置

如果已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

kubectl 访问集群配置

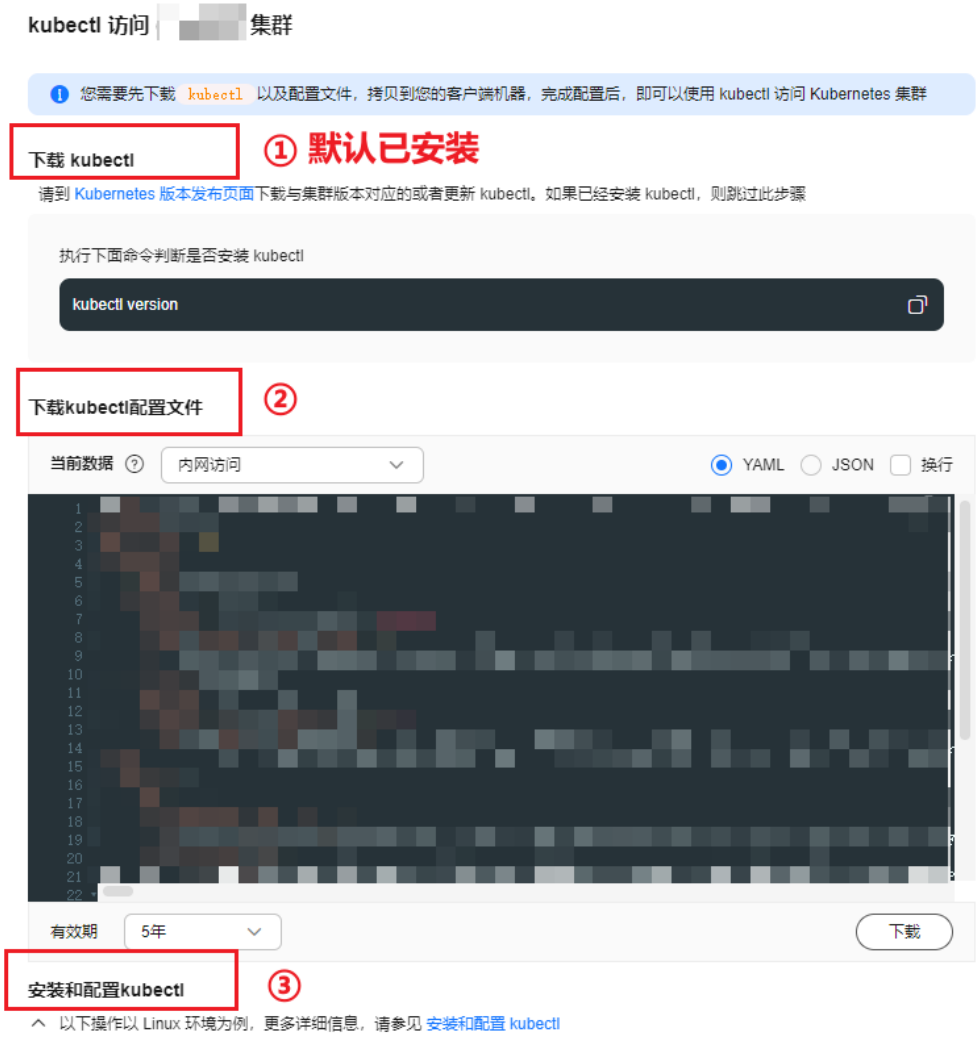
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的CCE集群控制版面中。根据**图4-322**的步骤进行操作，单击kubectl配置时，会弹出**图4-323**步骤页面。

图 4-322 配置中心

2. 根据**图4-323**，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-323 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。如果显示如图4-324的内容，则配置成功。

```
kubectl cluster-info
```

图 4-324 查看 Kubernetes 集群信息正确弹出内容

4.18.2.2 准备代码

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表4-159所示。

表 4-159 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
        ├── build.sh               # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
  ├──llm_tools          # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
    ├──AutoAWQ            # W4A16量化工具
       ├──convert_awq_to_npu.py   # awq权重转换脚本
       ├──quantize.py   # 昇腾适配的量化转换脚本
       ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
     ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
     ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py            #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考《 主流开源大模型基于Lite Cluster适配PyTorch训练指导 》。

4.18.2.3 准备镜像

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-160 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v   # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择 containerd 作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

   buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

   buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

   a. 下载并解压buildkit程序。
      ```
      # 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
      wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

      # 创建解压的目录
      mkdir /usr/local/buildkit

      # 解压到指定的目录
      tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit
      ```

```
# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

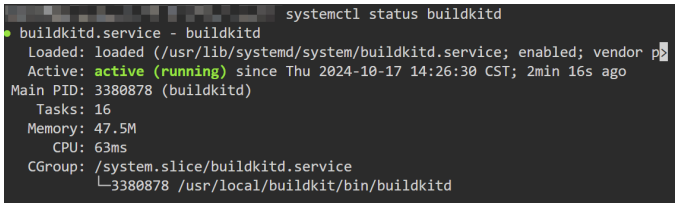
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-160**。

containerd 容器引擎有命名空间的概念。**Kubernetes 下使用的 containerd 默认命名空间是 k8s.io**。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。

  ```
  ctr -n k8s.io images pull {image_url}
  ```

- 使用 nerdctl 工具进行镜像拉取。

  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

注意：集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

Step3 制作推理镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-159**。

1. 解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.910-xxx.zip，并直接进入到llm_inference/ascend_vllm文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
   ```
2. 执行以下命令制作推理镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。
   ```
   nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称> --build-arg BASE_IMAGE=${base_image} .
   ```
   注意：nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以使用nerdctl pull命令拉取镜像，查看是否能拉取成功。
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
   - ${base_image}为基础镜像地址。

如果推理需要使用NPU加速图片预处理，适配了llava-1.5模型，启动时需要设置export ENABLE_USE_DVPP=1，需要安装torchvision_npu，可放到镜像制作脚本./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockfile中。内容如下：

```
git clone https://gitee.com/ascend/vision.git vision_npu
cd vision_npu
git checkout v0.16.0-6.0.rc3
# 安装依赖库
pip3 install -r requirement.txt
# 编包
python setup.py bdist_wheel
# 安装
cd dist
pip install torchvision_npu-0.16.*.whl
```

4.18.3 部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

前提条件

- 已准备好Lite k8s Cluster环境，具体参考**准备环境**。推荐使用“西南-贵阳一”Region上的Cluster和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保集群可以访问公网。

Step1 上传权重文件

将权重文件上传到集群节点机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**支持的模型列表和权重文件**。

如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step2 配置 pod

在节点自定义目录${node_path}下创建config.yaml文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
    app: infers
spec:
  replicas: 1
  selector:
    matchLabels:
      app: infers
  template:
    metadata:
      labels:
        app: infers
    spec:
      schedulerName: volcano
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      containers:
      - image: ${image_name}                # 推理镜像名称
        imagePullPolicy: IfNotPresent
        name: ${container_name}
        securityContext:
          runAsUser: 0
        ports:
        - containerPort: 8080
        command: ["/bin/bash", "-c"]
        args: ["${node-path}/run_vllm.sh"]          # 节点自定义目录，该目录下包含pod配置文件config.yaml和推理服务启动脚本run_vllm.sh
        resources:
          requests:
            huawei.com/ascend-1980: "8"             # 需求卡数，key保持不变。
          limits:
            huawei.com/ascend-1980: "8"             # 限制卡数，key保持不变。
        volumeMounts:                           # 容器内部映射路径
        - name: ascend-driver               #驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons          #驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: hccn                            #驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: localtime
          mountPath: /etc/localtime
        - name: npu-smi                           # npu-smi
          mountPath: /usr/local/sbin/npu-smi
        - name: model-path                      # 模型权重路径
          mountPath: ${model-path}
        - name: node-path
          mountPath: ${node-path}
      volumes:                                  # 物理机外部路径
      - name: ascend-driver
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
      - name: model-path
        hostPath:
          path: ${model-path}
```

```
  - name: node-path
    hostPath:
      path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。
- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml和推理服务启动脚本run_vllm.sh，run_vllm.sh内容见**Step3 创建服务启动脚本**。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

Step3 创建服务启动脚本

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--num-scheduler-steps=8 \
--trust-remote-code
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
# PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
# 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--chat-template ${chat_template_path} \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  多模态推理服务启动模板参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。

- PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
- --chat-template：对话构建模板，可选参数。如：

  （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：**Step1 上传权重文件**中上传的模型权重路径。
- --tensor-parallel-size：并行卡数。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。开启投机推理后无需配置该参数。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用

prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

```
export USE_PREFIX_HIGH_PRECISION_MODE=1
```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
--max-loras=32 \
--max-cpu-loras=32
```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。该参数可与投机推理配合使用，实现投机校验模型的量化功能。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 上传权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。
- --num-speculative-tokens：投机推理草稿模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

  可在run_vllm.sh增加如下环境变量开启高阶配置：

  i. 配置环境变量。

```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

ii. 如果要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，如果不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1   # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32    # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500    # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV    #可选
```

通过PTA_TORCHAIR_DECODE_GEAR_LIST设置动态分档位后，在PTA模式下，会根据服务启动时的max_num_seqs参数对档位进行调整，使得最终的最大档位为max_num_seqs，因此，请根据使用场景合理设置动态分档以及max_num_seqs参数，避免档位过大导致图编译错误。

在MoE模型和小模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8；和Qwen2-1.5B、Qwen2-0.5B。当前MoE模型图模式启动不支持multi step。

MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

iii. 如果要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，如果不开启，目前vllm0.6.0版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
export ENABLE_SPEC_METRIC=0  # 是否关闭投机推理的metric采集功能，关闭有助于提升投机推理性能，默认关闭
```

如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode/EAGLE 文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

具体可参考7 eagle投机小模型训练 **步骤五：训练生成权重转换成可以支持vLLM推理的格式**

Step4 创建 pod

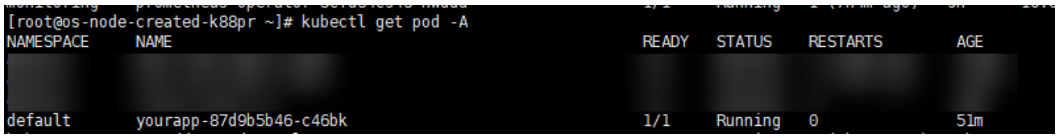
在节点自定义目录${node_path}下执行如下命令创建pod。

```
kubectl apply -f config.yaml
```

检查pod启动情况，执行下述命令。如果显示“1/1 running”状态代表启动成功。

```
kubectl get pod -A
```

图 4-325 启动 pod 成功



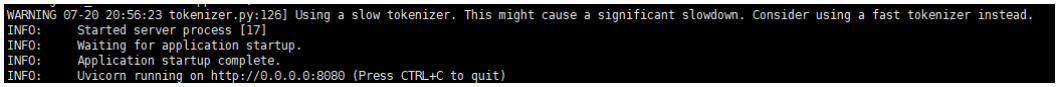
执行如下命令查看pod日志，如果打印类似下图信息表示服务启动成功。

```
kubectl logs -f ${pod_name}
```

参数说明：

- ${pod_name}：pod名，例如图4-325${pod_name}为yourapp-87d9b5b46-c46bk。

图 4-326 启动服务成功



Step5 推理请求

执行如下命令进入容器。

```
kubectl exec -it {pod_name} bash
```

参数说明：

- ${pod_name}：pod名，例如图4-325${pod_name}为yourapp-87d9b5b46-c46bk。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见表4-161。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和Step3 创建服务启动脚本中设置的宿主机端口保持一致。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
```

```
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

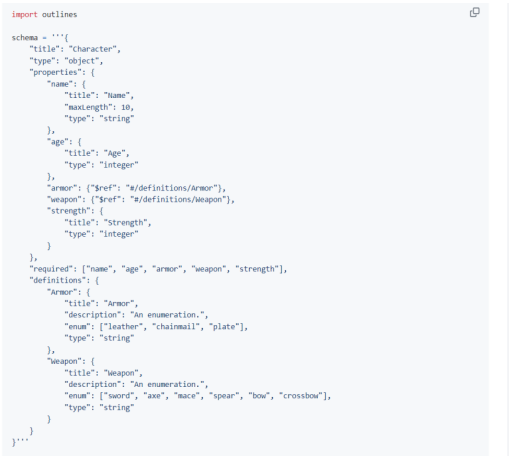
```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-161 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。

参数	是否必选	默认值	参数类型	描述
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-327** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-162 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-163 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

4.18.4 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step3 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 执行如下命令进入容器。
   ```
   kubectl exec -it {pod_name} bash
   ```
   ${pod_name}：pod名，例如**图4-325**${pod_name}为yourapp-87d9b5b46-c46bk。
3. 进入benchmark_tools目录下，切换conda环境并安装依赖。
   ```
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
   conda activate python-3.9.10
   pip install -r requirements.txt
   ```
4. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python benchmark_parallel.py --backend openai --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --num-scheduler-steps 8 --benchmark-csv benchmark_parallel.csv
   ```
   参数说明
   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务部署的IP。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
   - --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
   - --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1
   - --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。
   - --prefix-caching-num：构造的prompt的公共前缀的序列长度，prefix-caching-num值需小于prompt-tokens。

- --use-spec-decode：是否使用投机推理进行输出统计，不输入默认为false。当使用投机推理时必须开启，否则会导致输出token数量统计不正确。注：由于投机推理的性能测试使用随机输入意义不大，建议开启--dataset-type、--dataset-path，并选择性开启--use-real-dataset-output-tokens使用真实数据集进行测试。
- --dataset-type：当使用投机推理时开启，benchmark使用的数据类型，当前支持random、sharegpt、human-eval三种输入。random表示构造随机token的数据集进行测试；sharegpt表示使用sharegpt数据集进行测试；human-eval数据集表示使用human-eval数据集进行测试。注意：当输入为sharegpt或human-eval时，测试数据的输入长度为数据集的真实长度，--prompt-tokens的值会被忽略。
- --dataset-path：数据集的路径，仅当--dataset-type为sharegpt或者human-eval的时候生效。
- --use-real-dataset-output-tokens：当使用投机推理时开启，设置输出长度是否使用数据集的真实长度，不输入默认为false。当使用该选项时，测试数据的输出长度为数据集的真实长度，--output-tokens的值会被忽略。
- --num-speculative-tokens：仅当开启--use-spec-decode时生效，需和服务启动时配置的--num-speculative-tokens一致。默认为-1。当该值大于等于0时，会基于该值计算投机推理的接受率指标。

5. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-328 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 执行如下命令进入容器。
```
kubectl exec -it {pod_name} bash
```
${pod_name}：pod名，例如图4-325${pod_name}为yourapp-87d9b5b46-c46bk。

2. 进入benchmark_tools目录下，切换conda环境。
```
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
conda activate python-3.9.10
```

3. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   generate_dataset.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。
   - --num-requests：输出数据集的数量，可以根据实际需求设置。

4. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   ```
   python benchmark_serving.py --backend openai --host 127.0.0.1 --port 8080 --dataset
   custom_datasets.json --dataset-type custom \
   --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
   1000 1000 1000 \
   --max-tokens 4096 --max-prompt-tokens 3768 --num-scheduler-steps 8 --benchmark-csv
   benchmark_serving.csv
   ```

   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一

致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。

- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1

5. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-329 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.18.5 推理精度测试

本章节介绍如何使用lm-eval工具开展语言模型的推理精度测试，数据集包含mmlu、ARC_Challenge、GSM_8k、Hellaswag、Winogrande、TruthfulQA等。

约束限制

- 确保容器可以访问公网。
- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。
- 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

步骤一：配置精度测试环境

1. 精度评测可以在原先conda环境，进入到一个固定目录下，执行如下命令。

```
rm -rf lm-evaluation-harness/
git clone https://github.com/EleutherAI/lm-evaluation-harness.git
cd lm-evaluation-harness
git checkout 383bbd54bc621086e05aa1b030d8d4d5635b25e6
pip install -e .
```

2. 执行如下精度测试命令，可以根据参数说明修改参数。

```
lm_eval --model vllm  --model_args pretrained=${vllm_path},dtype=auto,tensor_parallel_size=${tensor_parallel_size},gpu_memory_utilization=${gpu_memory_utilization},add_bos_token=True,max_model_len=${max_model_len},quantization=${quantization} \
--tasks ${task}  --batch_size ${batch_size}  --log_samples  --cache_requests true  --trust_remote_code  --output_path ${output_path}
```

参数说明:

- model_args：标志向模型构造函数提供额外参数，比如指定运行模型的数据类型；
  - vllm_path是模型权重路径；
  - max_model_len 是最大模型长度，默认设置为4096；
  - gpu_memory_utilization是gpu利用率，如果模型出现oom报错，调小参数；
  - tensor_parallel_size是使用的卡数；
  - quantization是量化参数，使用非量化权重，去掉quantization参数；如果使用awq、smoothquant或者gptq加载的量化权重，根据量化方式选择对应参数，可选awq，smoothquant，gptq。
- model：模型启动模式，可选vllm，openai或hf，hf代表huggingface。
- tasks：评测数据集任务，比如openllm。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度，默认使用auto，代表自动选择batch大小。
- output_path：结果保存路径。

使用lm-eval，比如加载非量化或者awq量化，llama3.2-1b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer/Llama-3.2-1B-Instruct/",dtype=auto,tensor_parallel_size=1,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096 \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

使用lm-eval，比如smoothquant量化，llama3.1-70b模型的权重，参考命令：

```
lm_eval --model vllm   --model_args pretrained="/data/nfs/benchmark/tokenizer_w8a8/llama3.1-70b/",dtype=auto,tensor_parallel_size=4,gpu_memory_utilization=0.7,add_bos_token=True,max_model_len=4096,quantization="smoothquant" \
--tasks openllm   --batch_size auto   --log_samples  --cache_requests true  --trust_remote_code --output_path ./
```

4.18.6 推理模型量化

4.18.6.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

Step1 环境准备

1. 在节点自定义目录${node_path}下创建config.yaml文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
    app: infers
spec:
```

```
replicas: 1
selector:
  matchLabels:
    app: infers
template:
  metadata:
    labels:
      app: infers
  spec:
    schedulerName: volcano
    nodeSelector:
      accelerator/huawei-npu: ascend-1980
    containers:
    - image: ${image_name}                  # 推理镜像名称
      imagePullPolicy: IfNotPresent
      name: ${container_name}
      securityContext:
        runAsUser: 0
      ports:
      - containerPort: 8080
      command:
      - "sleep"
      - "1000000000000000000"
      resources:
        requests:
          huawei.com/ascend-1980: "8"            # 需求卡数，key保持不变。
        limits:
          huawei.com/ascend-1980: "8"            # 限制卡数，key保持不变。
      volumeMounts:                           # 容器内部映射路径
      - name: ascend-driver               #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/driver
      - name: ascend-add-ons           #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/add-ons
      - name: hccn                          #驱动hccn配置，保持不动
        mountPath: /etc/hccn.conf
      - name: localtime
        mountPath: /etc/localtime
      - name: npu-smi                          # npu-smi
        mountPath: /usr/local/sbin/npu-smi
      - name: model-path                       # 模型权重路径
        mountPath: ${model-path}
      - name: node-path                     # 节点自定义目录，该目录下包含pod配置文件config.yaml
        mountPath: ${node-path}
    volumes:                                # 物理机外部路径
    - name: ascend-driver
      hostPath:
        path: /usr/local/Ascend/driver
    - name: ascend-add-ons
      hostPath:
        path: /usr/local/Ascend/add-ons
    - name: hccn
      hostPath:
        path: /etc/hccn.conf
    - name: localtime
      hostPath:
        path: /etc/localtime
    - name: npu-smi
      hostPath:
        path: /usr/local/sbin/npu-smi
    - name: model-path
      hostPath:
        path: ${model-path}
    - name: node-path
      hostPath:
        path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。

- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

2. 参考**Step4 创建pod**创建pod以用于后续进行模型量化

Step2 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 执行如下命令进入容器，并进入AutoAWQ目录下， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
kubectl exec -it {pod_name} bash
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoAWQ
bash build.sh
```

2. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step3 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step4 启动 AWQ 量化服务

参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q awq 或者--quantization awq
```

4.18.6.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 参考**Step1 环境准备**创建pod准备量化环境。
2. 执行如下命令进入容器，并进入AutoSmoothQuant目录下
   ```
   kubectl exec -it {pod_name} bash
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples
   ```
3. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-330 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 92.5        51                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+

+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+====================================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

4. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

5. 启动smoothQuant量化服务。

参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.18.6.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.18.6.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 创建服务启动脚本**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.18.7 eagle 投机小模型训练

本章节提供eagle小模型自行训练的能力，客户可通过本章节，使用自己的数据进行训练eagle小模型，并使用自行训练的小模型进行eagle推理。

步骤一：安装 Eagle

Eagle训练适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/spec_decode/EAGLE目录下。

在目录下执行如下命令，即可安装 EAGLE。

```
bash  build.sh
```

步骤二：非 sharegpt 格式数据集转换（可选）

如果数据集json文件不是sharegpt格式，而是常见的

{

"prefix": "AAA"

"input": "BBB",

"output": "CCC"

}

格式，则需要执行convert_to_sharegpt.py 文件将数据集转换为share gpt格式。

```
python convert_to_sharegpt.py \
 --input_file_path data_test.json  \
 --out_file_name ./data_for_sharegpt.json \
 --prefix_name instruction \
 --input_name input  \
 --output_name output \
 --code_type  utf-8
```

其中：

input_file_path：预训练json文件地址。

out_file_name：输出的sharegpt格式文件地址。

prefix_name：预训练json文件的前缀 字段名称 （可设置为None，此时预训练数据集只有 input output 两段）输入前缀，（例如：您是一个xxx专家,您需要回答下面问题）

input_name：预训练json文件的指令输入 字段名称（例如：请问苹果是什么颜色）

output_name output：预训练json文件的output字段名称，例如：苹果是红色的。

code_type：预训练json文件编码 默认utf-8

当转换为share gpt格式时，prefix和 input会拼接成一段文字，作为human字段，提出问题，而output字段会作为gpt字段，做出回答。

步骤三：sharegpt 格式数据生成为训练 data 数据集

若使用开源数据集，推荐使用原论文代码仓数据集，下载地址：https://huggingface.co/datasets/Aeala/ShareGPT_Vicuna_unfiltered/blob/main/ShareGPT_V4.3_unfiltered_cleaned_split.json

否则使用第二步生成的开源数据集。

```
python allocation.py \
--outdir outdir0/sharegpt_0_99_mufp16 \
--end_num 100 \
--used_npus "0,1,2,3,4,5,6,7" \
--model_type llama \
--model_name ./llama-7B  \
--data_path data_for_sharegpt.json \
--seed 42 \
--max_length 2048 \
--dtype bfloat16
```

其中

outdir：生成的训练data 地址

end_num：生成的data总条数

used_npus：使用哪些NPU

model_type：使用模型类型 目前支持 qwen2 llama1 llama2 及 llama3，其中llama1、2及chat都填写llama

model_name：模型地址

data_path：预训练数据集地址 即一中生成的文件地址

seed：生成训练data所使用的seed（ 此处42为开源训练设定参数 ）

max_length：模型的max_length

dtype：为模型dtype 默认为bfloat16

步骤四：执行训练

安装完成后，执行：

```
accelerate launch -m --mixed_precision=bf16 eagle.train.main \
--tmpdir [path of data] \
--cpdir [path of checkpoints] \
--configpath [path of config file] \
--basepath  [path of base_model]
--bs [batch size]
```

tmpdir：即为步骤三中的outdir，训练data地址

cpdir：为训练生成权重的地址

configpath：为模型config文件的地址

basepath：为大模型权重地址

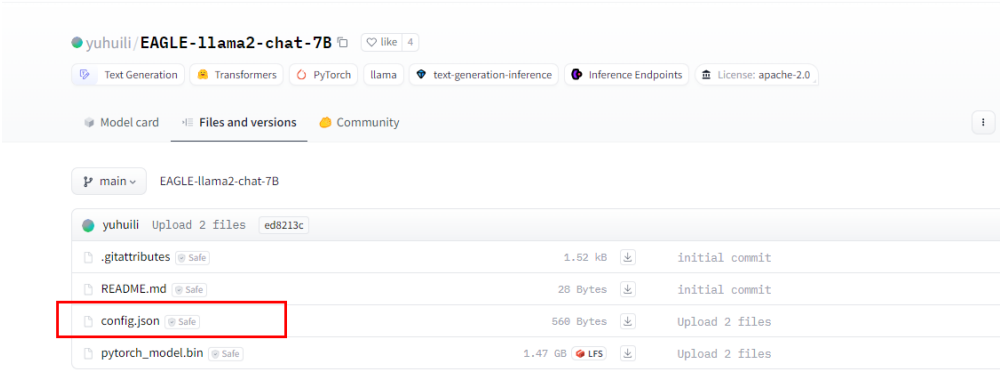
bs：为batch大小

其中，要获取模型config文件， 首先到https://github.com/SafeAILab/EAGLE/页找到对应eagle模型地址。

图 4-331 EAGLE Weights

Base Model	EAGLE on Hugging Face	# EAGLE Parameters	Base Model	EAGLE on Hugging Face	# EAGLE Parameters
Vicuna-7B-v1.3	yuhuili/EAGLE-Vicuna-7B-v1.3	0.24B	LLaMA2-Chat 7B	yuhuili/EAGLE-llama2-chat-7B	0.24B
Vicuna-13B-v1.3	yuhuili/EAGLE-Vicuna-13B-v1.3	0.37B	LLaMA2-Chat 13B	yuhuili/EAGLE-llama2-chat-13B	0.37B
Vicuna-33B-v1.3	yuhuili/EAGLE-Vicuna-33B-v1.3	0.56B	LLaMA2-Chat 70B	yuhuili/EAGLE-llama2-chat-70B	0.99B
Mixtral-8x7B-Instruct-v0.1	yuhuili/EAGLE-mixtral-instruct-8x7B	0.28B			
LLaMA3-Instruct 8B	yuhuili/EAGLE-LLaMA3-Instruct-8B	0.25B	LLaMA3-Instruct 70B	yuhuili/EAGLE-LLaMA3-Instruct-70B	0.99B
Qwen2-7B-Instruct	yuhuili/EAGLE-Qwen2-7B-Instruct	0.26B	Qwen2-72B-Instruct	yuhuili/EAGLE-Qwen2-72B-Instruct	1.05B

以llama2-chat-7B为例，单击进入后 ，如下图所示config文件，即为对应模型的eagle config文件。



步骤五：训练生成权重转换成可以支持 vLLM 推理的格式

将训练完成后的权重文件（.bin文件或. safetensors文件），移动到下载好的开源权重目录下（即步骤4中，config文件所在目录）。

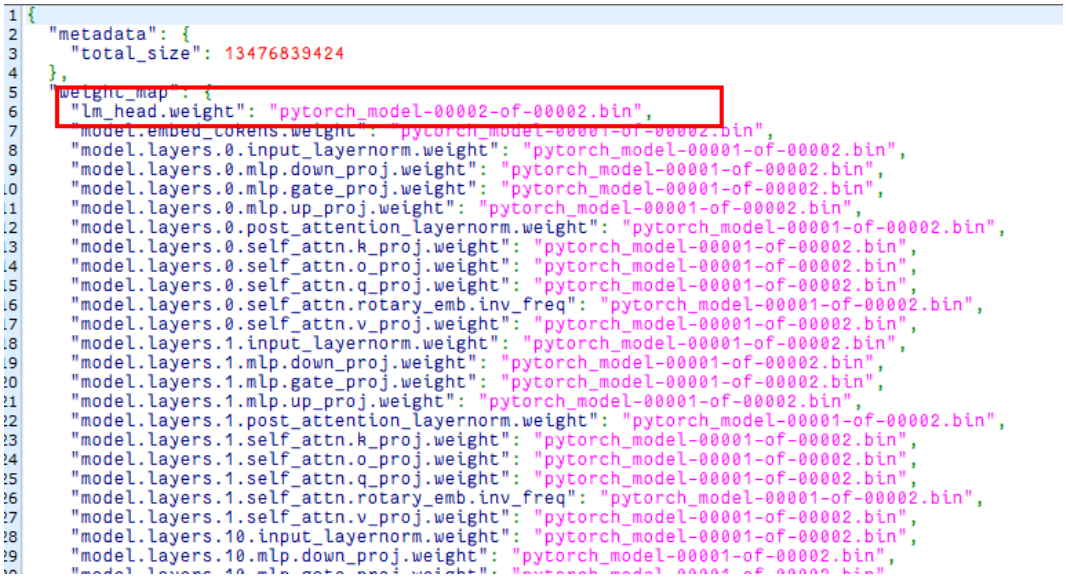
然后在llm_tools/spec_decode/EAGLE文件夹，执行

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

--base-path：为大模型权重地址，例如 ./llama2-7b-chat

--draft-path：小模型权重地址 即步骤四中config文件所在目录，例如 ./eagle_llama2-7b-chat

--base-weight-name：为大模型包含lm_head的权重文件名，可以在 base-path 目录下的 model.safetensors.index.json 文件获取，例如llama2-7b-chat 的权重名为 pytorch_model-00001-of-00002.bin



```
{
  "metadata": {
    "total_size": 13476839424
  },
  "weight_map": {
    "lm_head.weight": "pytorch_model-00002-of-00002.bin",
    "model.embed_tokens.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.gate_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.mlp.up_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.post_attention_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.k_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.o_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.q_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.rotary_emb.inv_freq": "pytorch_model-00001-of-00002.bin",
    "model.layers.0.self_attn.v_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.gate_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.mlp.up_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.post_attention_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.k_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.o_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.q_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.rotary_emb.inv_freq": "pytorch_model-00001-of-00002.bin",
    "model.layers.1.self_attn.v_proj.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.10.input_layernorm.weight": "pytorch_model-00001-of-00002.bin",
    "model.layers.10.mlp.down_proj.weight": "pytorch_model-00001-of-00002.bin",
```


--draft-weight-name 为小模型权重文件名，即刚才移动的.bin文件或者.safetensors文件

4.18.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-164 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16

3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3.1-8b	1	32	1	128
9	llama3-70b	8	32	4	64
10	llama3.1-70b	8	32	4	64
11	llama3.2-1b	1	128	1	128
12	llama3.2-3b	1	128	1	128
13	qwen-7b	1	8	1	32
14	qwen-14b	2	16	1	16
15	qwen-72b	8	8	4	16
16	qwen1.5-0.5b	1	128	1	256
17	qwen1.5-7b	1	8	1	32
18	qwen1.5-1.8b	1	64	1	128
19	qwen1.5-14b	2	16	1	16
20	qwen1.5-32b	4	32	2	64
21	qwen1.5-72b	8	8	4	16
22	qwen1.5-110b	-	-	8	128

23	qwen2-0.5b	1	128	1	256
24	qwen2-1.5b	1	64	1	128
25	qwen2-7b	1	8	1	32
26	qwen2-72b	8	32	4	64
27	qwen2.5-0.5b	1	32	1	32
28	qwen2.5-1.5b	1	32	1	32
29	qwen2.5-3b	1	32	1	32
30	qwen2.5-7b	1	32	1	32
31	qwen2.5-14b	2	32	1	32
32	qwen2.5-32b	4	32	2	64
33	qwen2.5-72b	8	32	4	32
34	chatglm2-6b	1	64	1	128
35	chatglm3-6b	1	64	1	128
36	glm-4-9b	1	32	1	128
37	baichuan2-7b	1	8	1	32
38	baichuan2-13b	2	4	1	4
39	yi-6b	1	64	1	128
40	yi-9b	1	32	1	64
41	yi-34b	4	32	2	64
42	deepseek-llm-7b	1	16	1	32

43	deepseek-coder-33b-instruct	4	32	2	64
44	deepseek-llm-67b	8	32	4	64
45	mistral-7b	1	32	1	128
46	mixtral-8x7b	4	8	2	32
47	gemma-2b	1	64	1	128
48	gemma-7b	1	8	1	32
49	falcon-11b	1	8	1	64
50	llava-1.5-7b	1	16	1	32
51	llava-1.5-13b	1	8	1	16
52	llava-v1.6-7b	1	16	1	32
53	llava-v1.6-13b	1	8	1	16
54	llava-v1.6-34b	4	32	2	64
55	internvl2-8b	2	8	1	16
56	internvl2-26b	2	8	1	8
57	internvl2-40b	-	-	2	32
58	MiniCPM-v2.6	2	4	1	32
59	llama-3.1-405B-AWQ	-	-	8	32

60	qwen2-57b-a14b	-	-	2	16
61	deepseek-v2-lite-16b	2	4	1	4
62	deepseek-v2-236b	-	-	8	4
63	qwen2-vl-7B	2	64	1	64
64	qwen-vl	1	64	1	64
65	qwen-vl-chat	1	64	1	64
66	MiniCPM-v2	2	16	1	16

“-”表示不支持。

4.18.9 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`,

解决方法：将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

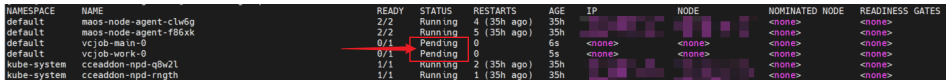
  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.18.10 附录：工作负载 Pod 异常问题和解决方法

Pod 状态为 Pending

当Pod状态长时间为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。

图 4-332 pod 状态 pending



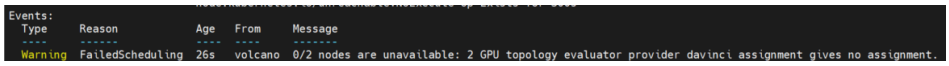
通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。

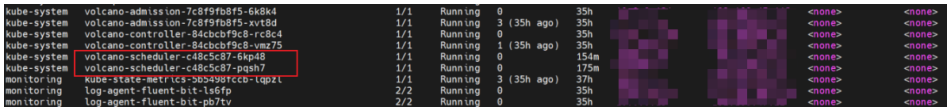
图 4-333 volcano 资源争抢



解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```

   图 4-334 scheduler

3. 如果重启后，还是会Pending，建议多重复重启几次。

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

4.19 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.910）

4.19.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**表4-167**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。
- Lite Server驱动版本要求23.0.6
- PyTorch版本：2.1.0
- 确保容器可以访问公网。

文档更新内容

6.3.910版本相对于6.3.909版本新增如下内容：

- 文档中新增对Qwen2.5的适配（包括0.5B、7B, 14B, 32B, and 72B），支持sft、lora、预训练。
- 文档中新增对Llama3.2的适配（包括1B和3B），支持sft、lora、预训练。
- 代码中ModelLink、MindSpeed已升级到最新版本，Python三方依赖版本已升级，其中：
  - MindSpeed的版本升级到commitID=4ea42a23
  - ModelLink的版本升级到commitID=8f50777
  - transformers版本升级到4.45.0
  - peft版本升级到0.12.0

训练支持的模型列表

本方案支持以下模型的训练，如**表4-165**所示。

表 4-165 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLM v3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan 2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-335 操作流程图

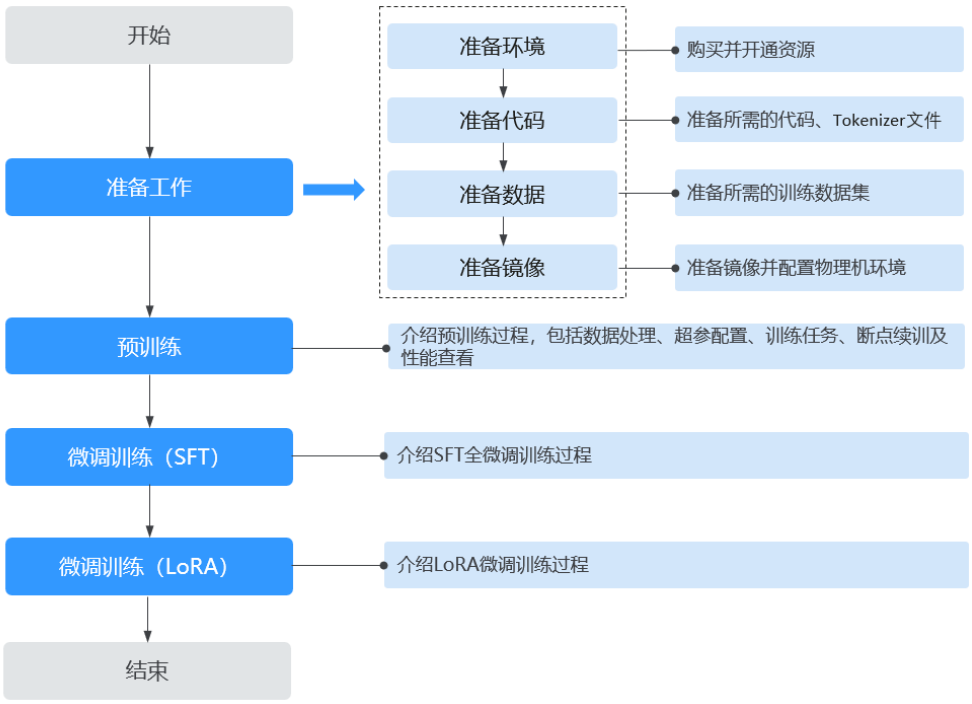


表 4-166 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.19.2 准备工作

4.19.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-175**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.19.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-167**所示，模型列表、对应的开源权重获取地址如**表4-168**所示。

表 4-167 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

表 4-168 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了git clone repo_url 的方式下载，但是不支持断点续传，并且clone会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
   ├──llm_train                # 模型训练代码包
      ├──AscendSpeed           # 基于AscendSpeed的训练代码
         ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         ├──scripts/             # 训练需要的启动脚本
            ├──llama2            # llama2系列模型执行脚本的文件夹
            ├──llama3            # llama3系列模型执行脚本的文件夹
            ├──qwen              # Qwen系列模型执行脚本的文件夹
            ├──qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
            ├── ...
            ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
            ├── install.sh       # 环境部署脚本
         ├──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
```

```
|──llm_inference                  # 推理代码包
|──llm_tools                     # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |─── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizers                   #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                       #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                 #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

   说明

   多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.19.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他
  ```

```
人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

如果用户希望将MOSS数据集的Excel格式转换为.json格式。可使用代码中提供的 **scripts/tools/ExcelToJson.p** 工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

  ```
  1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
  2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
  (随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
  3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
  (随机选择3个数据作为测试集)
  ```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败False。

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.19.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-169 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b

表 4-170 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
```

```
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```

⚠ 注意

- 在执行 scripts/install.sh 安装命令时，需要确认机器是否已连通网络。若无法连通网络，可使用离线安装的方式，具体参考**离线训练安装包准备说明**。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 脚本中的 transformers 的版本。由默认 transformers==4.45.0 修改为：transformers==4.44.2
- 为了避免因使用不同版本的 transformers 库进行训练和推理而导致冲突的问题，建议用户分别为训练和推理过程创建独立的容器环境。

通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/            # 训练需要的启动脚本
  |——src/                # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/        # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/          # MindSpeed昇腾大模型加速库
  |——ModelLink/          # ModelLink端到端的大语言模型方案
      |——megatron/      # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

如果git下载代码时报错，请参见**Git下载代码时报错**解决。

4.19.3 执行预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-171**所示。其他超参均有默认值，可以参考**表4-174**按照实际需求修改。

表 4-171 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**步骤二 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
<NNODES=4> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
```

- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。

```
示例:
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第三台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第四台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  示例:
  sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
  ```
  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_pretrain_7b.sh
  ```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-336 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

4.19.4 执行 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和**0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-171所示。其他超参均有默认值，可以参考表4-174按照实际需求修改。

表 4-172 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例：
  ```
  #第一台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  示例:
  sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
  ```
  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_sft_7b.sh
  ```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.19.5 执行 LoRA 微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-171**所示。其他超参均有默认值，可以参考**表4-174**按照实际需求修改。

表 4-173 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如表4-175所示。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:
  ```
  #第一台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例:
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
```

如果单机运行需要指定使用NPU卡的数量，可提前定义变量 NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令。

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/
llama2/0_pl_lora_7b.sh
```

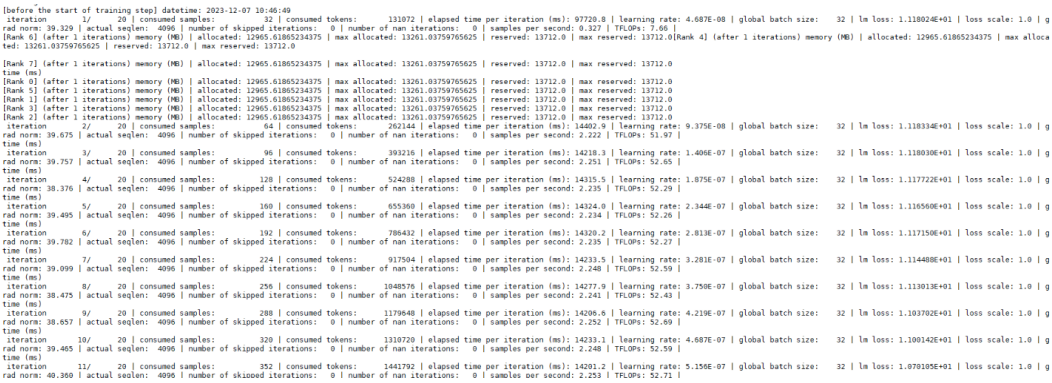
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

4.19.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-337 打印训练日志



```
[before the start of training step] datetime: 2023-12-07 10:46:49
 iteration        1/      20 | consumed samples:           32 | consumed tokens:       131072 | elapsed time per iteration (ms): 97720.8 | learning rate: 4.687E-08 | global batch size:    32 | lm loss: 1.118024E+01 | loss scale: 1.0 | g
rad norm: 39.329 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 0.327 | TFLOPs: 7.66 |
[Rank 6] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0[Rank 4] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max alloca
ted: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0

[Rank 7] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
time (ms)
[Rank 0] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 5] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 1] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 3] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 2] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
 iteration        2/      20 | consumed samples:           64 | consumed tokens:       262144 | elapsed time per iteration (ms): 14402.9 | learning rate: 9.375E-08 | global batch size:    32 | lm loss: 1.118334E+01 | loss scale: 1.0 | g
rad norm: 39.675 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.222 | TFLOPs: 51.97 |
time (ms)
 iteration        3/      20 | consumed samples:           96 | consumed tokens:       393216 | elapsed time per iteration (ms): 14218.3 | learning rate: 1.406E-07 | global batch size:    32 | lm loss: 1.118030E+01 | loss scale: 1.0 | g
rad norm: 39.757 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.251 | TFLOPs: 52.65 |
time (ms)
 iteration        4/      20 | consumed samples:          128 | consumed tokens:       524288 | elapsed time per iteration (ms): 14315.5 | learning rate: 1.875E-07 | global batch size:    32 | lm loss: 1.117722E+01 | loss scale: 1.0 | g
rad norm: 38.376 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.29 |
time (ms)
 iteration        5/      20 | consumed samples:          160 | consumed tokens:       655360 | elapsed time per iteration (ms): 14324.0 | learning rate: 2.344E-07 | global batch size:    32 | lm loss: 1.116560E+01 | loss scale: 1.0 | g
rad norm: 39.495 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.234 | TFLOPs: 52.26 |
time (ms)
 iteration        6/      20 | consumed samples:          192 | consumed tokens:       786432 | elapsed time per iteration (ms): 14320.2 | learning rate: 2.813E-07 | global batch size:    32 | lm loss: 1.117150E+01 | loss scale: 1.0 | g
rad norm: 39.782 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.27 |
time (ms)
 iteration        7/      20 | consumed samples:          224 | consumed tokens:       917504 | elapsed time per iteration (ms): 14233.5 | learning rate: 3.281E-07 | global batch size:    32 | lm loss: 1.114488E+01 | loss scale: 1.0 | g
rad norm: 39.099 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration        8/      20 | consumed samples:          256 | consumed tokens:      1048576 | elapsed time per iteration (ms): 14277.9 | learning rate: 3.750E-07 | global batch size:    32 | lm loss: 1.113013E+01 | loss scale: 1.0 | g
rad norm: 38.475 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.241 | TFLOPs: 52.43 |
time (ms)
 iteration        9/      20 | consumed samples:          288 | consumed tokens:      1179648 | elapsed time per iteration (ms): 14206.6 | learning rate: 4.219E-07 | global batch size:    32 | lm loss: 1.103702E+01 | loss scale: 1.0 | g
rad norm: 38.657 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.252 | TFLOPs: 52.69 |
time (ms)
 iteration       10/      20 | consumed samples:          320 | consumed tokens:      1310720 | elapsed time per iteration (ms): 14233.1 | learning rate: 4.687E-07 | global batch size:    32 | lm loss: 1.100142E+01 | loss scale: 1.0 | g
rad norm: 39.465 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration       11/      20 | consumed samples:          352 | consumed tokens:      1441792 | elapsed time per iteration (ms): 14201.2 | learning rate: 5.156E-07 | global batch size:    32 | lm loss: 1.070105E+01 | loss scale: 1.0 | g
rad norm: 40.360 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.253 | TFLOPs: 52.71 |
```

训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

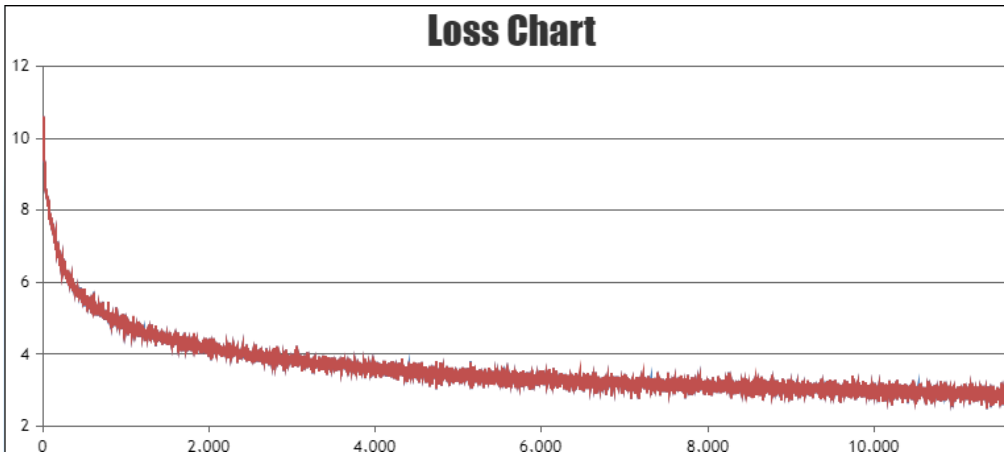
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-174**。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-338**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-338 Loss 收敛情况（示意图）



4.19.7 训练脚本说明参考

4.19.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以下参数取值主要以**llama2-70b预训练**为例，请根据实际模型修改。

表 4-174 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。

参数	示例值	参数说明
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-175所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-175 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
4	llama3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.19.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。

- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

- 先调用self.get_tokenized_data()对数据集进行encode
- self.get_tokenized_data()中调用self._filter方法处理每一个sample
- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

  - 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

  - 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

  - 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行
b. 循环处理其中的单轮对话
c. 在单轮对话中
    i. 对user和assistant的文本进行清洗
    ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
    iii. input_ids是user_ids和assistant_ids的拼接
    iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
d. attention_mask是和input_ids等长的全1序列
e. 返回input_ids\attention_mask\labels的字典
f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."
  ```

```
"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

  - 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-176 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.19.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

⚠ 注意

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-177 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.19.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-339所示。

图 4-339 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-340所示。

图 4-340 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297             """
298             # Load from model defaults
299             # assert self.padding_side == "left"
300
301             required_input = encoded_inputs[self.model_input_names[0]]
302             seq_length = len(required_input)
303
```

图 4-341 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-342所示。

图 4-342 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295
```

图 4-343 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-344所示。

图 4-344 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.19.7.5 离线训练安装包准备说明

申请的模型软件包一般依赖连通网络的环境。若用户的机器或资源池无法连通网络，并无法git clone下载代码、安装python依赖包的情况下，用户则需要找到已联网的机器（本章节以Linux系统机器为例）提前下载资源，以实现离线安装。用户可遵循以下步骤操作。

步骤一：资源下载

1. Python依赖包下载：进入 **scripts/install.sh** 文件中，找到需要安装的pip文件，如下列所示。直接下载pip文件，**注意：**下载要求的版本。

   ```
   pip install numpy==1.22.0 \
               transformers_stream_generator==0.0.5 \
               ...
   ```

2. 代码下载：访问 **scripts/install.sh** 文件中，找到需要git clone的文件，如下列所示。运行git clone命令，并git checkout切换到指定的版本。**注意：针对**Megatron-LM下载完成后，需要将megatron文件夹复制至ModelLink中。

   ```
   git clone https://gitee.com/ascend/ModelLink.git
   cd ModelLink
   git checkout 8f50777
   cd ..

   git clone https://gitee.com/lmzwhu/Megatron-LM.git
   cd Megatron-LM
   git checkout -f core_r0.6.0
   cp -r megatron ../ModelLink/
   cd ..

   git clone https://gitee.com/ascend/MindSpeed.git
   ```

```
cd MindSpeed
git checkout 4ea42a23
cd ..
```

完整的源码目录结构如下：

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/           # MindSpeed昇腾大模型加速库
       |——ModelLink/           # ModelLink端到端的大语言模型方案
            |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
            |——...
```

步骤二：资源安装

1. 将资源上传至机器中，确保容器能够访问，并进入已创建的容器。
2. Python依赖包本地安装：进入pip文件所在的路径，并运行安装命令。如下列所示。

```
pip install numpy
pip install transformers_stream_generator
...
```

3. 代码安装：访问 **scripts/install.sh** 文件，在最后执行的命令中需要分别进入ModelLink、MindSpeed、AscendSpeed目录，并运行以下命令。其中${INSTALL_DIR}为AscendSpeed所在路径。

```
cd  ${INSTALL_DIR}/ModelLink
pip install -e .
cd ${INSTALL_DIR}/MindSpeed
pip3 install -e .
cd ${INSTALL_DIR}
pip install -e .
```

4.19.8 常见错误原因和解决方法

4.19.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-175**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。

- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

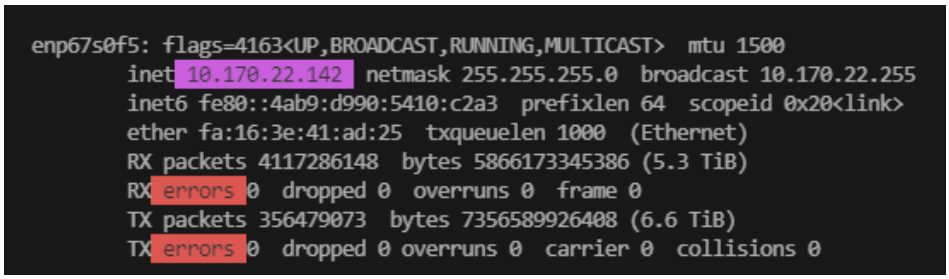
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.19.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-345 网卡名称错误



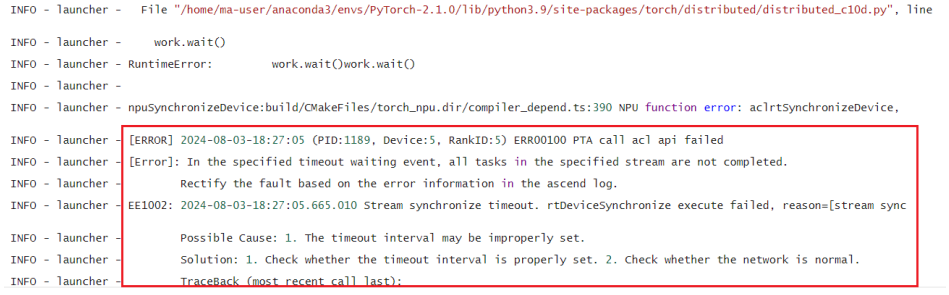

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.19.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-346 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.19.8.4 Git 下载代码时报错

在执行scripts/install.sh安装命令或使用Dockerfile构建镜像时，如遇到git下载代码出现以下类似的报错信息，关闭git验证即可。

报错信息：

```
fatal: unable to access 'https://gitee.com/ascend/ModelLink.git/': error setting certificate verify locations:
CAfile: /etc/pki/tls/certs/ca-bundle.crt CApath: none
```

关闭git验证命令如下：

```
git config --global http.sslverify false
```

4.20 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.910）

4.20.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的不同训练阶段方案，包括指令监督微调、DPO偏好训练、RM奖励模型训练、PPO强化训练方案。

- DPO(Direct Preference Optimization)：直接偏好优化方法，通过直接优化语言模型来实现对大模型输出的精确把控，不用进行强化学习，也可以准确判断和学习到使用者的偏好，最后，DPO算法还可以与其他优化算法相结合，进一步提高深度学习模型的性能。
- RM奖励模型(Reward Model)：是强化学习过程中一个关键的组成部分。它的主要任务是根据给定的输入和反馈来预测奖励值，从而指导学习算法的方向，帮助强化学习算法更有效地优化策略
- PPO强化学习(Proximal Policy Optimization)：是一种在强化学习中广泛使用的策略优化算法。它属于策略梯度方法的一种，旨在通过限制新策略和旧策略之间的差异来稳定训练过程。PPO通过引入一个称为“近端策略优化”的技巧来避免过大的策略更新，从而减少了训练过程中的不稳定性和样本复杂性。
- 指令监督式微调(Self-training Fine-tuning)：是一种利用有标签数据进行模型训练的方法。 它基于一个预先训练好的模型，通过调整模型的参数，使其能够更好地拟合特定任务的数据分布。 与从头开始训练模型相比，监督式微调能够充分利用预训练模型的知识和特征表示，从而加速训练过程并提高模型的性能。

训练阶段下有不同的训练策略，分为全参数训练、部分参数训练、LoRA、QLoRA，本文档主要支持全参数（Full）和LoRA、LoRA+。

- LoRA(Low-Rank Adaptation)： 这种策略主要针对如何在保持模型大部分参数固定的同时，通过引入少量可训练参数来调整模型以适应特定任务。
- LoRA+(Efficient Low Rank Adaptation of Large Models)：延续了 LoRA的精髓进一步提升了在复杂任务上对大模型进行微调的效率和性能，核心在于其独特的学习率比率（loraplus_lr_ratio）机制，适用于那些需要精确控制模型微调过程的场景，当前该策略仅支持qwen1.5-7B指令监督式微调。
- 全参训练（Full）： 这种策略主要对整个模型进行微调。这意味着在任务过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.910版本，请参考表4-180获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.RC3。
- Server驱动版本要求23.0.6
- PyTorch版本：2.2.0
- Python版本：3.10
- 确保容器可以访问公网。
- 仅支持313T、376T、400T

训练支持的模型列表

本方案支持以下模型的训练，如表4-178所示。

表 4-178 支持的模型列表及权重文件地址

支持模型	支持模型参数量	权重文件获取地址
Llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
	llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
	llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-chat-hf
Llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
	llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
Llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct/tree/main
	llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct/tree/main
Qwen1.5	qwen1.5-0.5b	https://huggingface.co/Qwen/Qwen1.5-0.5B
	qwen1.5-1.8b	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
	qwen1.5-4b	https://huggingface.co/Qwen/Qwen1.5-4B

支持模型	支持模型参数量	权重文件获取地址
	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
	qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
	yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
	qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
	qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
	qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
Qwen2_VL（支持多模态数据集）	qwen2_vl-2b	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main
	qwen2_vl-7b	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main
Falcon2	falcon-11B	https://huggingface.co/tiiuae/falcon-11B
GLM-4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本 4b556ad4d70c38924cb8c120adbf21a0012de6ce
Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
	qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
	qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
	qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
	qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct

支持模型	支持模型参数量	权重文件获取地址
	llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

表 4-179 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
训练	启动训练	介绍各个训练阶段：指令微调、PPO强化训练、RM奖励模型、DPO偏好训练使用全参/lora训练策略进行训练任务、性能查看。

4.20.2 准备工作

4.20.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-185**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.20.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-180**所示，模型列表、对应的开源权重获取地址如**表4-178**所示。

表 4-180 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                        # 模型训练代码包
      |——LLaMAFactory                  # 基于LLaMAFactory的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——demo.yaml            # 样例yaml配置文件
         |——demo.sh              # 指令微调启动shell脚本
         |——intall.sh            # 需要的依赖包
         |——LLaMA-Factory        # LLaMAFactory的代码目录
      |——AscendSpeed              # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── LLaMAFactory                # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──demo.sh              # 指令微调启动shell脚本
        |──demo.yaml            # 样例yaml配置文件
        |──intall.sh            # 需要的依赖包
        |──LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考步骤三 启动容器镜像
            |──data             # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|── tokenizers                   #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
    |── Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表4-183表格中output_dir参数值
|── saved_dir_for_output_lf          # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |── ${model_name}               # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen2-72B
   ```

4.20.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-181 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_2_ascend:pytorch_2.2.0-cann_8.0.rc3-py_3.10-hce_2.0.2406-aarch64-snt9b-20240910150953-6faa0ed

表 4-182 模型镜像版本

模型	版本
CANN	cann_8.0.RC3
驱动	23.0.6
PyTorch	2.2.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取基础镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --cpus 192 \
   ```

```
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如：  chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

4.20.2.4 DockerFile 构建镜像（可选）

本章节主要介绍通过DockerFile文件构建训练镜像，将训练过程中依赖包封装使用，过程中需要连接互联网git clone，请确保环境可以访问公网，详解操作如下：

进入代码包Dockerfile文件同级目录：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory
```

构建新镜像：

```
docker build -t <镜像名称>:<版本名称>  .
```

如无法访问公网则需配置代理，增加`--build-arg`参数指定代理地址确保访问公网。

```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx"
--network=host -t <镜像名称>:<版本名称>  .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_2_ascend:20241106

⚠ 注意

构建镜像前需保证Dockerfile文件内容中镜像名与本文档**镜像**保持一致，如不同则需修改为一致。

```
# 修改以下内容：
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/xxx
```

4.20.2.5 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前支持alpaca格式和sharegpt格式的微调数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加数据集描述；具体示例如下。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。
   ```
   cd  /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
   ```
2. 将自定义原始数据（指令监督微调样例数据集：alpaca_gpt4_data.json）按照下面的数据存放目录要求放置。

   说明

   指令微调样例数据集alpaca_gpt4_data.json的下载链接：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**

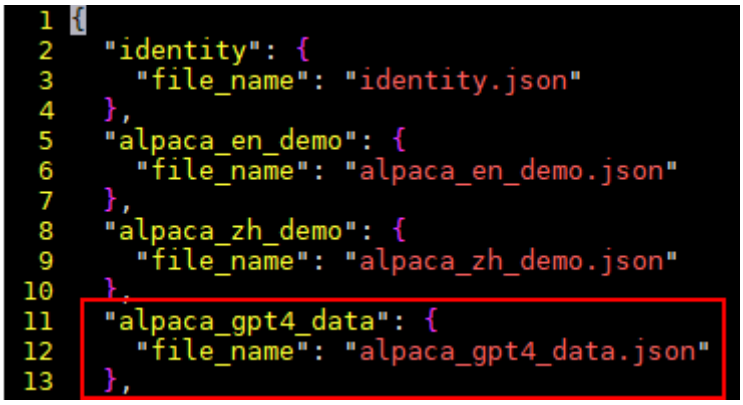
   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws/llm_train ）
    |── LLaMAFactory/data
        |── alpaca_en_demo.json                # 代码原有数据集
        |── identity.json                      # 代码原有数据集
        ...
        |── alpaca_gpt4_data.json              # 自定义数据集
   ```
3. 更新代码目录下data/dataset_info.json文件。如使用以下示例数据集则命令如下。关于数据集文件格式及配置，更多样例格式信息请参考**data/README_zh.md** 的内容。
   ```
   vim dataset_info.json
   ```

   新加配置参数如下：

   ```
   "alpaca_gpt4_data": {
     "file_name": "alpaca_gpt4_data.json"
   },
   ```

   样例截图：



4.20.3 执行训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

步骤二 修改训练 yaml 文件配置

LlamaFactroy配置文件为Yaml文件，启动训练前需修改Yaml配置文件，Yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示。

1. 选择训练阶段类型。
   - 指令监督微调,复制**tune_yaml样例模板**内容覆盖demo.yaml文件内容。
   - DPO偏好训练，复制**dpo_yaml样例模板**内容覆盖demo.yaml文件内容。
   - PPO强化训练，先进行**RM奖励训练**任务后，复制**ppo_yaml样例模板**内容覆盖demo.yaml内容。
   - RM奖励训练，复制**rm_yaml样例模板**内容覆盖demo.yaml文件内容。

   说明

   1、DPO偏好训练、Reward奖励模型训练、PPO强化学习目前仅限制支持于llama3系列

   2、PPO训练暂不支持 ZeRO-3存在通信问题，如llama3-70B使用ZeRO-3暂不支持

2. 训练策略类型
   - 全参full，配置如下：
     ```
     finetuning_type: full
     ```
   - lora，如dpo仅支持此策略；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     ```
   - lora+，目前仅支持qwen1.5-7B**指令监督微调**；配置如下：
     ```
     finetuning_type: lora
     lora_target: all
     loraplus_lr_ratio: 16.0
     ```
3. 修改**yaml文件**(demo.yaml)的参数如**表4-183**所示。

表 4-183 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-185**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	**可修改**。指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配。取值可参考**表4-185**中**梯度累积值**列。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	**【可选】**注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	**【可选】**dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

4. 是否选择加速深度学习训练框架Deepspeed，可参考**表4-185**选择不同的框架。
   - 是，选用ZeRO (Zero Redundancy Optimizer)优化器。
     - ZeRO-0，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z0_config.json
       ```

- ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
  ```
  deepspeed: examples/deepspeed/ds_z1_config.json
  ```
- ZeRO-2，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z2_config.json
  ```
- ZeRO-2-Offload，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z2_offload_config.json
  ```
- ZeRO-3，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z3_config.json
  ```
- ZeRO-3-Offload，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z3_offload_config.json
  ```

– 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。

5. 是否开启NPU FlashAttention融合算子，具体约束详见**NPU_Flash_Attn融合算子约束**
   – 是，配置以下参数。
   ```
   flash_attn: sdpa
   ```
   – 否，配置以下参数关闭。
   ```
   flash_attn: disabled
   ```
6. 是否使用固定句长。
   – 是，配置以下参数
   ```
   packing: true
   ```
   – 否，默认使用动态句长，**注释**掉packing参数。
7. 选用数据精度格式bf16或fp16二者选一，两者区别可查看**BF16和FP16说明**。
   – bf16，配置以下参数。
   ```
   bf16: true
   ```
   – fp16，相比bf16还需配置loss scale参数，配置如下。
   - 设置fp16为True。
     ```
     fp16: true
     ```
   - 修改deepspeed的"loss_scale"参数，配置如下。
     ○ 修改**ZeRO优化器**配置文件，如ZeRO2命令如下。
     ```
     cd /home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
     vim ds_z2_config.json
     ```
     ○ 使用fp16容易出现数值溢出，因此配置loss scale建议配置4096或4096以上：
     ```
     "loss_scale": 4096,
     ```
8. 是否使用自定义数据集。
   – 是，参考**准备数据（可选）**，以指令监督微调数据集为例，配置以下参数：参考**表4-183**dataset_dir和dataset参数说明；如alpaca_gpt4_data.json数据集前缀则为alpaca_gpt4_data。
   ```
   dataset: alpaca_gpt4_data
   dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
   ```
   – 否，使用代码包自带数据集，注释掉dataset_dir参数，配置参数如下。
   - 指令监督微调/PPO数据集
     ```
     dataset: identity,alpaca_en_demo
     ```

- 多模态数据集，如qwen2_vl系列模型
  dataset: mllm_demo,identity
- RM/DPO，目前仅支持llama3系列模型
  dataset: dpo_en_demo

9. 是否使用falcon-11b、qwen2_vl系列、glm4-9b模型。
   - 是，更新配置或命令。
     - falcon-11b，参考**falcon-11B模型**替换文件。
     - glm4-9b，参考**glm4-9b模型**修改文件内容。
     - qwen2_vl系列，数据集为**多模态数据集**，若前面步骤已配置请忽略。具体配置如下：
       数据集dataset配置：
       dataset: mllm_demo,identity
   - 否，忽略此步骤，执行下一步。
10. 如需其他配置参数，可参考**表4-184**按照实际需求修改。

步骤三 启动训练脚本

修改完yaml配置文件后，启动训练脚本。模型不同最少NPU卡数不同，NPU卡数建议值可参考**表4-185**。

1. **修改启动脚本demo.sh**

   进入代码目录{work_dir}/llm_train/LLaMAFactory下**修改启动脚本**，其中{work_dir}为容器挂载路径

   ①是否为PPO强化训练；

   - 是，demo.sh添加变量；
     export PYTORCH_NPU_ALLOC_CONF = expandable_segments:False
   - 否，忽略此步骤，执行下一步；

   ②修改yaml文件路径：修改demo.sh最后一行代码，将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，例如将以下命令

   - 修改前

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
   ```

   - 修改后：

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
   ```

2. **执行多机启动命令**（可选）

   多台机器执行训练启动命令如下。

   ```
   多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
   示例:
   #第一台节点
   sh demo.sh  xx.xx.xx.xx  4  0
   # 第二台节点
   sh demo.sh  xx.xx.xx.xx  4  1
   # 第三台节点
   sh demo.sh  xx.xx.xx.xx  4  2
   # 第四台节点
   sh demo.sh  xx.xx.xx.xx  4  3
   ```

   以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

3. **执行单机启动命令**（可选）

   一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

   进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

   ```
   # 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
   sh demo.sh localhost 1 0
   ```

   单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
   ```

   其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务。

训练成功标志

“***** train metrics *****”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：训练常见问题**解决。

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：训练常见问题**问题3小节。

3、大模型参数如（qwen2-72B、llama2-70B）等sft训练完成后多线程退出时报“torch.distributed.DistStoreError: Socket Timeout”时请参考**问题4：Error waiting on exit barrier错误**

4、需要开启profiling功能进行性能数据采集和解析请参考**录制Profiling**

5、训练过程中报"ModuleNotFoundError: No module named 'tyro'"可参考**依赖包tyro错误:"ModuleNotFoundError..."**

4.20.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-347 打印训练日志

```
  0%|              | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|▏             | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▎             | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍             | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌             | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋             | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊             | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉             | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█             | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏            | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▍            | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▌            | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▋            | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|█▊            | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表4-183**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

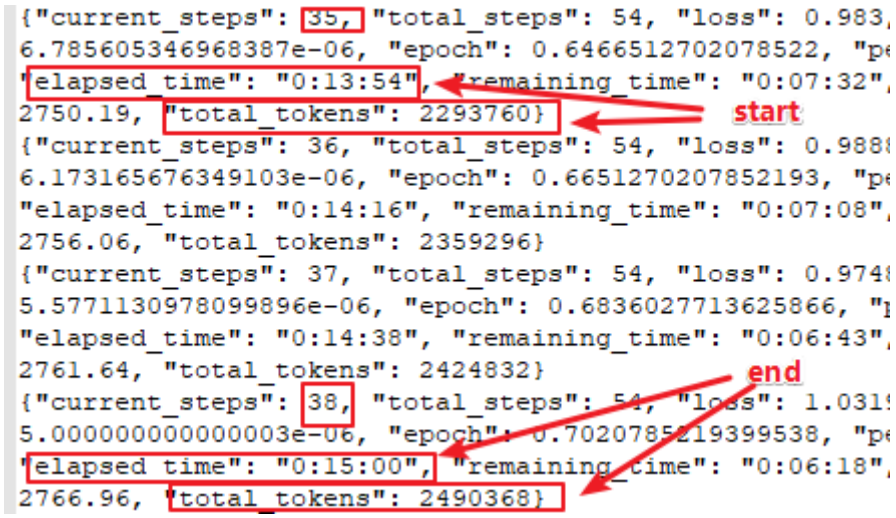
- 吞吐量（tokens/s/p）：可通过**表4-183**表格中**output_dir**参数值路径下的**trainer_log.jsonl**计算性能。取中间过程多steps平均值吞吐计算公式为：

  delta_tokens = end_total_tokens-start_ total_tokens

  delta_time = end_elapsed_time - start_elapsed_time

  吞吐值(tps) = delta_tokens / delta_time / 训练卡数

  如图所示：

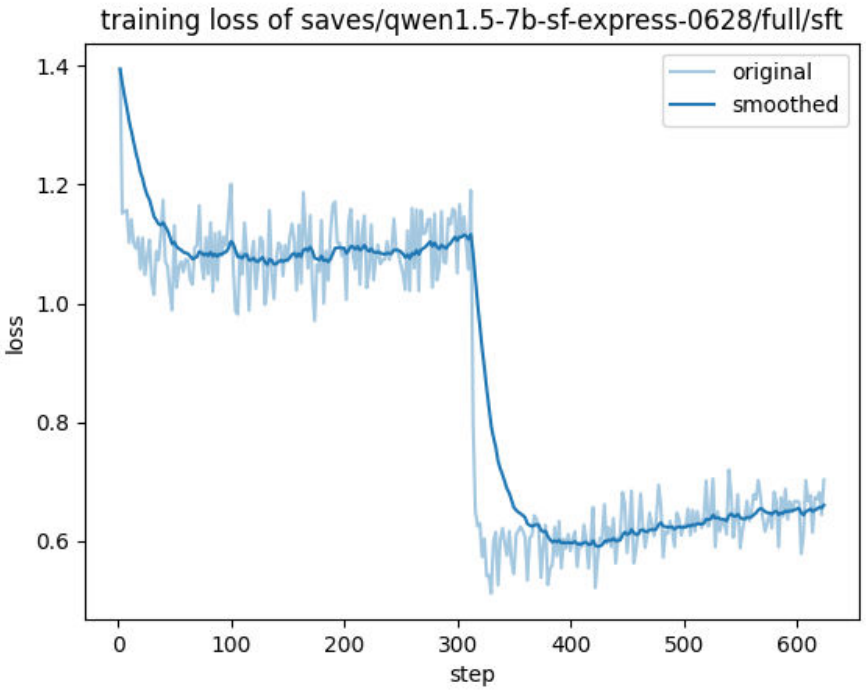


- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-183**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，将**trainer_log.jsonl**文件长传至可视化工具页面，如**图4-348**所示。

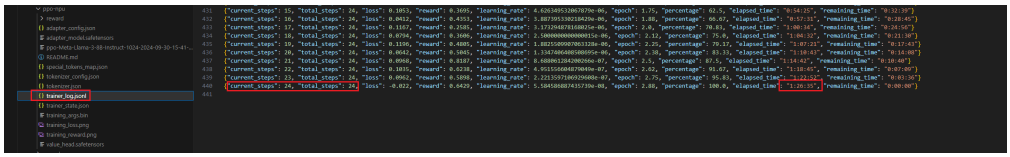
  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

  图 4-348 Loss 收敛情况（示意图）



注：ppo训练结束不会打印性能。建议根据保存路径下的trainer_log.jsonl文件的最后一行总的训练steps和时间来判断性能



4.20.5 训练 benchmark 工具

4.20.5.1 工具介绍及准备工作

本章节主要介绍针对LLaMAFactory开发的测试工具benchmark，支持训练、性能对比、下游任务评测、loss和下游任务对比能力。对比结果以excel文件呈现。方便用户验证发布模型的质量。所有配置都通过yaml文件设置，用户查看默认yaml文件即可知道最优性能的配置。

说明

目前仅支持SFT指令监督微调训练阶段。

代码目录

benchmark工具脚本存放在**代码包**AscendCloud-LLM-*xxx*.zip的LLM/LLaMAFactory/benchmark目录下，包含训练性能测试和训练精度测试脚本。

代码目录如下:

```
benchmark
├── config      #  默认的配置，使用前根据实际情况修改数据集路径dataset_dir、权重路径model_name_or_path
  ├── deepspeed                    # zero配置文件目录
  ├── accuracy_cfgs.yaml             # 精度测试yaml配置文件
  ├── performance_cfgs.yaml          # 性能测试yaml配置文件
  ├── llama_factory_performance_baseline.yaml # LlamaFactory基于GP-Ant8训练性能基线值
  ├── llama_factory_accuracy_baseline.yaml  # LlamaFactory基于GP-Ant8训练精度基线值
  ......
  ├── ......
├── src          # 工具代码目录
  ├── accuracy.py        #精度测试脚本
  ├── common_utils.py    #获取训练日志工具
  ├── performance.py     #性能测试脚本
  ├── trainer.py         #训练启动脚本
├── data.tgz             # 样例数据
├── setup.py             # 构建工具包
```

约束限制

目前仅支持以下模型：

qwen2.5-7b

qwen2-7b

qwen1.5-7b

llama3.2-3b

llama3.1-8b

llama3-8b

llama2-7b

yi-6b

准备工作

1. 完成**准备工作**内容，生成benchmark-cli工具。
2. 解压版本包data.tgz：测试样例数据；比如工作目录为：/homa/ma-user/LLaMAFactory
   ```
   # 将默认数据解压config同级目录
   tar -zxvf ./benchmark/data.tgz ./benchmark/
   ```
3. 创建test-benchmark目录，该目录存放训练生成的权重文件及训练日志。
   ```
   # 任意目录创建
   mkdir test-benchmark
   ```
4. 修改yaml文件参数中model_name_or_path、dataset_dir和dataset或eval_dataset参数配置，修改**代码目录**下accuracy_cfgs.yaml或performance_cfgs.yaml文件内容，参数详解可参考**表4-183**。
   ```
   # 默认参数；根据自己实际要求修改
   ## accuracy_cfgs.yaml、performance_cfgs.yaml
   dataset_dir: /xxxx/benchmark/data/dataset
   dataset: gsm8k_train_alpaca
   model_name_or_path: /data/wulan1/model/qwen2.5-7b
   ## accuracy_cfgs.yaml
   eval_dataset: gsm8k_test
   ```
   样例yaml配置文件结构分为
   - base块：基础配置块
   - ModelName块：该模型所需配置的参数，如qwen2.5-7b块

   样例截图如下：

```
.base: &base
  ### model
  ### method
  do_train: true
  stage: sft
  lora_target: all
  ### dataset
  dataset_dir: /data/wulan1/user/lmz/third-party-large-mode-labraries/LLM/L
  dataset: gsm8k_train_alpaca
  cutoff_len: 4096
  packing: true
  max_samples: 10240
  overwrite_cache: true
  preprocessing_num_workers: 16
  ### output
  logging_steps: 1
  save_steps: 5000
  plot_loss: true
  overwrite_output_dir: true
  include_tokens_per_second: true
  include_num_input_tokens_seen: true
  ### train
  per_device_train_batch_size: 1
  gradient_accumulation_steps: 8
  learning_rate: 1.0e-5
  num_train_epochs: 1.0
  lr_scheduler_type: cosine
  warmup_ratio: 0.1
  bf16: true
  ddp_timeout: 180000000
  flash_attn: sdpa

qwen2.5-7b:
  _base: &qwen2_5-7b
    <<: *base
    model_name_or_path: /data/wulan1/model/qwen2.5-7b
    template: qwen
  lora:
    <<: *qwen2_5-7b
    finetuning_type: lora
    output_dir: ./saves/qwen2.5-7b/sft_lora
    disable_gradient_checkpointing: true
  full:
```

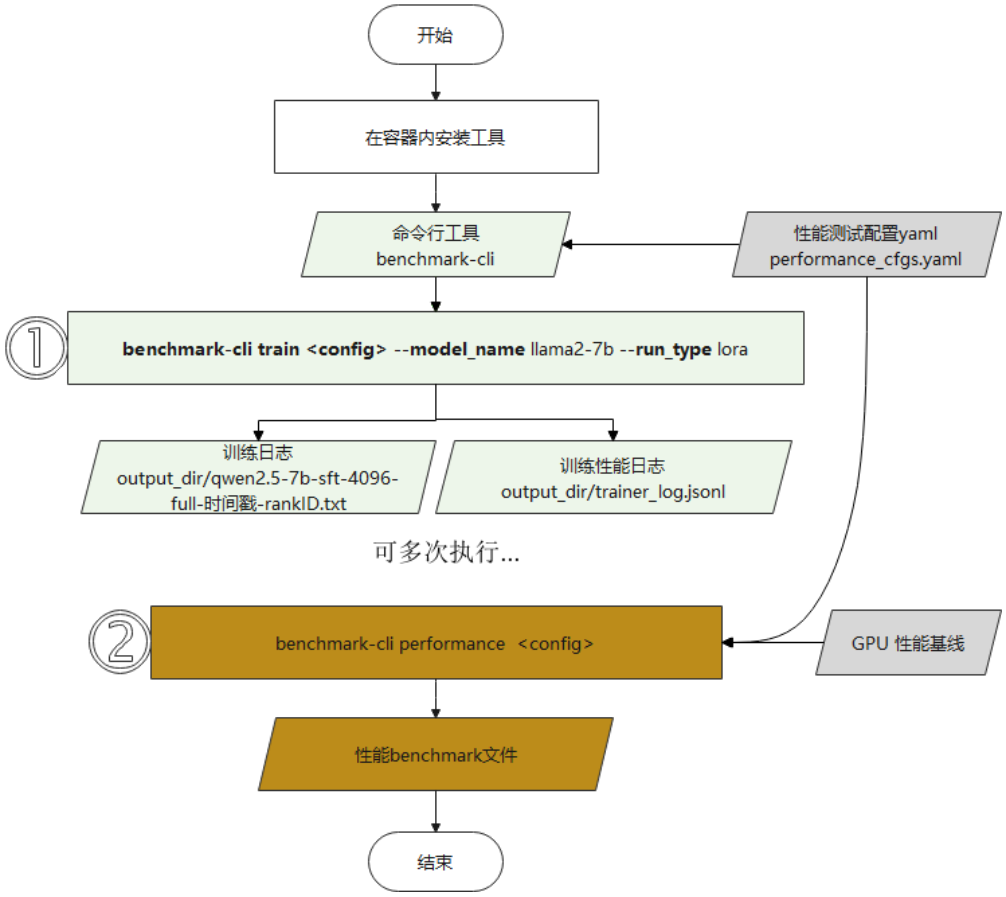
5. 开始训练测试，具体步骤参考**训练性能测试**或**训练精度测试**，根据实际情况决定。

4.20.5.2 训练性能测试

流程图

训练性能测试流程图如下图所示：

图 4-349 训练性能测试流程



执行训练任务

1. 进入**test-benchmark目录**执行训练命令，可以多次执行，卡数及其它配置参考**NPU卡数取值表**按自己实际情况决定。

   单机<可选>：

   ```
   # 默认8卡
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   # 指定设备卡数，如2卡
   ASCEND_RT_VISIBLE_DEVICES=0,1 benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   ```

   多机<可选>多机同时执行：

   ```
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type> --master_addr <master_addr> --num_nodes <nodes> --rank <rank>
   ```

   - <cfgs_yaml_file>：性能测试配置的yaml文件地址，如**代码目录**中performance_cfgs.yaml相对或绝对路径。
   - <model_name>：训练模型名，如qwen2-7b
   - <run_type>：训练策略类型及数据序列长度：【lora：4096-lora、full：4096-full、lora-8k：8192-lora、full-8k：8192-full】
   - --master_addr <master_addr>：主master节点IP，一般选rank0为主master。
   - --num_nodes <nodes>：训练节点总个数
   - --rank <rank>：节点ID

2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志，如qwen2.5-7b日志。

- qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt，打印吞吐值及训练参数

```
10/28/2024 16:54:54 - INFO - trainer - pkill -9 npu-smi
10/28/2024 16:54:58 - INFO - trainer - +++++++++ 性能: 3855.0588235294117 tokens/s/p +++++++++

10/28/2024 16:47:46 - INFO - trainer - {'do_train': True, 'stage': 'sft', 'lora_target':
'/data/wulan1/user/lmz/third-party-large-mode-labraries/LLM/LLaMAFactory/benchmark/data/
'overwrite_cache': True, 'preprocessing_num_workers': 16, 'logging_steps': 1, 'save_steps
'include_num_input_tokens_seen': True, 'per_device_train_batch_size': 1, 'gradient_accum
0.1, 'bf16': True, 'ddp_timeout': 180000000, 'flash_attn': 'sdpa', 'model_name_or_path':
'./saves/qwen2.5-7b/sft_lora', 'disable_gradient_checkpointing': True}
```

- qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt，打印训练过程中AICORE利用率

NpuID(Idx)	ChipId(Idx)	Pwr(W)	Temp(C)	AI Core(%)	AI Cpu(%)	Ctrl Cpu(%)	Memory(%)	Memory BW(%)
0	0	93.3	52	0	0	1	5	0
1	0	99.8	50	0	0	2	5	0
2	0	92.8	49	0	0	2	5	0
3	0	93.8	54	0	0	1	5	0
4	0	95.5	51	0	0	0	5	0

执行性能比较脚本

进入test-benchmark目录执行命令:

```
benchmark-cli performance <cfgs_yaml_file> --baseline <baseline> --o <output_dir>
```

- <cfgs_yaml_file>：性能测试配置的yaml文件地址，指**代码目录**中performance_cfgs.yaml相对或绝对路径，此配置文件为训练最优配置参数。
- --baseline <baseline>：<可选>GP-Ant8机器性能基线yaml文件路径，用户可自行修改，不填则使用工具自带基线配置，默认基线配置样例如下：

```
# 性能依次是 A800, ModelLink 313T, ModelLink 400T
qwen2.5-7b:
    full:    [2380, -1, -1]
    lora:    [3254, -1, -1]
    full-8k: [2394, -1, -1]
    lora-8k: [3199, -1, -1]
```

- --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径。

查看性能结果

任务完成之后会在test-benchmark目录下生成excel表格：

性能结果 LLaMAFactory_train_performance_benchmark_<版本号>_<时间戳>.xlsx

表格样例如下：

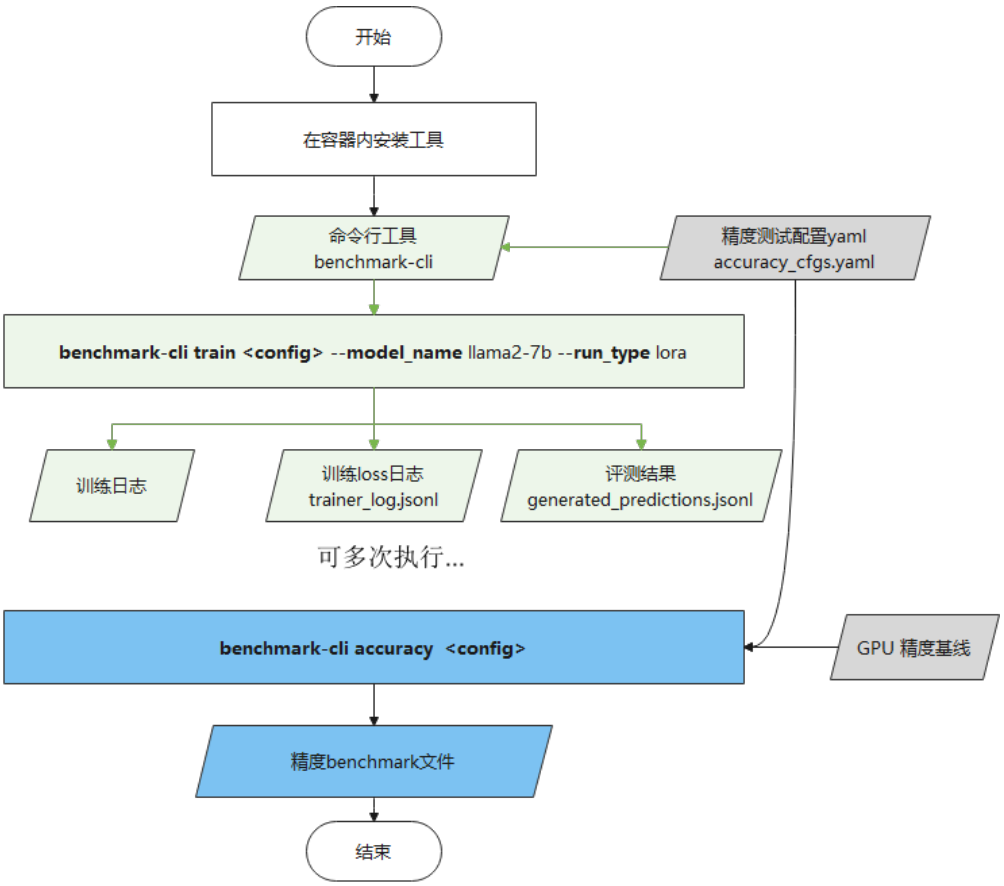
分类	训练框架	训练类型	序列长度	训练策略	ra_targ	加速框架	训练卡数	GBS	MBS	zero并行	FA	CORE 利用存	利用率值(tok	性能值(t	ory VS	能基线(to	vs GP-Ant8
qwen2.5-7	LLaMAFact	SFT	4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	43	51	743.9821	-	-	-	-
qwen2-7b	LLaMAFact	SFT	4096	lora	all	DeepSpeed	8	32	1	zero-0	sdpa	38.3	50	508.0854	-	-	2500	0.203234

4.20.5.3 训练精度测试

流程图

训练精度测试流程图如下图所示：

图 4-350 训练精度测试流程图



执行训练任务

1. 进入**test-benchmark目录**执行训练命令，可以多次执行，按自己实际情况。
   ```
   benchmark-cli train <cfgs_yaml_file> <model_name> <run_type>
   ```
   - <cfgs_yaml_file>：精度测试配置的yaml文件地址，指**代码目录**中accuracy_cfgs.yaml相对或绝对路径
   - <model_name>：训练模型名，如qwen2.5-7b
   - <run_type>：训练策略类型及数据序列长度：【lora：4096-lora、full：4096-full】
2. 训练完成后，test-benchmark目录下会生成训练日志及NPU利用率日志及权重文件，如qwen2.5-7b日志：
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-0.txt
   - qwen2.5-7b-sft-4096-lora-313T-20241028_164746-npu_info-0.txt

执行精度比较脚本

进入test-benchmark目录执行命令:

```
benchmark-cli accuracy <cfgs_yaml_file> --o <output_dir> --baseline <baseline>
```

- <cfgs_yaml_file>：精度测试配置的yaml文件地址，如**代码目录**中accuracy_cfgs.yaml相对或绝对路径
- --o <output_dir>： <可选>任务完成输出excel表格路径，默认为"./"当前所在路径

- --baseline <baseline>：<可选>GP-Ant8机器精度基线Yaml文件路径，不填则使用工具自带基线配置，默认基线配置样例如下：

```
qwen2-7b:
  full:
    loss: [0.4339,0.1433,0.2185,0.172,0.2376,0.2519,0.22
    score: 0.6815769522365428
  lora:
    loss: [0.4339,0.3791,0.3035,0.1714,0.1674,0.1443,0.1
    score: 0.7490523123578469
```

说明

客户使用工具自带精度基线Yaml则需使用accuracy_cfgs.yaml文件中默认配置，权重使用**表1 模型权重**中指定的Huggingface地址，数据指定data.tgz里面提供的gsm8k数据。

查看精度结果

任务完成之后会在test-benchmark目录下生成excel表格：

精度结果 LLaMAFactory_train_accuracy_benchmark_<版本号>_<时间戳>.xlsx

样例截图：

benchmark示例：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	warmup	学习率衰减策	数据集	MBS	GBS	平均相对	平均绝对	loss绝	推理评分差	NPU推理评分	GP-A推理评分
qwen2-7b	lora	233	adamw	0.0002	0	0	cosine	gsm8k_trs	1	32	0.418063	0.016463	0.0162	0.025018	0.728582	0.7536

具体精度benchmark包含如下字段：

分类	训练类型	训练总step	优化器	初始学习率	最小学习率	学习率warmup ratio	学习率衰减策略	数据集	MBS	GBS	loss平均相对误差	loss平均绝对误差	step1 loss绝对误差	推理评分差	NPU推理评分	GP-A推理评分

4.20.6 训练脚本说明

4.20.6.1 Yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-184 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。

参数	示例值	参数说明
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示当前的训练阶段。可选择值：sft、rm、ppo、dpo。 • sft代表指令监督微调； • rm代表奖励模型训练； • ppo代表PPO训练； • dpo代表DPO训练。
finetuning_type	full	用于指定微调策略类型，可选择值full、lora。 如果设置为full，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
lora_target	all	采取lora策略方法的目标模块，默认为all
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表4-185**中的**template列**

参数	示例值	参数说明
max_samples	50000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。
per_device_train_batch_size	1	指定每个设备的训练批次大小。
gradient_accumulation_steps	8	**必须修改**，指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可参考**表4-185**
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
save_steps	5000	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	2.0e-5	指定学习率

参数	示例值	参数说明
disable_gradient_checkpointing	true	关闭重计算，用于禁用梯度检查点，**默认开启**梯度检查点;在深度学习模型训练中用于保存模型的状态，以便在需要时恢复。这种技术可以帮助减少内存使用，特别是在训练大型模型时，但同时影响性能。True表示关闭重计算功能。
include_tokens_per_second include_num_input_tokens_seen	true	用于在训练过程中包含每秒处理的tokens和已经看到的输入tokens，方便计算性能。
reward_mode	/home/ma-user/ws/saves/rm/llama3-8b/lora	**PPO强化必修改**；指定Reward奖励任务完成时output_dir目录，PPO强化训练前提为完成Reward奖励学习；请根据实际规划修改。
loraplus_lr_ratio	16.0	lora+策略算法独有参数；设置Lora+算法的lambda值为16.0

tune_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true

# 全参
finetuning_type: full

# lora
# finetuning_type: lora
# lora_target: all

deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
dataloader_num_workers: 8
### output
output_dir: /home/ma-user/ws/saves/tune/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
```

```
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

dpo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: dpo
do_train: true

# lora
finetuning_type: lora
lora_target: all

pref_beta: 0.1
pref_loss: sigmoid
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: dpo_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
dataloader_num_workers: 8
### output
output_dir: /home/ma-user/ws/saves/dpo/llama3-8b/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 5.0e-6
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ppo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
reward_model: /home/ma-user/ws/saves/rm/llama3-8b/lora
### method
stage: ppo
do_train: true

# 全参
# finetuning_type: full
# reward_model_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
```

```
### dataset
dataset: identity,alpaca_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
dataloader_num_workers: 8
packing: true
### output
output_dir: /home/ma-user/ws/saves/ppo/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
### generate
max_new_tokens: 512
top_k: 0
top_p: 0.9
```

rm_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
### method
stage: rm
do_train: true

# 全参
# finetuning_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: dpo_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
dataloader_num_workers: 8
packing: true
### output
output_dir: /home/ma-user/ws/saves/rm/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
```

```
bf16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
  }
}
```

4.20.6.2 模型 NPU 卡数、梯度累积值取值表

不同模型推荐的训练参数和计算规格要求如表4-185所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推。

表 4-185 NPU 卡数、加速框架、梯度配置取值表

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
llama2	llama2	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		13B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3	llama3	70B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full		gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
		8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
llama3.1	llama3	8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3.2	llama3	1B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		3B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 2*Ascend
Qwen2	qwen	72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		0.5/1.5B	lora/full	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
Qwen2_vl	qwen2_vl	2B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 2*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-2-Offload	1*节点 & 8*Ascend
Qwen1.5	qwen	0.5/1.8B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	2*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
Qwen2.5	qwen	0.5B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		32B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	2*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
falcon2	falcon	11B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
GLM4	glm4	9B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
Yi	yi	6B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 8*Ascend
		34B	full	4096	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	8192	gradient_accumulation_steps: 8	ZeRO-3	8*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

说明

以上参数为开启NPU FlashAttention融合算子，上述参数值仅供参考，请根据自己实际要求合理配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器、NPU节点数及其他配置。

具体优化工具使用说明可参考**如何选择最佳性能的zero-stage和-offloads**。

4.20.6.3 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

glm4-9b 模型

在训练开始前，需要修改glm4-9b模型中的tokenizer文件modeling_chatglm.py内容，具体步骤如下：

- 进入到tokenizer目录下{work_dir}/tokenizers/glm4-9B/，命令如下：

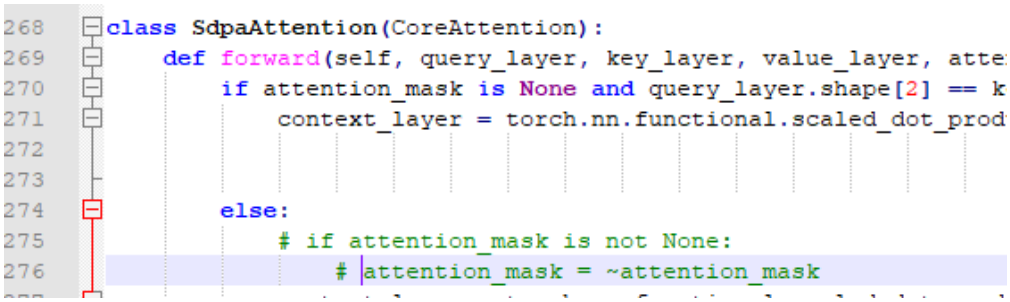
```
cd /home/ma-user/ws/tokenizers/glm4-9B
```

- 修改modeling_chatglm.py文件内容：

```
vim modeling_chatglm.py
# 注释掉以下两行内容
```

```
# if attention_mask is not None
    # attention_mask = ~attention_mask
```

样例图：



4.20.6.4 NPU_Flash_Attn 融合算子约束

1. query、key、value都需要梯度。默认开启重计算，则前向时qkv没有梯度，如果需要关闭重计算，可以在yaml配置 `disable_gradient_checkpointing: true` 关闭，但显存占用会直线上升。
2. attn_mask 只支持布尔（bool）数据类型，或者为None。
3. query的shape仅支持 [B, N1, S1, D]，其中N1≤ 2048，D≤ 512并且dim== 4。
4. 对于GQA，key的shape是 [B, N2, S2, D]，其中 N2 ≤ 2048，并且N1是N2的正整数倍。

不满足以上场景，则不能实现NPU_Flash_Attn功能。

4.20.6.5 BF16 和 FP16 说明

在大模型训练中，BF16（Brain Floating Point）和FP16（Float16）都是使用的半精度浮点数格式，但它们在结构和适用性上有一些重要的区别。

BF16：具有8个指数位和7个小数位。在处理大模型时有优势，能够避免在训练过程中数值的上溢或下溢，从而提供更好的稳定性和可靠性，在大模型训练和推理以及权重存储方面更受欢迎。

FP16：用于深度学习训练和推理过程中，可以加速计算并减少内存的占用，对模型准确性的影响在大多数情况下较小。与BF16相比在处理非常大或非常小的数值时遇到困难，导致数值的精度损失。

综上所述，BF16因其与FP32相似的数值范围和稳定性，在大模型训练中提供了优势。而FP16则在计算效率和内存使用方面有其独特的优点，但可能在数值范围和稳定性方面略逊一筹。因此，选择哪种格式往往取决于具体的应用场景和训练需求。

4.20.6.6 录制 Profiling

Ascend PyTorch Profiler是针对PyTorch框架开发的性能数据采集和解析工具，通过在PyTorch训练脚本中插入Ascend PyTorch Profiler接口，执行训练的同时采集性能数据，完成训练后直接输出可视化的性能数据文件，提升了性能分析效率。

Ascend PyTorch Profiler接口可全面采集PyTorch训练场景下的性能数据，主要包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等，可以全方位分析PyTorch训练时的性能状态。

录制命令如下：

在启动训练脚本基础：**步骤三 启动训练脚本** 新加DO_PROFILER=1和PROF_SAVE_PATH=/save_path参数，单机启动举例说明：

```
PROF_ENABLE=1 PROF_SAVE_PATH=/save_path sh demo.sh localhost 1 0
```

- PROF_SAVE_PATH：Profiling录制结果存放路径
- PROF_ENABLE：是否开启Profiling录制功能

4.20.7 附录：训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

1. 容器内执行以下命令，指定NPU内存分配策略的环境变量，开启动态内存分配，即在需要时动态分配内存，可以提高内存利用率，减少OOM错误的发生。
   ```
   export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True
   ```
2. 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。
3. 替换深度学习训练加速的工具或增加zero等级，可参考**模型NPU卡数、梯度累积值取值表**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。
   a. - ZeRO-0 数据分布到不同的NPU
   b. - ZeRO-1 Optimizer States分布到不同的NPU
   c. - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
   d. - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU
4. 增加卡数重新训练，未解决找相关人员定位。

问题 2：访问容器目录时提示 Permission denied

解决方法：

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: XXX not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

问题 4：Error waiting on exit barrier 错误

错误截图：


```
[ERROR] Error waiting on exit barrier. Elapsed: 300.1057639122009 seconds
[ERROR] Traceback (most recent call last):
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py

[ERROR]     store_util.barrier(
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     synchronize(store, data, rank, world_size, key_prefix, barrier_timeout)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li

[ERROR]     agent_data = get_all(store, rank, key_prefix, world_size)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     store_get(f"{prefix}{node_rank}_FIN")
[ERROR] torch.distributed.DistStoreError: Socket Timeout
```

报错原因：多线程退出各个节点间超时时间默认为300s，时间设置过短。

解决措施：

修改容器内torch/distributed/elastic/agent/server/api.py文件参数：

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py
```

修改def _exit_barrier(self)方法中的barrier_timeout参数，修改后如图4-351所示。

```
#修改前
barrier_timeout=self._exit_barrier_timeout
#修改后
barrier_timeout=3000
```

图 4-351 修改后的 barrier_timeout 参数

```
913     def _exit_barrier(self):
914         """
915         Define a barrier that keeps the agent process alive until all workers finish.
916
917         Wait for ``exit_barrier_timeout`` seconds for all agents to finish
918         executing their local workers (either successfully or not). This
919         acts as a safety guard against user scripts that terminate at different
920         times.
921         """
922         log.info(
923             "Local worker group finished (%s). "
924             "Waiting %s seconds for other agents to finish",
925             self._worker_group.state, self._exit_barrier_timeout
926         )
927         start = time.time()
928         try:
929             store_util.barrier(
930                 self._store,
931                 self._worker_group.group_rank,
932                 self._worker_group.group_world_size,
933                 key_prefix= TERMINAL_STATE_SYNC_ID,
934                 barrier_timeout=3000,
935             )
936             log.info(
937                 "Done waiting for other agents. Elapsed: %s seconds", time.time() - start
938             )
939         except SignalException as e:
940             log.warning("Got termination signal: %s", e.sigval)
941             raise
942         except Exception:
943             log.exception(
944                 "Error waiting on exit barrier. Elapsed: %s seconds",
945                 time.time() - start
946             )
```

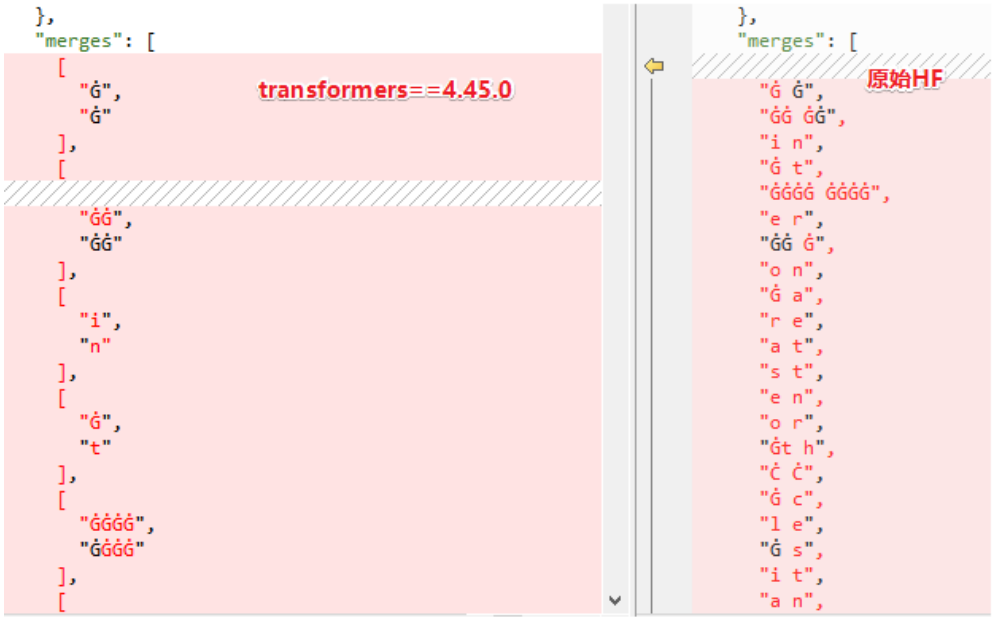
问题 5：训练完成使用 vllm0.6.0 框架推理失败：

错误截图：

```
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/tokenization_u
tils_fast.py", line 115, in __init__
    fast_tokenizer = TokenizerFast.from_file(fast_tokenizer_file)
Exception: data did not match any variant of untagged enum ModelWrapper at line 757272 column 1
```

报错原因：

训练时transformers版本要求为4.45.0，训练完成后保存的tokenizer.json文件中的“merges”时保存的是拆开的列表不是字符串，导致推理异常



解决措施，以下两种方法任选其一：

1. 更新transformes和tokenizers版本
   - GLM4-9B模型，容器内执行以下步骤：
     ```
     pip install transformers==4.43.2
     ```
   - 其它模型，容器内执行以下步骤：
     ```
     pip install transformers==4.45.0
     pip install tokenizers==0.20.0
     ```
2. 使用原始hf权重的tokenizer.json覆盖保存的tokenizer.json即可，如llama3-8b_lora具体过程如下：
   ```
   # 进入模型tokenizer目录
   cd /home/ma-user/ws/tokenizers/llama3-8b/
   # 替换tokenizer.json文件
   cp -f tokenizer.json /home/ma-user/ws/saves/rm/llama3-8b/lora/tokenizer.json
   ```

问题 6：训练过程中报依赖包 tyro 错误:"ModuleNotFoundError: No module named 'tyro'"

错误截图：

```
ModuleNotFoundError: No module named 'tyro'
```

报错原因：未指定tyro依赖包版本，导致安装依赖为最新0.9.0版本导致与其他依赖冲突

解决措施：任务前容器内更新'tyro'版本为0.8.14或以下版本

```
pip install tyro==0.8.14
```

4.21 主流开源大模型基于 Standard+OBS 适配 ModelLink PyTorch NPU 训练指导（6.3.910）

4.21.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.910版本相对于6.3.909版本新增如下内容：

- 文档中新增对Qwen2.5的适配（包括0.5B、7B, 14B, 32B, and 72B），支持sft、lora、预训练。
- 文档中新增对Llama3.2的适配（包括1B和3B），支持sft、lora、预训练。
- 代码中ModelLink、MindSpeed已升级到最新版本，Python三方依赖版本已升级，其中：
  - MindSpeed的版本升级到commitID=4ea42a23
  - ModelLink的版本升级到commitID=8f50777
  - transformers版本升级到4.45.0
  - peft版本升级到0.12.0

支持的模型列表

本方案支持以下模型的训练，如表4-186所示。

表 4-186 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

序号	支持模型	支持模型参数量	权重文件获取地址
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLM v3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan 2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
19		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
20		qwen2-72b	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**
24	llama3.1	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct**
25		llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct**
26	Qwen2.5	qwen2.5-0.5b	**https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct**
27		qwen2.5-7b	**https://huggingface.co/Qwen/Qwen2.5-7B-Instruct**
28		qwen2.5-14b	**https://huggingface.co/Qwen/Qwen2.5-14B-Instruct**
29		qwen2.5-32b	**https://huggingface.co/Qwen/Qwen2.5-32B-Instruct**
30		qwen2.5-72b	**https://huggingface.co/Qwen/Qwen2.5-72B-Instruct**
31	llama3.2	llama3.2-1b	**https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct**
32		llama3.2-3b	**https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct**

操作流程

图 4-352 操作流程图

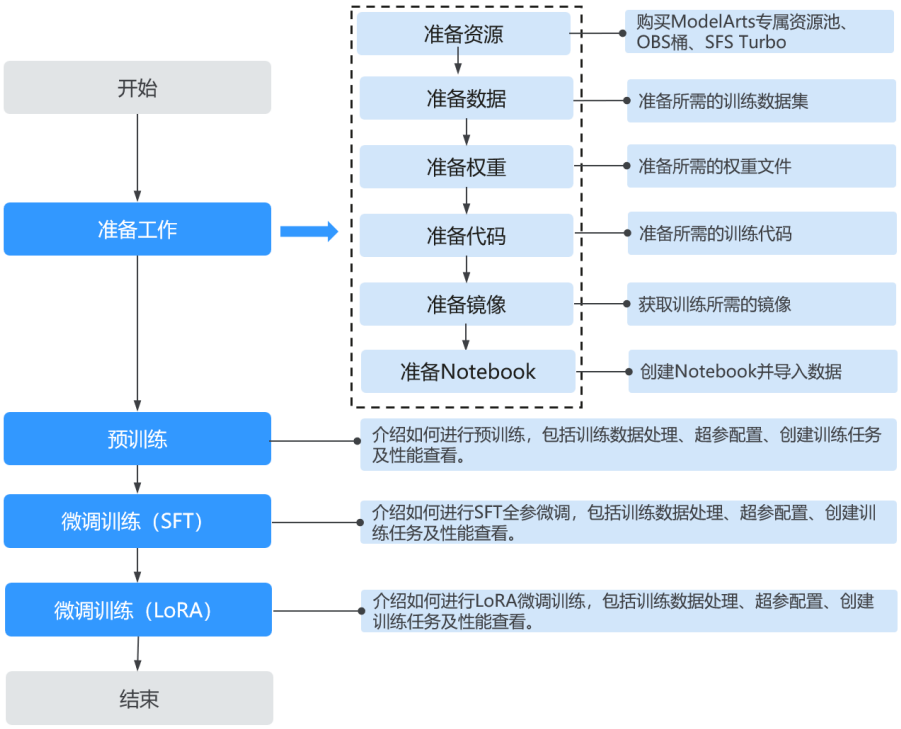


表 4-187 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.21.2 准备工作

4.21.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-193**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.21.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
     'id': '1',
     'url': 'https://simple.wikipedia.org/wiki/April',
     'title': 'April',
     'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

  如果用户希望将MOSS数据集的Excel 格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst

- MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败False。

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.21.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-186**。

   权重文件下载有如下几种方式，但不仅限于以下方式：
   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
```

如果要下载指定版本的模型文件，则命令如下：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.21.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-188**所示。

表 4-188 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.910代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
              |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |——scripts/             # 训练需要的启动脚本
                   |——llama2          # llama2系列模型执行脚本的文件夹
                   |——llama3          # llama3系列模型执行脚本的文件夹
                   |——qwen            # Qwen系列模型执行脚本的文件夹
                   |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   |—— ...
                   |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   |—— install.sh       # 环境部署脚本
              |———src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference               # 推理代码包
|——llm_tools                   # 推理工具
```

代码上传至 OBS

将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|——llm_train                      # 解压代码包后自动生成的代码目录，无需用户创建
     |—— AscendSpeed                # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|—— training_data                                    #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
     |—— train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
     |—— alpaca_gpt4_data.json                          #微调数据文件
|—— tokenizers                    #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
     |—— llama2-13b-hf
|—— models                        #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
     |—— llama2-13b-hf
```

4.21.2.5 准备镜像

4.21.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-189 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行install.sh文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

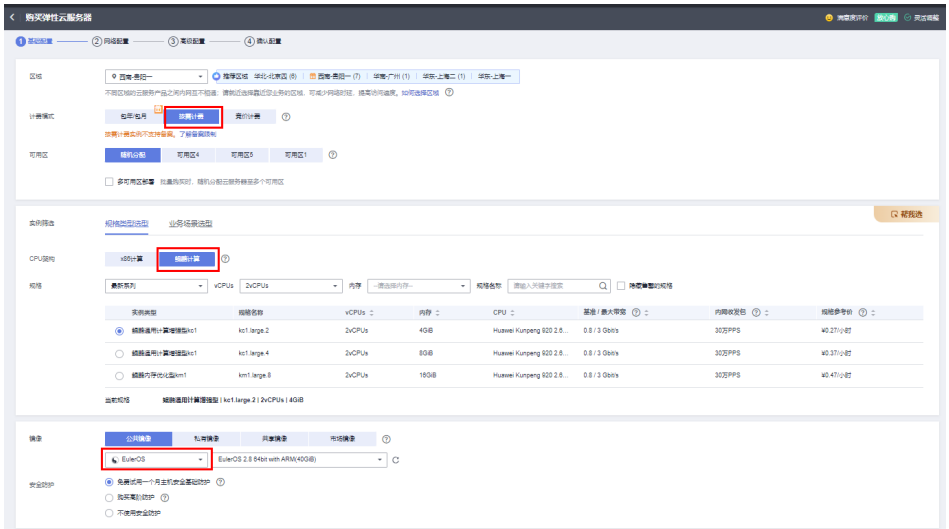
4.21.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

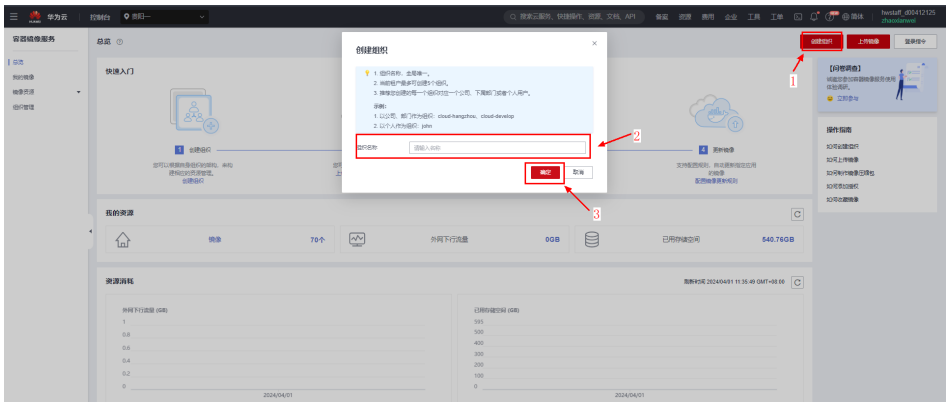
图 4-353 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-354 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

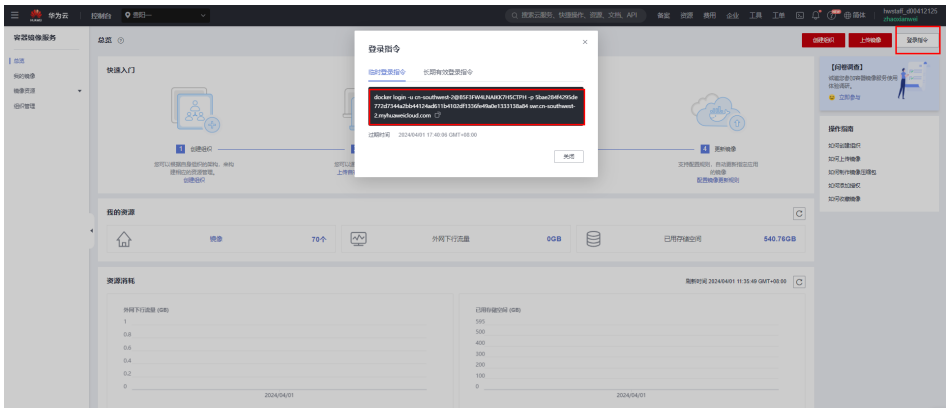
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-189**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-355 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.21.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-356**中都需要执行install.sh文件，来安装依赖以及下载完整代码。命令如下：

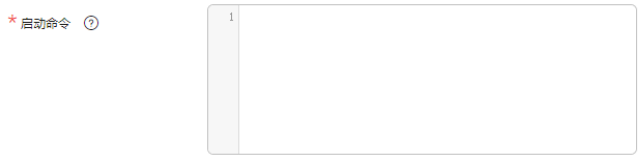
```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

⚠ 注意

- 使用基础镜像的方法，需要确认训练作业的资源池是否联通公网，否则执行install.sh 文件时下载代码会失败。因此可以选择配置网络或使用**ECS中构建新镜像**的方法。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 中的transformers 的版本。
  由默认 transformers==4.45.0 修改为：transformers==4.44.2

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-356 训练作业启动命令



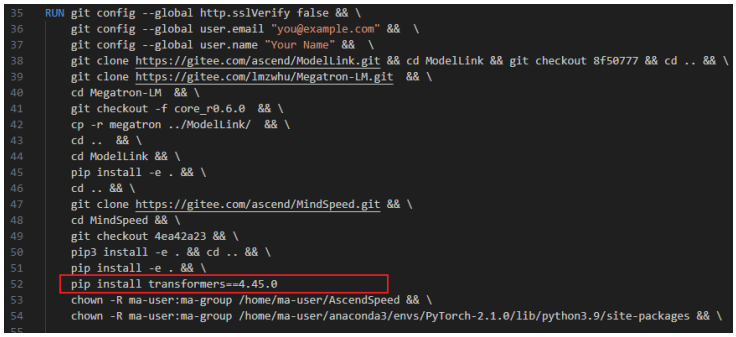
4.21.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-188**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.910-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。



⚠ 注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网

```
docker build -t <镜像名称>:<版本名称> .
```

如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

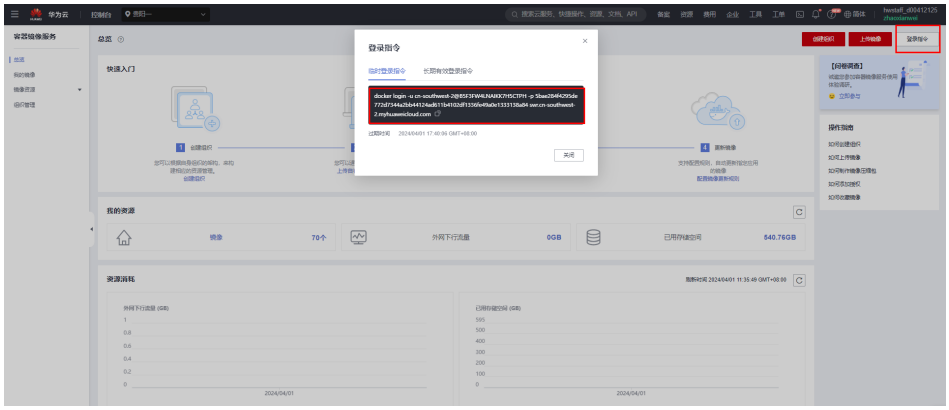
```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称> .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-357 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.21.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中，如果用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-358 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，如果需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-359 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

4.21.3 预训练

前提条件

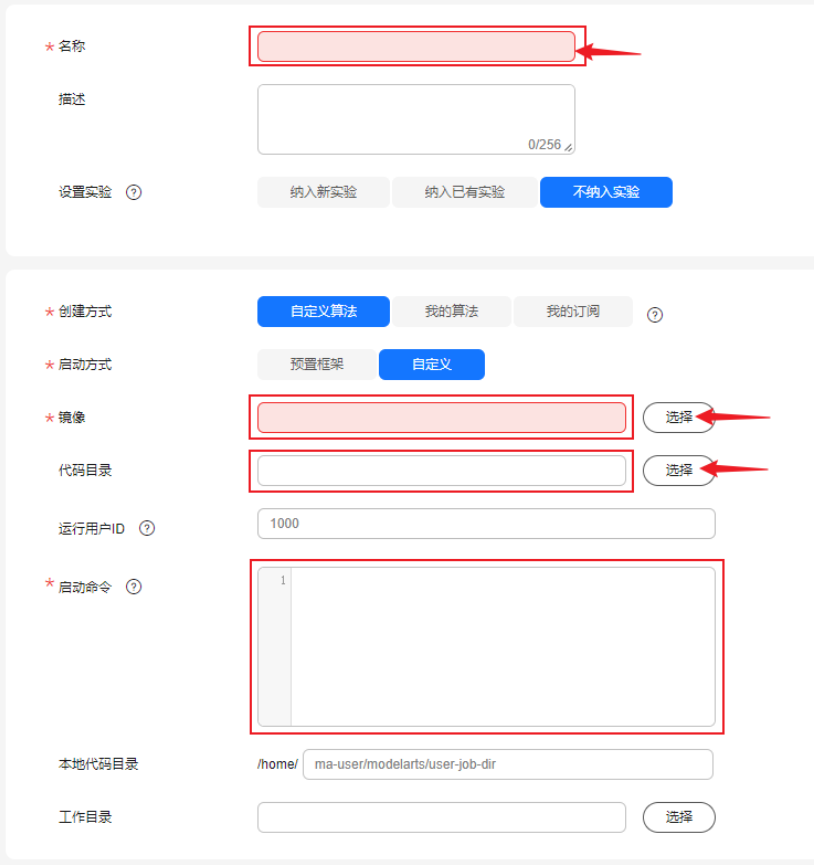
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-360 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-190**表格中的配置进行填写。

表 4-190 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

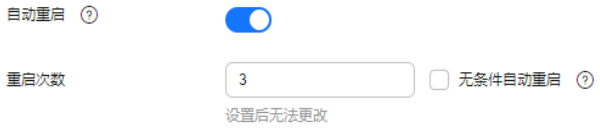
- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。

- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-361 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-193**进行配置。

图 4-362 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.21.4 SFT 全参微调训练

前提条件

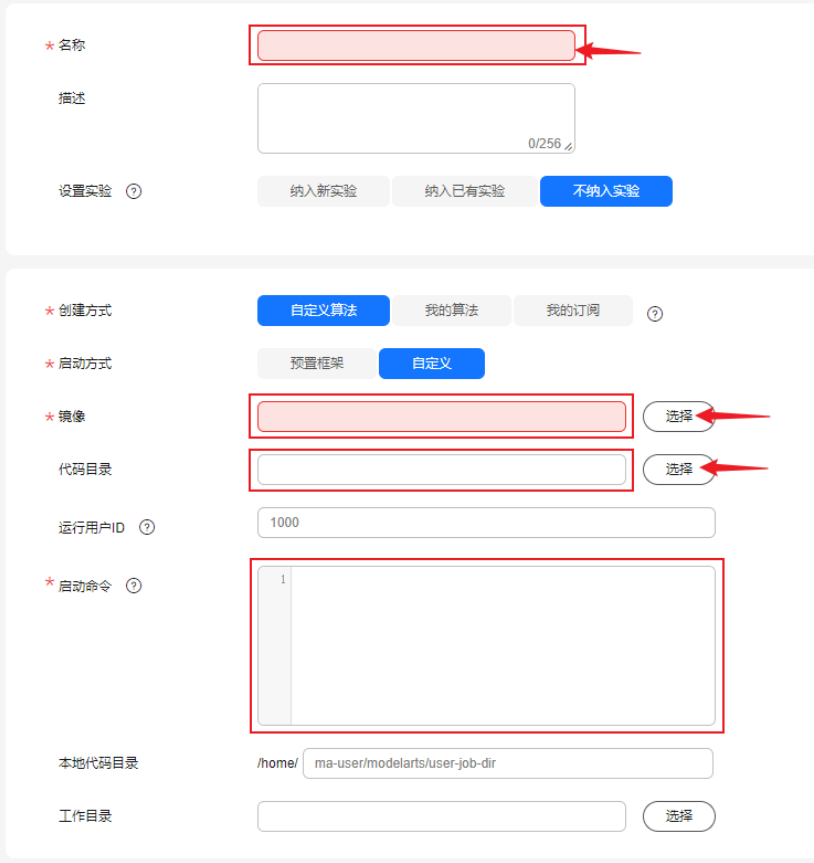
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-363 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-190**表格中的配置进行填写。

图 4-364 环境变量

表 4-191 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。

环境变量	示例值	参数说明
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

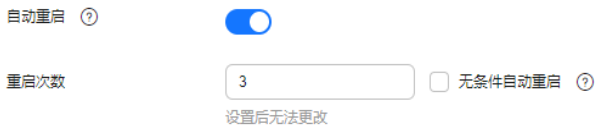
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-365 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-193**进行配置。

图 4-366 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.21.5 LoRA 微调训练

前提条件

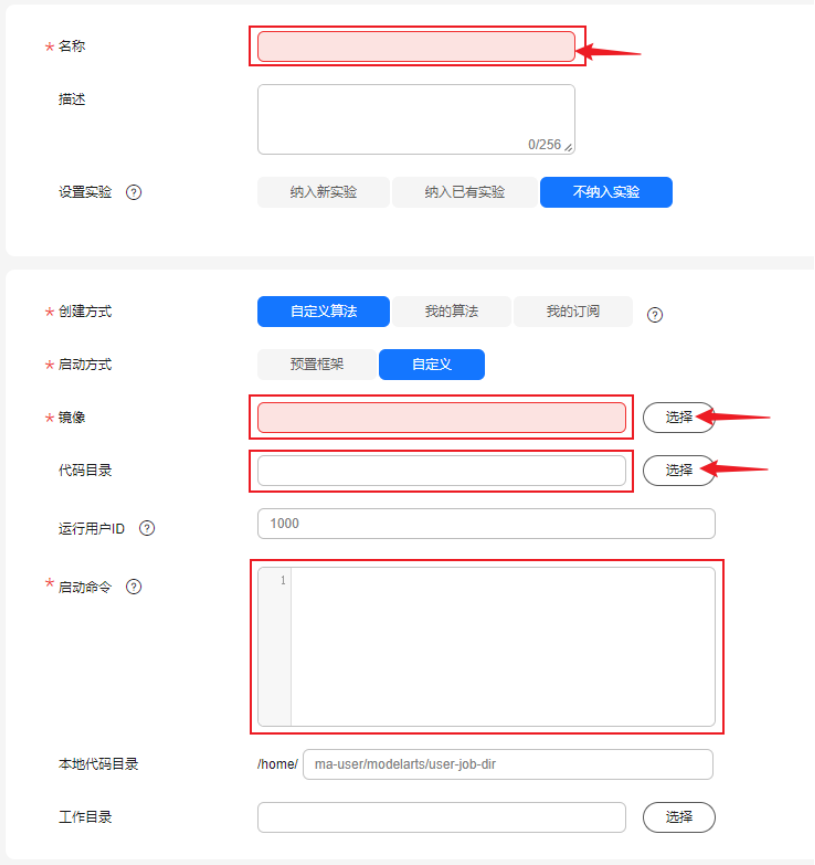
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-367 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-190**表格中的配置进行填写。

表 4-192 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

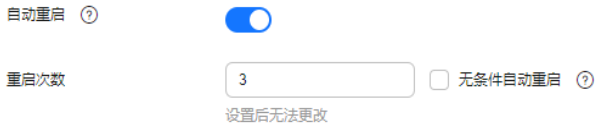
- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。

- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-368 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-193**进行配置。

图 4-369 选择资源池规格

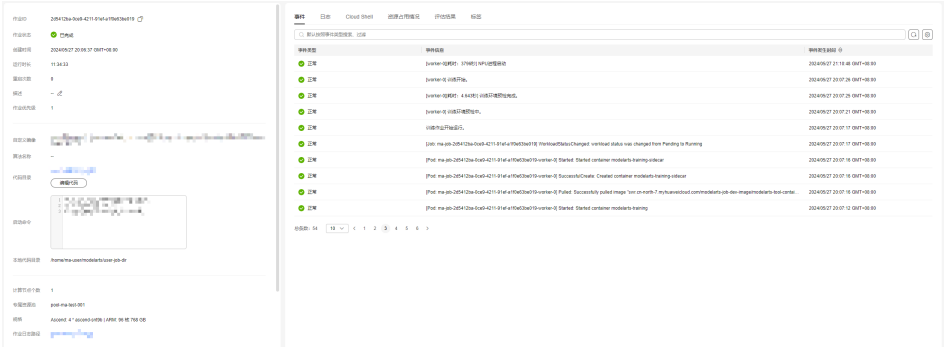
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.21.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

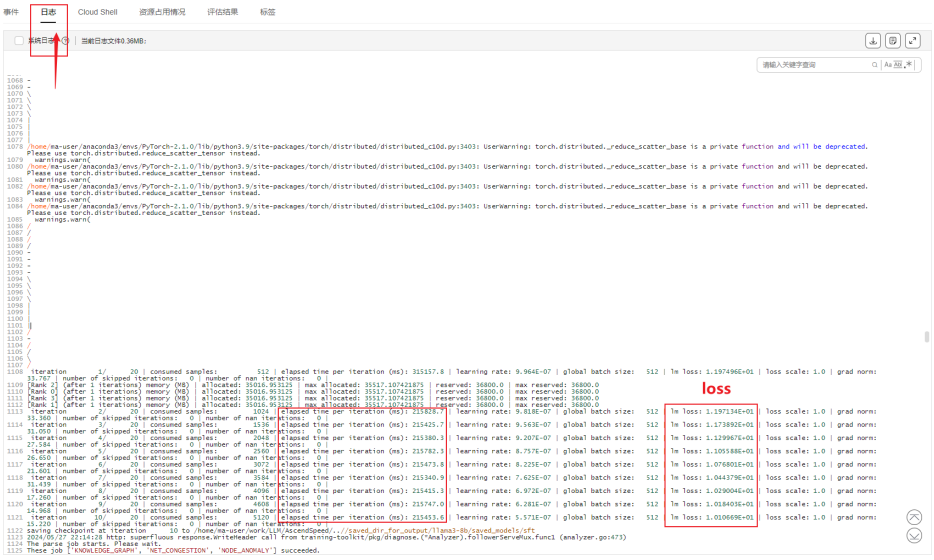
图 4-370 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-371 查看日志和性能



4.21.7 训练脚本说明

4.21.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 ……）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据

和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-193**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-193 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
4	llama3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.21.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。如果未进行数据集预处理，则会自动执行**scripts/llama2/1_preprocess_data.sh**。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。

- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

  ```
  [
    {"text": "document"},
    {"other keys": "optional content"}
  ]
  ```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

  ```
  doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
  ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

  ```
  def _filter(self, sample):
      messages = self._format_msg(sample)
      full_prompt = self.prompter.generate_training_prompt(messages)
      tokenized_full_prompt = self._tokenize(full_prompt)
      if self.args.append_eod:
          tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
          tokenized_full_prompt["attention_mask"].append(1)
          tokenized_full_prompt["labels"].append(self.tokenizer.eod)
      if not self.train_on_inputs:
          user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
              self.prompter.template.assistant_token + "\n"
          tokenized_user_prompt = self._tokenize(user_prompt)
          user_prompt_len = len(tokenized_user_prompt["input_ids"])
          tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
      for key in self.args.json_keys:
          tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
      return tokenized_full_prompt
  ```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

iii. input_ids是user_ids和assistant_ids的拼接

iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-194 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。

环境变量	示例	参数说明
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.21.7.3 训练的权重转换说明

以**llama2-13b**举例，使用训练作业运行**obs_pipeline.sh**脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

如果用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。如果用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-195 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.21.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-372所示。

图 4-372 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-373所示。

图 4-373 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-374 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-375所示。

图 4-375 修改 ChatGLMv4-9B tokenizer 文件

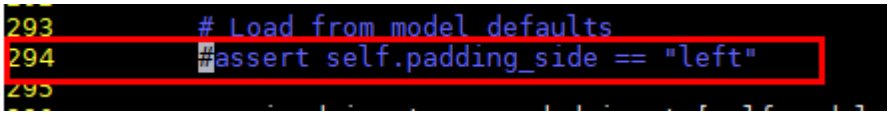
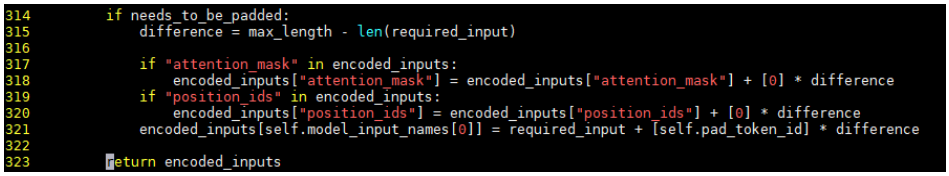


图 4-376 修改 ChatGLMv4-9B tokenizer 文件

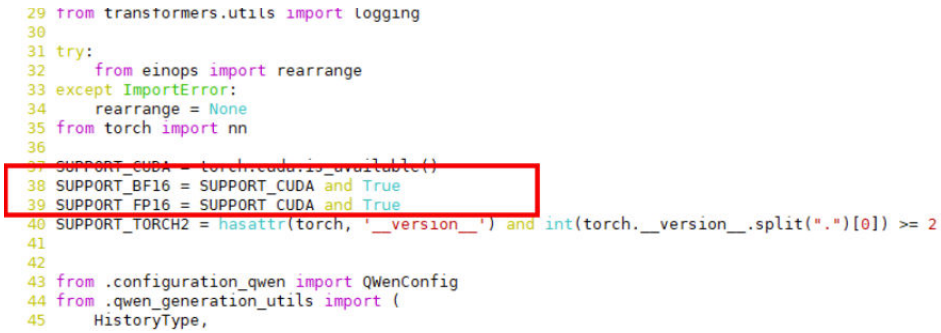


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-377所示。

图 4-377 修改 Qwen tokenizer 文件



4.21.8 常见错误原因和解决方法

4.21.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-193**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.21.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.22 主流开源大模型基于 Standard+OBS+SFS 适配 ModelLink PyTorch NPU 训练指导（6.3.910）

4.22.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.910版本相对于6.3.909版本新增如下内容：

- 文档中新增对Qwen2.5的适配（包括0.5B、7B, 14B, 32B, and 72B），支持sft、lora、预训练。
- 文档中新增对Llama3.2的适配（包括1B和3B），支持sft、lora、预训练。
- 代码中ModelLink、MindSpeed已升级到最新版本，Python三方依赖版本已升级，其中：
  - MindSpeed的版本升级到commitID=4ea42a23
  - ModelLink的版本升级到commitID=8f50777
  - transformers版本升级到4.45.0
  - peft版本升级到0.12.0

支持的模型列表

本方案支持以下模型的训练，如表4-196所示。

表 4-196 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

序号	支持模型	支持模型参数量	权重文件获取地址
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLM v3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan 2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
19		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
20		qwen2-72b	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**
24	llama3.1	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct**
25		llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct**
26	Qwen2.5	qwen2.5-0.5b	**https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct**
27		qwen2.5-7b	**https://huggingface.co/Qwen/Qwen2.5-7B-Instruct**
28		qwen2.5-14b	**https://huggingface.co/Qwen/Qwen2.5-14B-Instruct**
29		qwen2.5-32b	**https://huggingface.co/Qwen/Qwen2.5-32B-Instruct**
30		qwen2.5-72b	**https://huggingface.co/Qwen/Qwen2.5-72B-Instruct**
31	llama3.2	llama3.2-1b	**https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct**
32		llama3.2-3b	**https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct**

操作流程

图 4-378 操作流程图

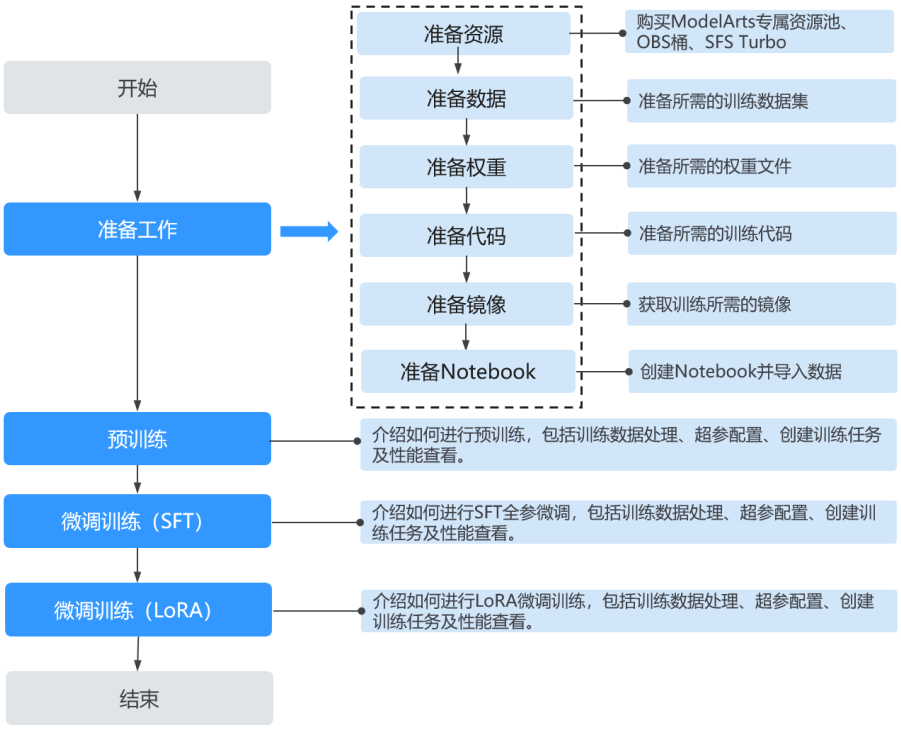


表 4-197 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.22.2 准备工作

4.22.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-204**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。具体过程请参考**创建OBS桶**，例如桶名：standard-llama2-13b。

由于ModelArts创建训练作业时，需要将作业日志输出至OBS桶中，因此创建OBS桶为必选项。用户可通过**OBS Browser+**、**obsutil**等工具访问和管理OBS桶，将代码、模型文件、数据集等数据上传或下载进行备份。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

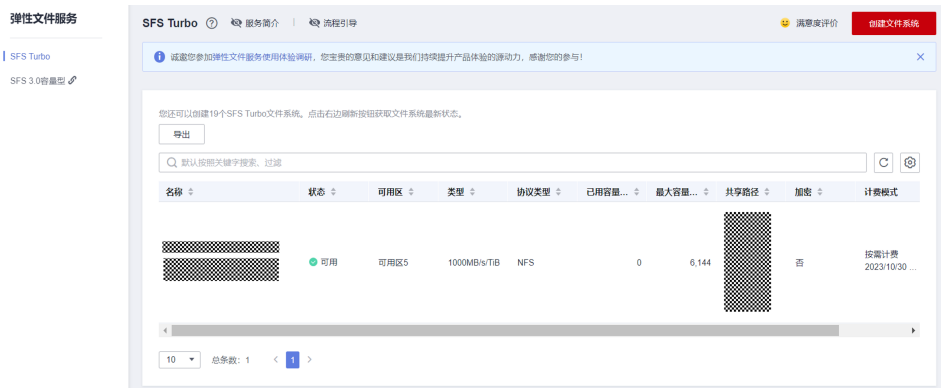
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-379 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

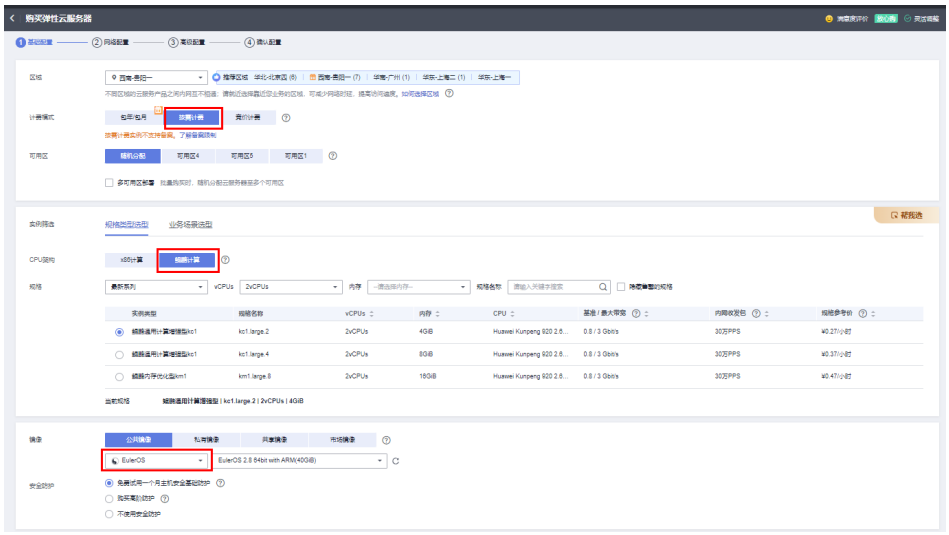
图 4-380 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 4-381 购买 ECS

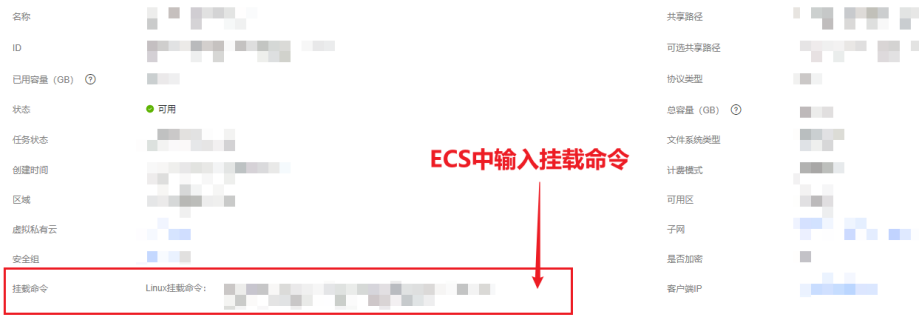


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看基本信息图4-382，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 4-382 SFS Turbo 基本信息



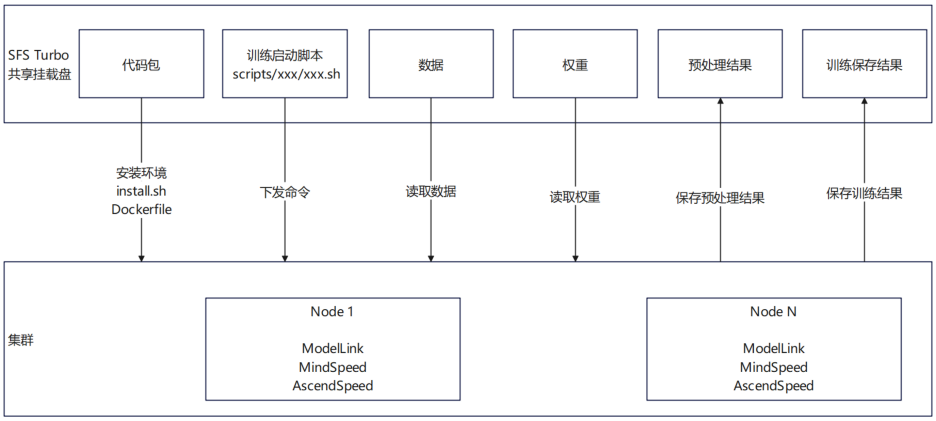
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-383 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表4-199**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.22.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败False。

上传数据集至 SFS Turbo

准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。可通过两种方式，将数据集上传至SFS Turbo中。

方式一：将下载的原始数据通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/training_data目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具。数据存放参考目录：

```
/mnt/sfs_turbo/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
2. 利用**OBS Browser+工具**将下载的数据集上传至创建的文件夹目录下。得到OBS下数据集结构：
```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```
3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.22.2.3 准备权重

获取对应模型的权重文件，获取链接参考**表4-196**。权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。文件会直接下载用户本地，需要再上传至SFS Turbo中。
- **方法二：huggingface-cli**：**huggingface-cli**是Hugging Face官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
```
  如果要下载指定版本的模型文件，则命令如下：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```
- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

随后可通过以下两种方式，将下载到本地的模型文件上传至SFS Turbo中。

本地上传权重文件至 SFS Turbo

通过以下两种方式将下载到本地的模型文件上传至SFS Turbo中。方式一操作简单，但是数据传输速度比较慢，费时间。方式二操作相对方式一复杂一些，但是数据传输速度较快。

方式一：将已下载的模型文件通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/model目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放模型，例如在桶standard-llama2-13b中创建文件夹model/llama-2-13b-hf。
2. 利用**OBS Browser+工具**将下载的模型文件上传至创建的文件夹目录下。

3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.22.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-198**所示。

表 4-198 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.910代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
                |——llama2           # llama2系列模型执行脚本的文件夹
                |——llama3           # llama3系列模型执行脚本的文件夹
                |——qwen             # Qwen系列模型执行脚本的文件夹
                |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |—— ...
                |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |—— install.sh       # 环境部署脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

代码上传至 SFS Turbo

将AscendSpeed代码包AscendCloud-LLM-xxx.zip直接上传至ECS服务器中的SFS Turbo中，例如存放在/mnt/sfs_turbo/AscendCloud-LLM-xxx.zip目录下并解压缩。

```
unzip AscendCloud-*.zip
```

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至SFS Turbo后，目录结构如下。

```
/mnt/sfs_turbo/
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          # 目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
            |── pretrain   # 预训练加载的数据
            |── finetune   # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output          # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
                |── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                          #微调数据文件
|── tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
|── models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.22.2.5 准备镜像

4.22.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-199 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**可二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

使用以上方案时，都会下载Megatron-LM、MindSpeed、ModelLink源码至AscendSpeed文件夹中。下载后的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/             # 训练需要的启动脚本
  |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/           # MindSpeed昇腾大模型加速库
  |——ModelLink/           # ModelLink端到端的大语言模型方案
      |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

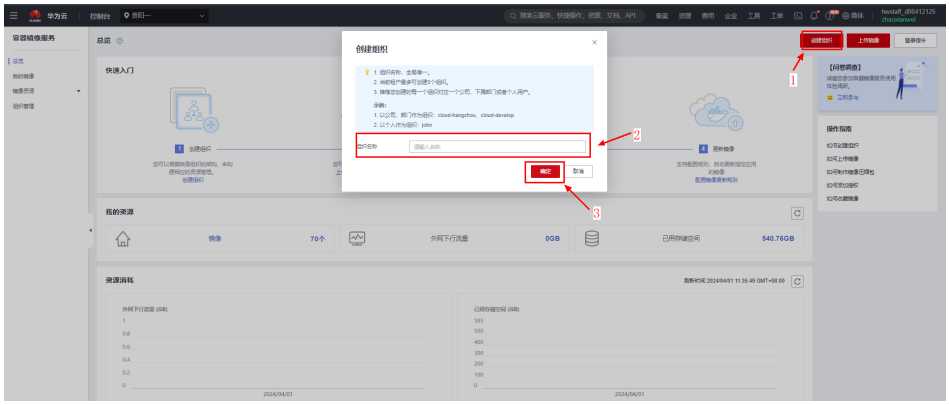
训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

4.22.2.5.2 ECS 获取和上传基础镜像

Step1 创建镜像组织

在SWR服务页面创建镜像组织。

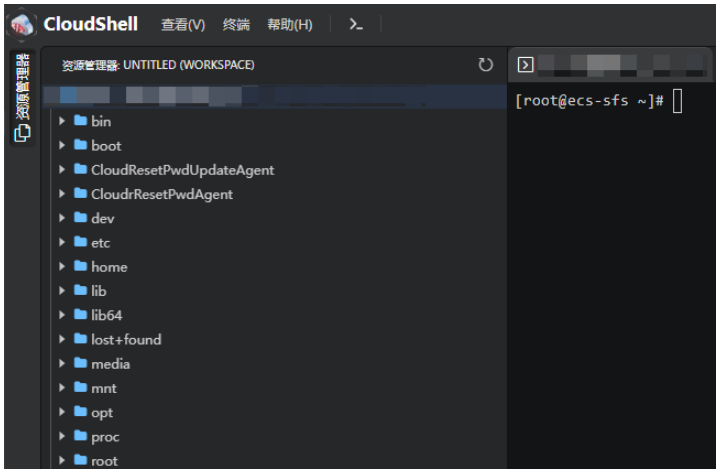
图 4-384 创建镜像组织



Step2 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录如图所示。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

图 4-385 CloudShell 远程登录界面



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

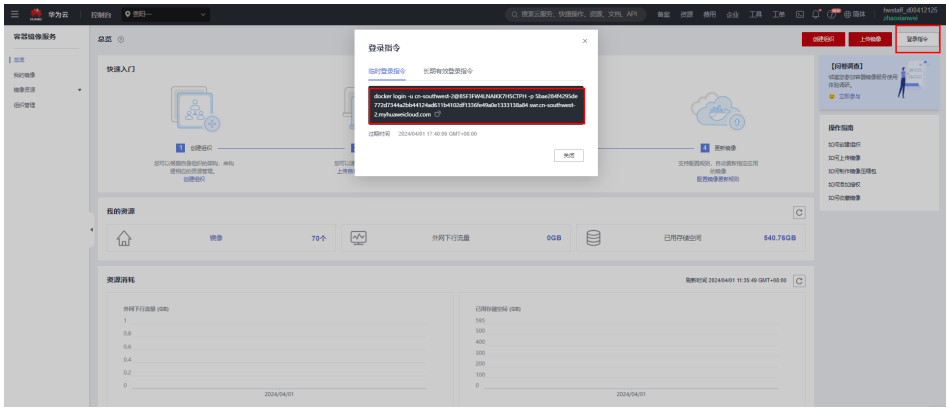
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-199**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-386 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.22.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-387**中都需要执行 install.sh文件，来安装依赖以及下载完整代码。

⚠ 注意

- 使用基础镜像的方法，需要确认训练作业的资源池是否联通公网，否则执行 install.sh 文件时下载代码会失败。因此可以选择配置网络或使用**ECS中构建新镜像**的方法。
- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 中的 transformers 的版本。

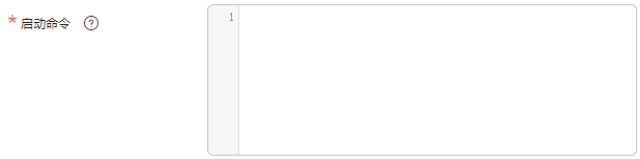
  由默认 transformers==4.45.0 修改为：transformers==4.44.2

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-387 训练作业启动命令



4.22.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-198**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   cd ./llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。

   ```
   RUN git config --global http.sslVerify false && \
       git config --global user.email "you@example.com" &&  \
       git config --global user.name "Your Name" &&  \
       git clone https://gitee.com/ascend/ModelLink.git && cd ModelLink && git checkout 8f50777 && cd .. && \
       git clone https://gitee.com/lmzwhu/Megatron-LM.git  && \
       cd Megatron-LM  && \
       git checkout -f core_r0.6.0  && \
       cp -r megatron ../ModelLink/  && \
       cd ..  && \
       cd ModelLink && \
       pip install -e . && \
       cd .. && \
       git clone https://gitee.com/ascend/MindSpeed.git && \
       cd MindSpeed && \
       git checkout 4ea42a23 && \
       pip3 install -e . && cd .. && \
       pip install -e . && \
       pip install transformers==4.45.0
       chown -R ma-user:ma-group /home/ma-user/AscendSpeed && \
       chown -R ma-user:ma-group /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages && \
   ```

注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称> .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

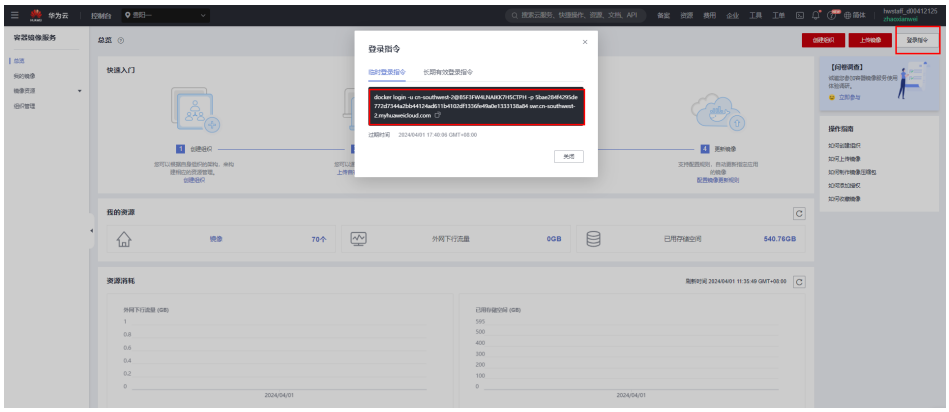
```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://
xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-388 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.22.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-200**所示。其他超参均有默认值，可以参考**表4-203**按照实际需求修改。

表 4-200 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	**可添加**。该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。

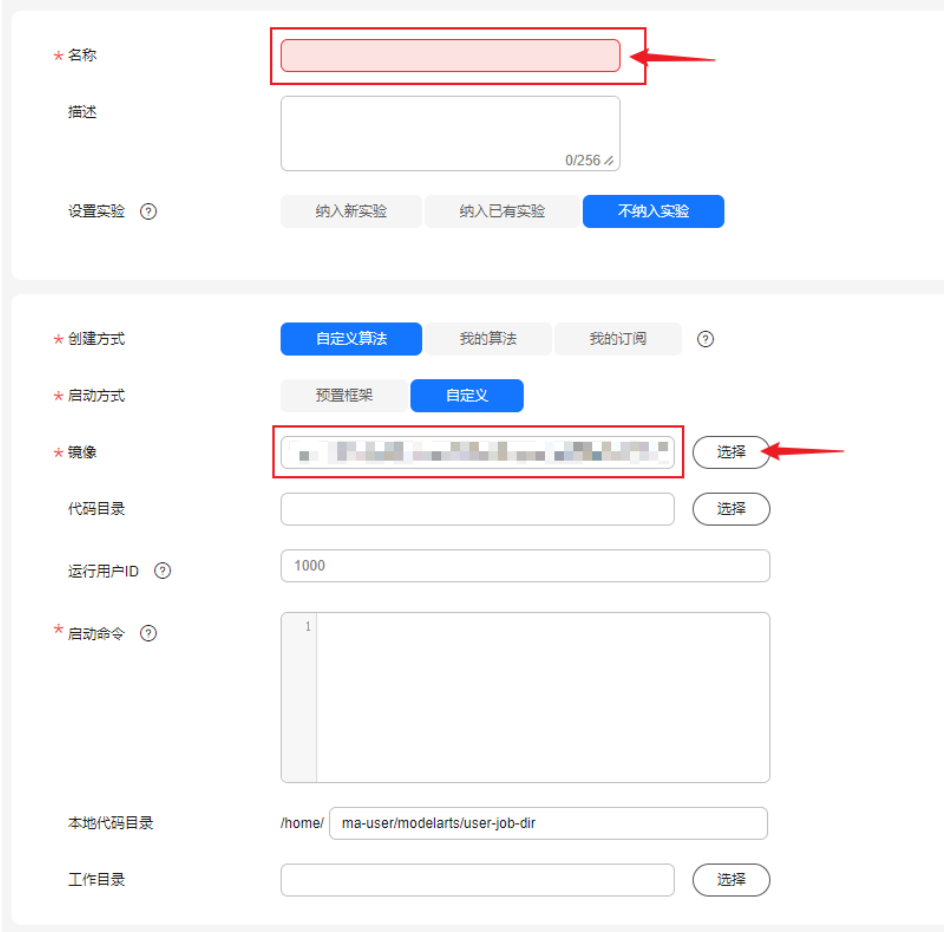
参数	示例值	参数说明
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-389 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

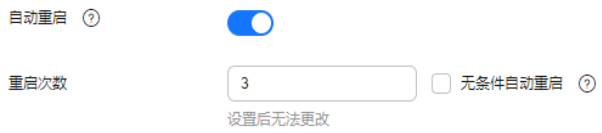
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-390 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

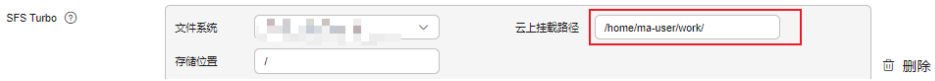
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-204**进行配置。

图 4-391 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-392 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.22.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-201所示。其他超参均有默认值，可以参考表4-203按照实际需求修改。

表 4-201 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

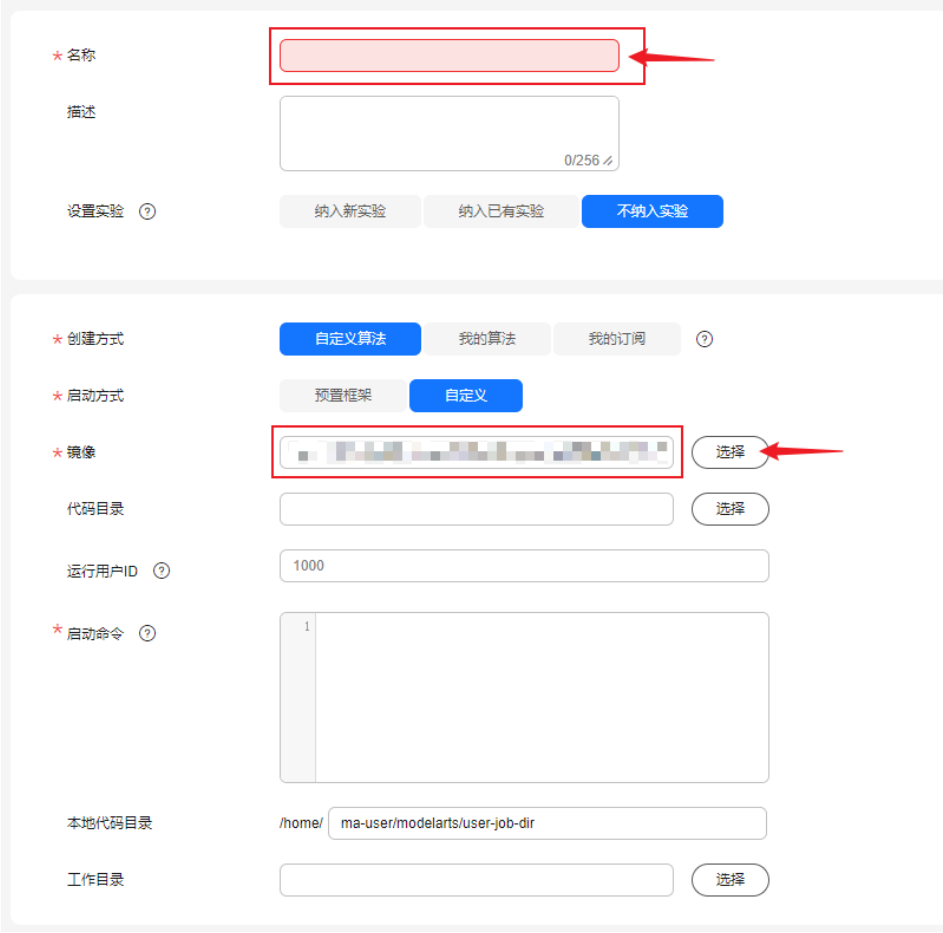
参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-393 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

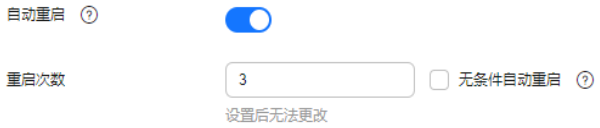
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-394 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-204**进行配置。

图 4-395 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-396 选择 SFS Turbo

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.22.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-202**所示。其他超参均有默认值，可以参考**表4-203**按照实际需求修改。

表 4-202 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

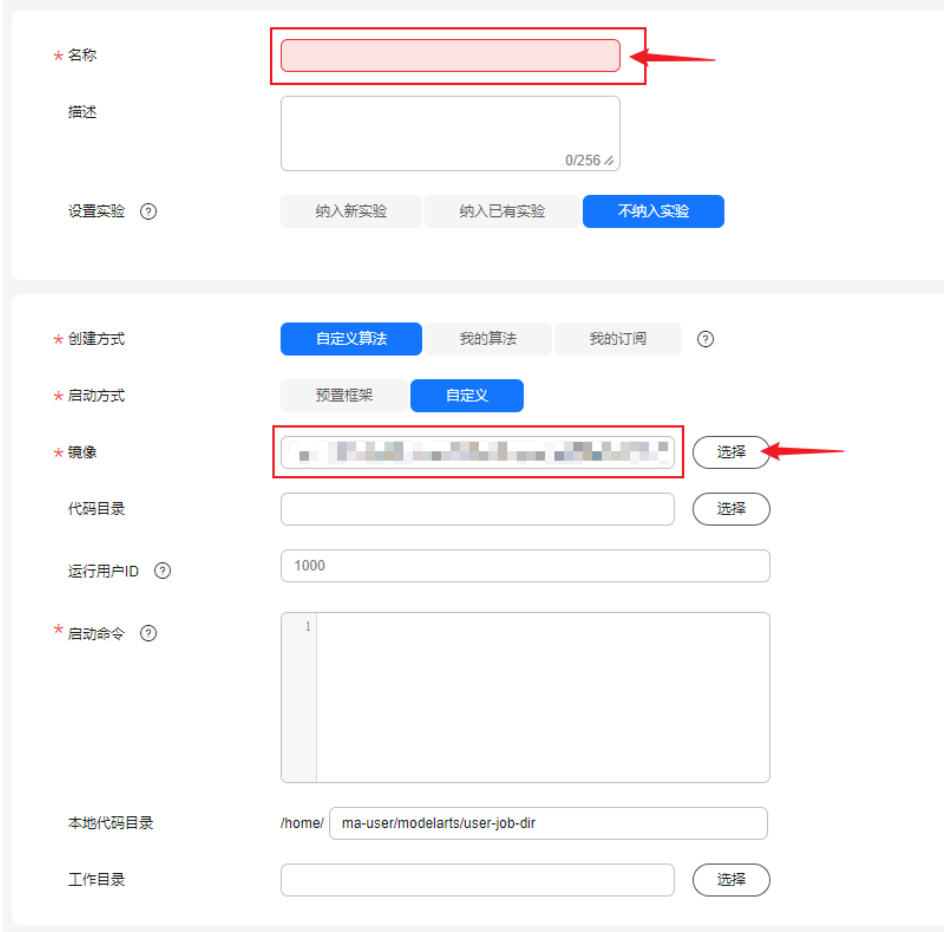
参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-397 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

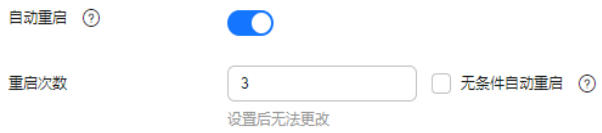
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-398 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

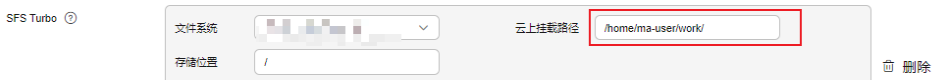
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-204**进行配置。

图 4-399 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-400 选择 SFS Turbo



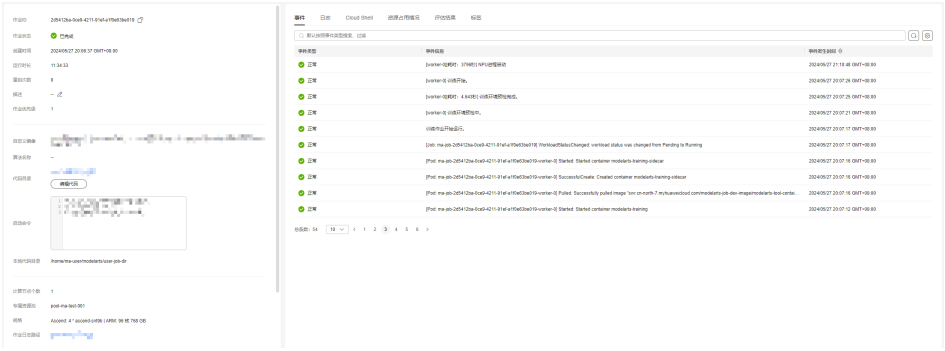
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.22.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

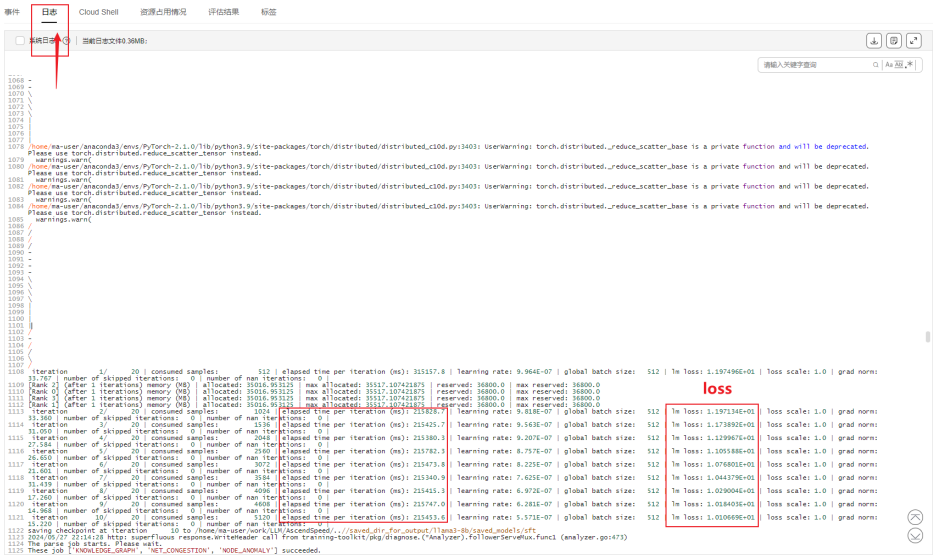
图 4-401 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-402 查看日志和性能



4.22.7 训练脚本说明

4.22.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**

环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-203 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。

参数	示例值	参数说明
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-204所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-204 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点&8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点&8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点&8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.22.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**0_pl_pretrain_13b.sh**训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。支持 .parquet \ .csv \ .json \ .jsonl \ .txt \ .arrow 格式。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

  ```
  def serialize_to_disk(self):
      """save idx and bin to disk"""
      startup_start = time.time()
      if not self.tokenized_dataset:
          self.tokenized_dataset = self.get_tokenized_data()
      output_bin_files = {}
      output_idx_files = {}
      builders = {}
      level = "document"
      if self.args.split_sentences:
          level = "sentence"
      logger.info("Vocab size: %s", self.tokenizer.vocab_size)
      logger.info("Output prefix: %s", self.args.output_prefix)
      for key in self.args.json_keys:
          ## 写入磁盘
  ```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

  ```
  def _filter(self, sample):
      sample = self._pre_process(sample)
      for key in self.args.json_keys:
          text = sample[key]
          doc_ids = []
          for sentence in self.splitter.tokenize(text):
              if len(sentence) > 0:
                  sentence_ids = self._tokenize(sentence)
                  doc_ids.append(sentence_ids)
          if len(doc_ids) > 0 and self.args.append_eod:
              doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
              doc_ids[-1]['attention_mask'].append(1)
              doc_ids[-1]['labels'].append(self.tokenizer.eod)
          sample[key] = doc_ids
          # for now, only input_ids are saved
          sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
      return sample
  ```

  - 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

    ```
    [
      {"text": "document"},
      {"other keys": "optional content"}
    ]
    ```

  - 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

    ```
    doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
    ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

  - 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

  - 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

     iii. input_ids是user_ids和assistant_ids的拼接

     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

  d. attention_mask是和input_ids等长的全1序列

  e. 返回input_ids\attention_mask\labels的字典

  f. 处理完单一sample

  注：labels中用-100填充的地方，表示会被loss_mask给mask掉

  - 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

  - 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
```

```
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  {MOSS}
  ```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  ```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以llama2为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-205 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.22.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。

- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-206 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.22.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-403所示。

图 4-403 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-404 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-405 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-406 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-407 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-408 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.22.8 常见错误原因和解决方法

4.22.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-204**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.22.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.23 主流开源大模型基于 Lite Cluster 适配 ModelLink PyTorch NPU 训练指导（6.3.910）

4.23.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Cluster上的训练方案。训练框架使用的是ModelLink。

本方案目前仅适用于企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**表4-209**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Cluster。
- 本文档中的**CCE集群版本选择v1.27~1.28**。版本使用的容器引擎为**Containerd**。
- 镜像适配的Cann版本是cann_8.0.rc3，驱动版本是23.0.6。

- 确保集群可以访问公网。

文档更新内容

6.3.910版本相对于6.3.909版本新增如下内容：

- 文档中新增对Qwen2.5的适配（包括0.5B、7B, 14B, 32B, and 72B），支持sft、lora、预训练。
- 文档中新增对Llama3.2的适配（包括1B和3B），支持sft、lora、预训练。
- 代码中ModelLink、MindSpeed已升级到最新版本，Python三方依赖版本已升级，其中：
  - MindSpeed的版本升级到commitID=4ea42a23
  - ModelLink的版本升级到commitID=8f50777
  - transformers版本升级到4.45.0
  - peft版本升级到0.12.0

训练支持的模型列表

本方案支持以下模型的训练，如表4-207所示。

表 4-207 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLM v3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan 2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
26	Qwen2.5	qwen2.5-0.5b	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
27		qwen2.5-7b	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
28		qwen2.5-14b	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
29		qwen2.5-32b	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
30		qwen2.5-72b	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
31	llama3.2	llama3.2-1b	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
32		llama3.2-3b	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

操作流程

图 4-409 操作流程图

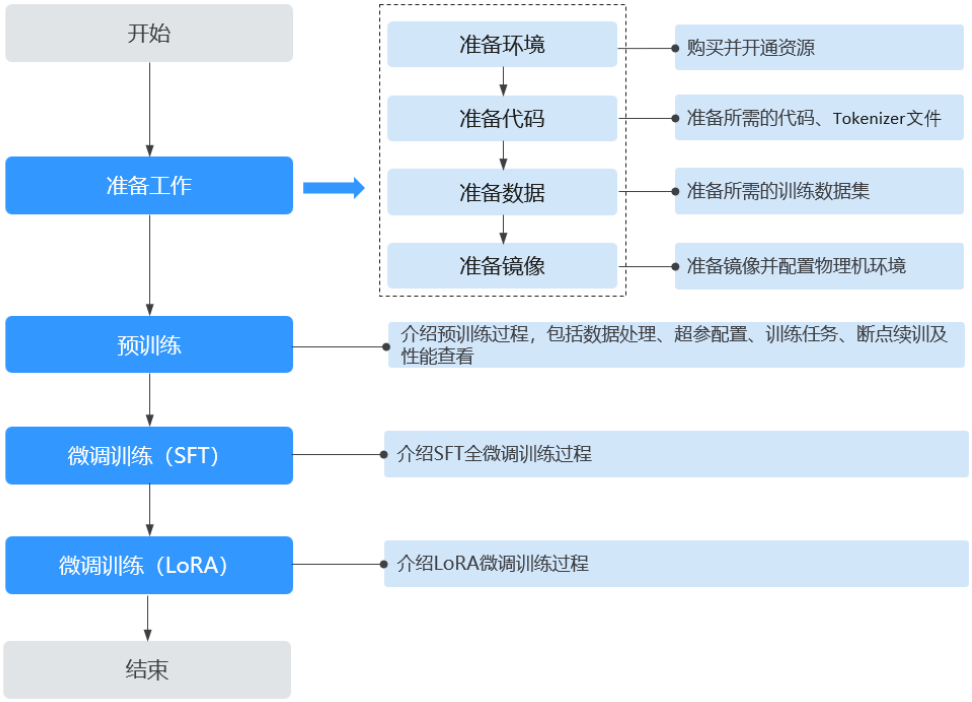


表 4-208 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite k8s Cluster运行的，需要购买并开通k8s Cluster资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.23.2 准备工作

4.23.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Cluster。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-216**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**k8s Cluster资源购买**，熟悉集群资源开通流程，再开始操作购买Cluster资源。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

k8s Cluster 资源配置

若已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

其中k8s Cluster的容器中挂载存储支持OBS、SFS Turbo等方案进行挂载。例如OBS支持静态挂载和动态挂载，而SFS Turbo仅支持静态挂载，详细的挂载操作流程可阅读**通过静态存储卷使用已有极速文件存储**和**通过动态存储卷使用对象存储**。

kubectl 访问集群配置

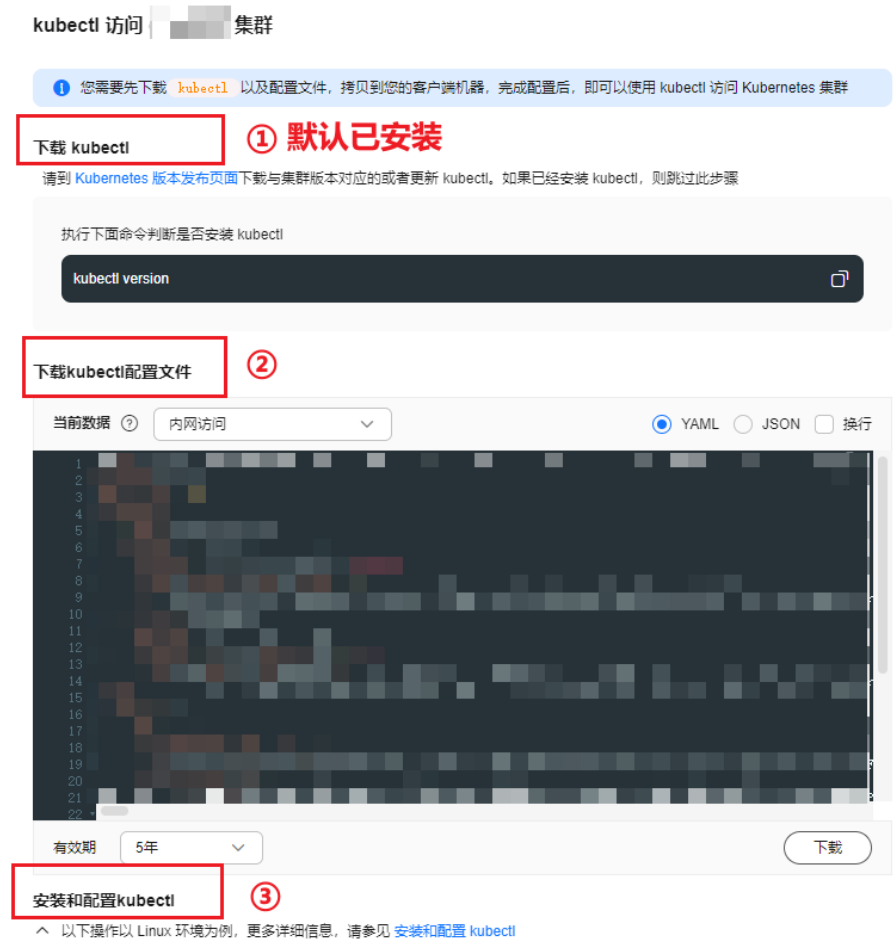
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的 CCE 集群控制版面中。根据**图4-410**的步骤进行操作，单击kubectl配置时，会弹出**图4-411**步骤页面。

图 4-410 配置中心

2. 根据图4-411，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-411 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。若显示如图图4-412的内容，则配置成功。

```
kubectl cluster-info
```

图 4-412 查看 Kubernetes 集群信息正确弹出内容

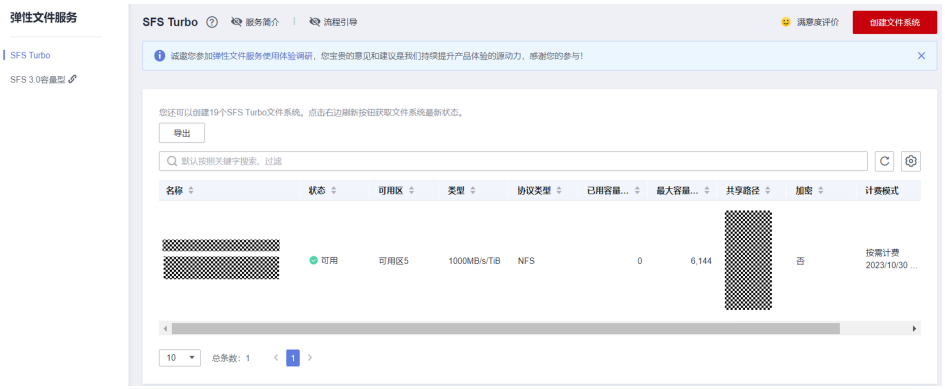
创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。

图 4-413 创建 SFS Turbo



2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

   详细操作指导请参考**创建SFS Turbo文件系统**。

   其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-414 SFS 类型和容量选择

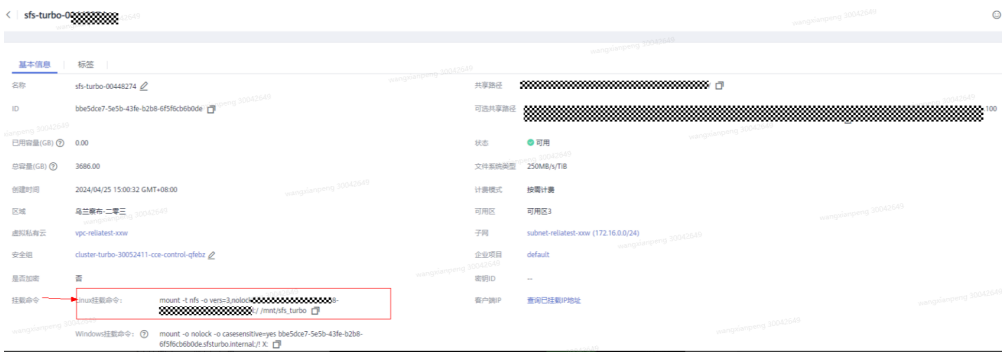
CCE 集群关联 SFS Turbo

进入已购买创建的CCE集群，选择存储，随后单击“创建存储卷声明PVC”。

- 选择“极速文件存储”，随后输入PVC名称。
- 选择“新建存储卷PV”，并单击“选择极速文件存储”。
- 进入选择页面，选择已经创建好的SFS Turbo，最后输入PV名称。

接下来需要通过访问集群节点，挂载SFS Turbo。

- 可通过ssh登录CCE集群中的某个节点（ssh使用的是eip地址）。
- 创建/mnt/sfs_turbo目录作为挂载目录，命令为：mkdir /mnt/sfs_turbo
- SFS Turbo存储手动挂载到安装节点中，挂载命令如下截图：
- 挂载完成后，可通过以下步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。



4.23.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如表4-209所示，模型列表、对应的开源权重获取地址如表4-207所示。

表 4-209 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.910-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

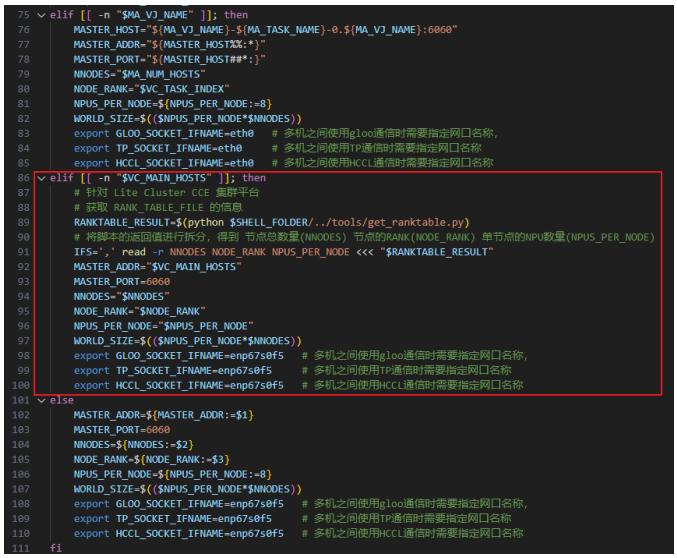
修改代码

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后。在上传代码前，需要对解压后的训练脚本代码进行修改。具体文件为：修改llm_train/AscendSpeed/scripts/dev_pipeline.sh以及新建文件llm_train/AscendSpeed/scripts/tools/get_rank_table.py 。

1. dev_pipeline.sh 具体添加代码内容以及位置，如下所示。

```
elif [[ -n "$VC_MAIN_HOSTS" ]]; then
    # 针对 Lite Cluster CCE 集群平台
    # 获取 RANK_TABLE_FILE 的信息
    RANKTABLE_RESULT=$(python $SHELL_FOLDER/../tools/get_ranktable.py)
    # 将脚本的返回值进行拆分，得到 节点总数量(NNODES) 节点的RANK(NODE_RANK) 单节点的NPU数量(NPUS_PER_NODE)
    IFS=',' read -r NNODES NODE_RANK NPUS_PER_NODE <<< "$RANKTABLE_RESULT"
    MASTER_ADDR="$VC_MAIN_HOSTS"
    MASTER_PORT=6060
    NNODES="$NNODES"
    NODE_RANK="$NODE_RANK"
    NPUS_PER_NODE="$NPUS_PER_NODE"
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

图 4-415 dev_pipeline.sh 添加代码位置和内容



```
elif [[ -n "$MA_VJ_NAME" ]]; then
    MASTER_HOST="${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:6060"
    MASTER_ADDR="${MASTER_HOST%%:*}"
    MASTER_PORT="${MASTER_HOST##*:}"
    NNODES="$MA_NUM_HOSTS"
    NODE_RANK="$VC_TASK_INDEX"
    NPUS_PER_NODE=${NPUS_PER_NODE:=8}
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=eth0   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=eth0     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=eth0   # 多机之间使用HCCL通信时需要指定网口名称
elif [[ -n "$VC_MAIN_HOSTS" ]]; then
    # 针对 Lite Cluster CCE 集群平台
    # 获取 RANK_TABLE_FILE 的信息
    RANKTABLE_RESULT=$(python $SHELL_FOLDER/../tools/get_ranktable.py)
    # 将脚本的返回值进行拆分，得到 节点总数量(NNODES) 节点的RANK(NODE_RANK) 单节点的NPU数量(NPUS_PER_NODE)
    IFS=',' read -r NNODES NODE_RANK NPUS_PER_NODE <<< "$RANKTABLE_RESULT"
    MASTER_ADDR="$VC_MAIN_HOSTS"
    MASTER_PORT=6060
    NNODES="$NNODES"
    NODE_RANK="$NODE_RANK"
    NPUS_PER_NODE="$NPUS_PER_NODE"
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
else
    MASTER_ADDR=${MASTER_ADDR:=$1}
    MASTER_PORT=6060
    NNODES=${NNODES:=$2}
    NODE_RANK=${NODE_RANK:=$3}
    NPUS_PER_NODE=${NPUS_PER_NODE:=8}
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
fi
```

2. 在llm_train/AscendSpeed/scripts/tools的路径下，新建脚本文件get_rank_table.py ，具体代码如下所示。

```
import os
import re
import sys
import json

from extras.logging import get_logger, set_file_handler

logger = get_logger(__name__)


def get_rank_table():
    rank_table_file_path = os.getenv("RANK_TABLE_FILE")
    env_ip = os.getenv("ip")

    # Lite Cluster中的RANK_TABLE_FILE实际名称为 jobstart_hccl.json
    job_start_file = "jobstart_hccl.json"
    # job_start_file_path 路径默认为 "/user/config/jobstart_hccl.json"
    job_start_file_path = rank_table_file_path.rsplit("/", 1)[0] + "/" + job_start_file

    # 读取RANK_TABLE_FILE文件
    with open(job_start_file_path, 'r', encoding='utf-8') as file:
        data = json.load(file)

    # RANK_TABLE_FILE文件缺少字段，表示文件缺少关键信息
    if "status" not in data.keys() and "server_count" not in data.keys() and "server_list" not in
data.keys():
        logger.error(f"Get RANK_TABLE Error: RANK_TABLE_FILE missing key value.")
        sys.exit(1)

    # RANK_TABLE_FILE文件status不是completed，表示训练作业未创建成功
    if data["status"] != "completed":
        logger.error(f"Get RANK_TABLE Error: RANK_TABLE_FILE is incomplete, and the training job
creation was not successful.")
        sys.exit(1)

    # 获取：节点总数量:server_count 节点的RANK:server_index 单节点的NPU数量:device_count
    server_count = data["server_count"]
    server_list = data["server_list"]
    server_index = -1
    device_count = 0
    for index, server in enumerate(server_list):
        # RANK_TABLE_FILE文件中，节点ip为空
        if server["server_id"] == "":
```

```
            logger.error(f"Get RANK_TABLE Error: the IP address of the server is null.")
            sys.exit(1)

        if server["server_id"] == env_ip:
            server_index = index
            if server["device"]:
                device_count = len(server["device"])

    # RANK_TABLE_FILE文件中，节点总数量为0，表示未获取到节点
    if server_count == 0:
        logger.error(f"Get RANK_TABLE Error: the server does not exist.")
        sys.exit(1)

    # RANK_TABLE_FILE文件中，未找到对应ip的节点
    if server_index == -1:
        logger.error(f"Get RANK_TABLE Error:  the IP address {env_ip} was not found.")
        sys.exit(1)

    # RANK_TABLE_FILE文件中，NPU卡数为0，表示未获取到NPU
    if device_count == 0:
        logger.error(f"Get RANK_TABLE Error: NPU does not exist.")
        sys.exit(1)

    return server_count, server_index, device_count


if __name__ == '__main__':
    result = get_rank_table()
    print(','.join(map(str, result)))
```

获取模型权重文件

获取对应模型的权重文件，获取链接参考**表4-207**。

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  若要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.910中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
           |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
           |——scripts/             # 训练需要的启动脚本
               |——llama2           # llama2系列模型执行脚本的文件夹
```

```
                |──llama3          # llama3系列模型执行脚本的文件夹
                |──qwen            # Qwen系列模型执行脚本的文件夹
                |──qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                |── ...
                |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |── install.sh       # 环境部署脚本
          |──src/                  # 启动命令行封装脚本，在install.sh里面自动构建
  |──llm_inference                   # 推理代码包
  |──llm_tools                      # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}（例如使用SFS Turbo的路径：/mnt/sfs_turbo/）
|──llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed             # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs            # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora             # lora微调输出权重
            |── sft              # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizers                   #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                       #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                 #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录服务器。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如SFS Turbo的路径：/mnt/sfs_turbo目录下，以下都以/mnt/sfs_turbo为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/mnt/sfs_turbo/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/mnt/sfs_turbo，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /mnt/sfs_turbo
   mkdir -p tokenizers/Llama2-70B
   ```

4.23.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下this格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注他们自己
  ```

```
和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

若用户希望将MOSS数据集的Excel 格式转换为.json格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id：用户的唯一不重复的ID值，必选。
  - excel_addr：待处理的excel文件的地址，必选。
  - dataset_name：处理后的数据集名称，必选。
  - proportion：测试集所占份数，范围[1,9]，可选。
  - test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion和test_count二选一即可，若同时输入，则优先使用test_count，若都未输入，则返回处理失败False。

上传数据到指定目录

将下载的原始数据存放在/mnt/sfs_turbo/training_data目录下。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

```
mkdir training_data
```

数据存放参考目录结构如下：

```
${workdir}
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.23.2.4 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-210 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b

表 4-211 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v   # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择containerd作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

a. 下载并解压buildkit程序。

```
# 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

# 创建解压的目录
mkdir /usr/local/buildkit

# 解压到指定的目录
tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit

# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。

```
[redacted] systemctl status buildkitd
● buildkitd.service - buildkitd
   Loaded: loaded (/usr/lib/systemd/system/buildkitd.service; enabled; vendor p>
   Active: active (running) since Thu 2024-10-17 14:26:30 CST; 2min 16s ago
 Main PID: 3380878 (buildkitd)
    Tasks: 16
   Memory: 47.5M
      CPU: 63ms
   CGroup: /system.slice/buildkitd.service
           └─3380878 /usr/local/buildkit/bin/buildkitd
```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

containerd 容器引擎有命名空间的概念。Kubernetes 下使用的 containerd 默认命名空间是 k8s.io。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。
  ```
  ctr -n k8s.io pull {image_url}
  ```
- 使用nerdctl工具拉取镜像。
  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

⚠ 注意

集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

步骤三 构建 ModelArts Lite 训练镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-209**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.908-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。

   ```
   35 RUN git config --global http.sslVerify false && \
   36     git config --global user.email "you@example.com" &&  \
   37     git config --global user.name "Your Name" &&  \
   38     git clone https://gitee.com/ascend/ModelLink.git && cd ModelLink && git checkout 8f50777 && cd .. && \
   39     git clone https://gitee.com/lmzwhu/Megatron-LM.git  && \
   40     cd Megatron-LM  && \
   41     git checkout -f core_r0.6.0  && \
   42     cp -r megatron ../ModelLink/  && \
   43     cd ..  && \
   44     cd ModelLink && \
   45     pip install -e . && \
   46     cd .. && \
   47     git clone https://gitee.com/ascend/MindSpeed.git && \
   48     cd MindSpeed && \
   49     git checkout 4ea42a23 && \
   50     pip3 install -e . && cd .. && \
   51     pip install -e . && \
   52     pip install transformers==4.45.0
   53     chown -R ma-user:ma-group /home/ma-user/AscendSpeed && \
   54     chown -R ma-user:ma-group /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages && \
   ```

⚠ 注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。

```
nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称> .
```

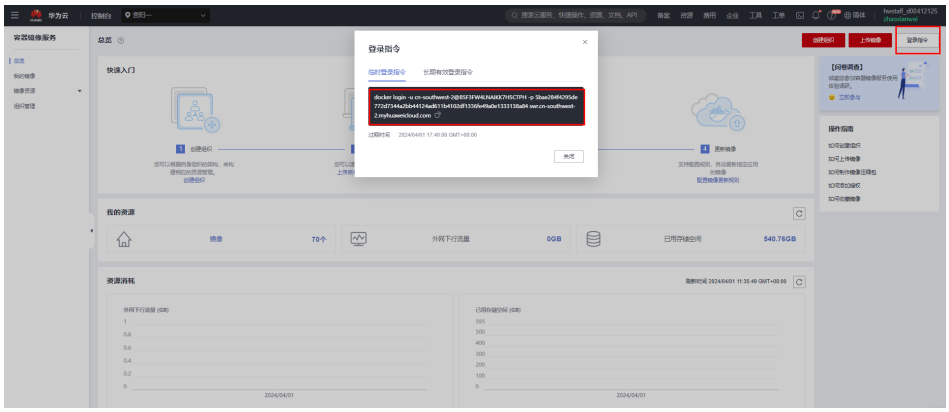
nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以nerdctl pull 拉取测试以下镜像是否能拉取成功。

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤四 在节点机器中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。

图 4-416 复制登录指令



由于使用的容器引擎是containerd，不再是docker，因此需要改写复制的登录指令，将docker进行替换，使用nerdctl工具。

```
# docker login 替换为：
nerdctl login
```

步骤五 修改并上传镜像

1. 在机器中输入Step4登录指令后，使用下列示例命令将镜像上传至SWR：

```
nerdctl --namespace k8s.io tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在**步骤三 构建ModelArts Lite训练镜像**中使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

nerdctl --namespace k8s.io tag ${dockerfile_image_name} swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
nerdctl --namespace k8s.io push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

nerdctl --namespace k8s.io push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

步骤六 编写 Config.yaml 文件

k8s有两种方式来管理对象：

- 命令式，即通过Kubectl指令直接操作对象。
- 声明式，通过定义资源YAML格式的文件来操作对象。

首先给出单个节点训练的config.yaml文件模板，用于配置pod。而在训练中，需要按照参数说明修改**${}**中的参数值。该模板使用SFS Turbo挂载方案。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob              # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                     # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980     # 保持不动
data:                                    # data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                            # job名字，需要和configmap中名字保持联系
  namespace: default                     # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano                 # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
```

```
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
                - key: volcano.sh/job-name
                  operator: In
                  values:
                    - vcjob
            topologyKey: kubernetes.io/hostname
    hostNetwork: true                    # 采用宿主机网络模式
    containers:
    - image: ${image_name}           # 镜像地址
      imagePullPolicy: IfNotPresent  # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
      name: ${container_name}
      securityContext:                       # 容器内 root 权限
        allowPrivilegeEscalation: false
        runAsUser: 0
      env:
      - name: name
        valueFrom:
          fieldRef:
            fieldPath: metadata.name
      - name: ip
        valueFrom:
          fieldRef:
            fieldPath: status.hostIP
      - name: framework
        value: "PyTorch"
      command: ["/bin/sh", "-c"]
      args:
      - ${command}
      resources:
        requests:
          huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
          memory: ${requests_memory}                # 容器请求的最小内存
          cpu: ${requests_cpu}                    # 容器请求的最小 CPU
        limits:
          huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
          memory: ${limits_memory}                 # 容器可使用的最大内存
          cpu: ${limits_cpu}                     # 容器可使用的最大 CPU
      volumeMounts:                          # 容器内部映射路径
      - name: shared-memory-volume
        mountPath: /dev/shm
      - name: ascend-driver                    # 驱动挂载，保持不动
        mountPath: /usr/local/Ascend/driver
      - name: ascend-add-ons                   # 驱动挂载，保持不动
        mountPath: /usr/local/Ascend/add-ons
      - name: localtime
        mountPath: /etc/localtime
      - name: hccn                           # 驱动hccn配置，保持不动
        mountPath: /etc/hccn.conf
      - name: npu-smi                          # npu-smi
        mountPath: /usr/local/sbin/npu-smi
      - name: ascend-install
        mountPath: /etc/ascend_install.info
      - name: log
        mountPath: /var/log/npu/
      - name: sfs-volume
        mountPath: /mnt/sfs_turbo
    nodeSelector:
      accelerator/huawei-npu: ascend-1980
    volumes:                                 # 物理机外部路径
    - name: shared-memory-volume                   # 共享内存
      emptyDir:
        medium: Memory
        sizeLimit: "200Gi"
    - name: ascend-driver
```

```
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
      - name: ascend-install
        hostPath:
          path: /etc/ascend_install.info
      - name: log
        hostPath:
          path: /usr/slog
      - name: sfs-volume
        persistentVolumeClaim:
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
```

双个节点训练的config.yaml文件模板，用于实现双机分布式训练。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob     # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                  # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
data:                    #data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                         # job名字，需要和configmap中名字保持联系
  namespace: default                  # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano               # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
```

```
        app: ascendspeed
        ring-controller.cce: ascend-1980  # 保持不动
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: volcano.sh/job-name
                    operator: In
                    values:
                      - vcjob
              topologyKey: kubernetes.io/hostname
      hostNetwork: true                    # 采用宿主机网络模式
      containers:
      - image: ${image_name}         # 镜像地址
        imagePullPolicy: IfNotPresent      # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
        name: ${container_name}
        securityContext:                        # 容器内 root 权限
          allowPrivilegeEscalation: false
          runAsUser: 0
        env:
        - name: name
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ip
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: framework
          value: "PyTorch"
        command: ["/bin/sh", "-c"]
        args:
        - ${command}
        resources:
          requests:
            huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
            memory: ${requests_memory}                # 容器请求的最小内存
            cpu: ${requests_cpu}                    # 容器请求的最小 CPU
          limits:
            huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
            memory: ${limits_memory}                  # 容器可使用的最大内存
            cpu: ${limits_cpu}                      # 容器可使用的最大 CPU
        volumeMounts:                           # 容器内部映射路径
        - name: shared-memory-volume
          mountPath: /dev/shm
        - name: ascend-driver                   # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons                  # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: localtime
          mountPath: /etc/localtime
        - name: hccn                            # 驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: npu-smi                         # npu-smi
          mountPath: /usr/local/sbin/npu-smi
        - name: ascend-install
          mountPath: /etc/ascend_install.info
        - name: log
          mountPath: /var/log/npu/
        - name: sfs-volume
          mountPath: /mnt/sfs_turbo
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      volumes:                                  # 物理机外部路径
      - name: shared-memory-volume                    # 共享内存
        emptyDir:
```

```
          medium: Memory
          sizeLimit: "200Gi"
      - name: ascend-driver
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
      - name: ascend-install
        hostPath:
          path: /etc/ascend_install.info
      - name: log
        hostPath:
          path: /usr/slog
      - name: sfs-volume
        persistentVolumeClaim:
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
  - name: work
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}         # 镜像地址
          imagePullPolicy: IfNotPresent       # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                    # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
          env:
          - name: name
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: ip
            valueFrom:
              fieldRef:
                fieldPath: status.hostIP
          - name: framework
            value: "PyTorch"
          command: ["/bin/sh", "-c"]
          args:
          - ${command}
```

```
    resources:
      requests:
        huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
        memory: ${requests_memory}                # 容器请求的最小内存
        cpu: ${requests_cpu}                      # 容器请求的最小 CPU
      limits:
        huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
        memory: ${limits_memory}                  # 容器可使用的最大内存
        cpu: ${limits_cpu}                        # 容器可使用的最大 CPU
    volumeMounts:                                 # 容器内部映射路径
    - name: shared-memory-volume
      mountPath: /dev/shm
    - name: ascend-driver                         # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons                        # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                                  # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                               # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                        # 物理机外部路径
  - name: shared-memory-volume                    # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
      claimName: ${pvc_name}    #已创建的PVC名称
  restartPolicy: OnFailure
```

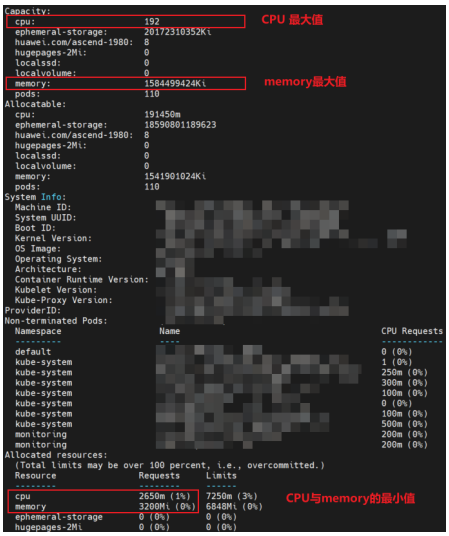
参数说明：

- ${container_name} 容器名称，此处可以自己定义一个容器名称，例如ascendspeed。
- ${image_name} 为**步骤五 修改并上传镜像**中，上传至SWR上的镜像链接。

- ${command} 使用config.yaml文件创建pod后，在容器内自动运行的命令。在进行训练任务中会给出替换命令。
- /mnt/sfs_turbo 为宿主机中默认挂载SFS Turbo的工作目录，目录下存放着训练所需代码、数据等文件。
  - 同样，/mnt/sfs_turbo 也可以映射至容器中，作为容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。为方便访问两个地址可以相同。
- ${pvc_name} 为在**CCE集群关联SFS Turbo**步骤中创建的PVC名称。
- 在设置容器中需要的CPU与内存大小时，可通过运行以下命令查看申请的节点机器中具体的CPU与内存信息。

```
kubectl describe node
```

  - ${requests_cpu} 指在容器中请求的最小CPU核心数量，可使用Requests中的值，例如2650m。
  - ${requests_memory} 指在容器中请求的最小内存空间大小，可使用Requests中的值，例如3200Mi。
  - ${limits_cpu} 指在容器中可使用的最大CPU核心数量，例如192。
  - ${limits_memory} 指在容器中可使用的最大内存空间大小，例如换算成1500Gi。



4.23.3 预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-212所示。其他超参均有默认值，可以参考表4-215按照实际需求修改。

表 4-212 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 修改 config.yaml 中的${command}

请根据**步骤二 修改训练超参配置**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
  - name: work1
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
  - name: work2
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
```

```
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
- name: work3
  template:
    ...
    spec:
      ...
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_13b.sh localhost 1 0; # 单机训练执行命令
```

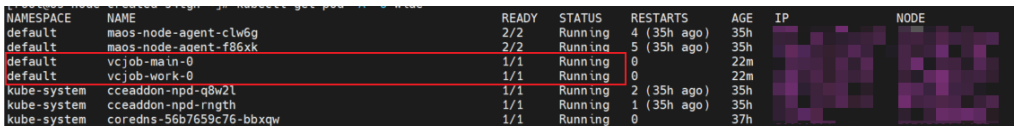
步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```



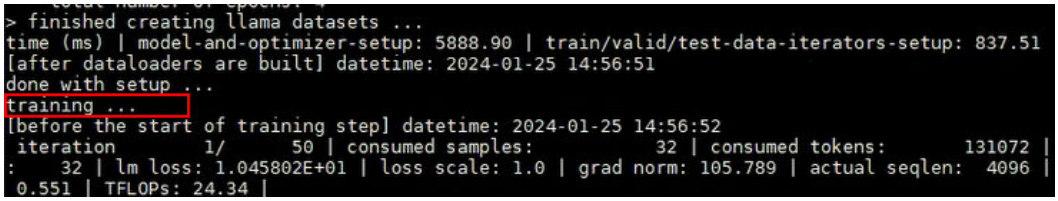
若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-417 等待模型载入



训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.23.4 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-213**所示。其他超参均有默认值，可以参考**表4-215**按照实际需求修改。

表 4-213 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据表4-213修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g               2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk               2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                        1/1     Running   0             22m
default       vcjob-work-0                        1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                  1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                  1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw            1/1     Running   0             37h
```

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-418 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/       50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看sft微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.23.5 LoRA 微调训练

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-214**所示。其他超参均有默认值，可以参考**表4-215**按照实际需求修改。

表 4-214 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**表4-214**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g               2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk               2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                        1/1     Running   0             22m
default       vcjob-work-0                        1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                  1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                  1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw            1/1     Running   0             37h
```

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-419 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看lora微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.23.6 查看日志和性能

查看日志

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

训练过程中，训练日志会在最后的Rank节点打印。

图 4-420 打印训练日志

```
[before the start of training step] datetime: 2023-12-07 10:46:49
 iteration        1/      20 | consumed samples:           32 | consumed tokens:       131072 | elapsed time per iteration (ms): 97720.8 | learning rate: 4.687E-08 | global batch size:    32 | lm loss: 1.118024E+01 | loss scale: 1.0 | g
rad norm: 39.329 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 0.327 | TFLOPs: 7.66 |
[Rank 6] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0[Rank 4] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max alloca
ted: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0

[Rank 7] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
time (ms)
[Rank 0] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 5] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 1] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 3] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 2] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
 iteration        2/      20 | consumed samples:           64 | consumed tokens:       262144 | elapsed time per iteration (ms): 14402.9 | learning rate: 9.375E-08 | global batch size:    32 | lm loss: 1.118334E+01 | loss scale: 1.0 | g
rad norm: 39.675 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.222 | TFLOPs: 51.97 |
time (ms)
 iteration        3/      20 | consumed samples:           96 | consumed tokens:       393216 | elapsed time per iteration (ms): 14218.3 | learning rate: 1.406E-07 | global batch size:    32 | lm loss: 1.118030E+01 | loss scale: 1.0 | g
rad norm: 39.757 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.251 | TFLOPs: 52.65 |
time (ms)
 iteration        4/      20 | consumed samples:          128 | consumed tokens:       524288 | elapsed time per iteration (ms): 14315.5 | learning rate: 1.875E-07 | global batch size:    32 | lm loss: 1.117722E+01 | loss scale: 1.0 | g
rad norm: 38.376 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.29 |
time (ms)
 iteration        5/      20 | consumed samples:          160 | consumed tokens:       655360 | elapsed time per iteration (ms): 14324.0 | learning rate: 2.344E-07 | global batch size:    32 | lm loss: 1.116560E+01 | loss scale: 1.0 | g
rad norm: 39.495 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.234 | TFLOPs: 52.26 |
time (ms)
 iteration        6/      20 | consumed samples:          192 | consumed tokens:       786432 | elapsed time per iteration (ms): 14320.2 | learning rate: 2.813E-07 | global batch size:    32 | lm loss: 1.117150E+01 | loss scale: 1.0 | g
rad norm: 39.782 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.27 |
time (ms)
 iteration        7/      20 | consumed samples:          224 | consumed tokens:       917504 | elapsed time per iteration (ms): 14233.5 | learning rate: 3.281E-07 | global batch size:    32 | lm loss: 1.114488E+01 | loss scale: 1.0 | g
rad norm: 39.099 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration        8/      20 | consumed samples:          256 | consumed tokens:      1048576 | elapsed time per iteration (ms): 14277.9 | learning rate: 3.750E-07 | global batch size:    32 | lm loss: 1.113013E+01 | loss scale: 1.0 | g
rad norm: 38.475 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.241 | TFLOPs: 52.43 |
time (ms)
 iteration        9/      20 | consumed samples:          288 | consumed tokens:      1179648 | elapsed time per iteration (ms): 14206.6 | learning rate: 4.219E-07 | global batch size:    32 | lm loss: 1.103702E+01 | loss scale: 1.0 | g
rad norm: 38.657 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.252 | TFLOPs: 52.69 |
time (ms)
 iteration       10/      20 | consumed samples:          320 | consumed tokens:      1310720 | elapsed time per iteration (ms): 14233.1 | learning rate: 4.687E-07 | global batch size:    32 | lm loss: 1.100142E+01 | loss scale: 1.0 | g
rad norm: 39.465 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration       11/      20 | consumed samples:          352 | consumed tokens:      1441792 | elapsed time per iteration (ms): 14201.2 | learning rate: 5.156E-07 | global batch size:    32 | lm loss: 1.070105E+01 | loss scale: 1.0 | g
rad norm: 40.360 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.253 | TFLOPs: 52.71 |
```

训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

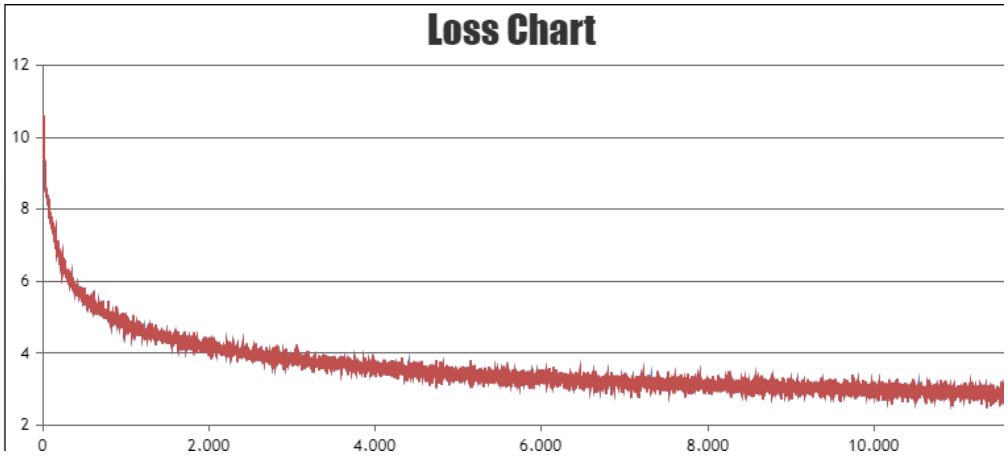
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-215**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-421**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 4-421 Loss 收敛情况（示意图）



4.23.7 训练脚本说明

4.23.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 4-215 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。若训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。

参数	示例值	参数说明
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-216所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-216 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
1	llama 2	llama2-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
2		llama2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
3		llama2-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
4	llama 3	llama3-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
5		llama3-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
6	Qwen	qwen-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
7		qwen-14b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
8		qwen-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
10		qwen1.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
11		qwen1.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
12		qwen1.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
13	Yi	yi-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
14		yi-34b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	1	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
15	ChatGLMv3	glm3-6b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1	2*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
18		qwen2-1.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
19		qwen2-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
20		qwen2-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
22	mistral	mistral-7b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
23	mixtral	mixtral-8x7b	pretrain/sft	4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	1	2*节点 & 8*Ascend
24	llama 3.1	llama3.1-8b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
25		llama3.1-70b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
26	Qwen 2.5	qwen2.5-0.5b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
27		qwen2.5-7b	pretrain/sft	4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	1	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=2	2	1*节点 & 8*Ascend
28		qwen2.5-14b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	2	1*节点 & 8*Ascend
29		qwen2.5-32b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	4	2*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	1	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2	2*节点 & 8*Ascend
30		qwen2.5-72b	pretrain/sft	4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	1	4*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4	4*节点 & 8*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	1	8*节点 & 8*Ascend
			lora		TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	2	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
31	llama 3.2	llama3.2-1b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend
32		llama3.2-3b	pretrain/sft	4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	2	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	训练策略类型	文本序列长度（SEQ_LEN）	并行参数设置	micro batch size (MBS)	规格与节点数
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	2	1*节点 & 4*Ascend
			pretrain/sft	8192	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=2	1	1*节点 & 4*Ascend
			lora		TP(tensor model parallel size)=1 PP(pipeline model parallel size)=1	1	1*节点 & 4*Ascend

4.23.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。

- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
```

```
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

- 先调用self.get_tokenized_data()对数据集进行encode
- self.get_tokenized_data()中调用self._filter方法处理每一个sample
- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

  - 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

    ```
    [
      {"text": "document"},
      {"other keys": "optional content"}
    ]
    ```

  - 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

    ```
    doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
    ```

  - 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
```

```
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {instruction} + "\n" + {input}

  "### Response:"
  {output}
  ```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {instruction}

  "### Response:"
  ```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
```

```
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

   i. 对user和assistant的文本进行清洗

   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

   iii. input_ids是user_ids和assistant_ids的拼接

   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
```

```
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  {MOSS}
  ```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  ```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/1_preprocess_data.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-217 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.23.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-218 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.23.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-422所示。

图 4-422 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-423所示。

图 4-423 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-424 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-425所示。

图 4-425 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-426 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下代码信息进行查找，修改后如图4-427所示。

图 4-427 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.23.8 常见错误原因和解决方法

4.23.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-216**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.23.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.23.8.3 工作负载 Pod 异常

Pod 状态为 Pending

当Pod状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。具体参考链接为**工作负载状态异常定位方法**。

```
NAMESPACE     NAME                                  READY   STATUS    RESTARTS      AGE   IP            NODE          NOMINATED NODE   READINESS GATES
default       maos-node-agent-clw6g                 2/2     Running   4 (35h ago)   35h   [redacted]                  <none>           <none>
default       maos-node-agent-f86xk                 2/2     Running   5 (35h ago)   35h   [redacted]                  <none>           <none>
default       vcjob-main-0                          0/1     Pending   0             6s    <none>        <none>        <none>           <none>
default       vcjob-work-0                          0/1     Pending   0             5s    <none>        <none>        <none>           <none>
kube-system   cceaddon-npd-q8w2l                    1/1     Running   2 (35h ago)   35h   [redacted]                  <none>           <none>
kube-system   cceaddon-npd-rngth                    1/1     Running   1 (35h ago)   35h   [redacted]                  <none>           <none>
```

通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。

```
Events:
  Type     Reason            Age   From     Message
  ----     ------            ----  ----     -------
  Warning  FailedScheduling  26s   volcano  0/2 nodes are unavailable: 2 GPU topology evaluator provider davinci assignment gives no assignment.
```

解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```
3. 若重启后，还是会Pending，建议多重复重启几次。

```
kube-system   volcano-admission-7c8f9fb8f5-6k8k4          1/1   Running   0             35h   [redacted]   <none>   <none>
kube-system   volcano-admission-7c8f9fb8f5-xvt8d          1/1   Running   3 (35h ago)   35h   [redacted]   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-rc8c4         1/1   Running   0             35h   [redacted]   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-vmz75         1/1   Running   1 (35h ago)   35h   [redacted]   <none>   <none>
kube-system   volcano-scheduler-c48c5c87-6kp48            1/1   Running   0             154m  [redacted]   <none>   <none>
kube-system   volcano-scheduler-c48c5c87-pqsh7            1/1   Running   0             175m  [redacted]   <none>   <none>
monitoring    kube-state-metrics-5b5498fccb-tqpzl         1/1   Running   3 (35h ago)   37h   [redacted]   <none>   <none>
monitoring    log-agent-fluent-bit-ls6fp                  2/2   Running   0             35h   [redacted]   <none>   <none>
monitoring    log-agent-fluent-bit-pb7tv                  2/2   Running   0             35h   [redacted]   <none>   <none>
```

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

如何删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.24 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.909）

4.24.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc3。
- Server驱动版本要求23.0.6。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-219 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	cann_8.0.rc3

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-220**所示。

表 4-220 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-221**所示。

表 4-221 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
39	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
42	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
43	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
44	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
45	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
46	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
47	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
48	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)
49	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
50	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
51	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
       ├── build.sh               # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile             # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
   ├──AutoSmoothQuant  # W8A8量化工具
       ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
       ├── autosmoothquant_ascend          # 量化代码
       ├── build.sh                    # 安装量化模块的脚本
   ├──AutoAWQ            # W4A16量化工具
       ├──convert_awq_to_npu.py   # awq权重转换脚本
       ├──quantize.py   # 昇腾适配的量化转换脚本
       ├──build.sh      # 安装量化模块的脚本
 ├──llm_evaluation     # 推理评测代码包
   ├──benchmark_tools    #性能评测
       ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
       ├── benchmark_parallel.py   # 评测静态性能脚本
       ├── benchmark_serving.py    # 评测动态性能脚本
       ├── benchmark_utils.py      # 抽离的工具集
```

```
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型基于Lite Server适配PyTorch NPU训练指导**。

4.24.2 部署推理服务

4.24.2.1 非分离部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

什么是非分离部署

全量推理和增量推理在同一节点上进行。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-219**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.909-xxx.zip和算子包AscendCloud-OPP-6.3.909-xxx.zip到主机中，包获取路径请参见**表4-220**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-221**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下。

   ```
   df -h
   ```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.909-xxx.zip和算子包AscendCloud-OPP-6.3.909-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

步骤五 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

步骤六 启动推理服务

1. 进入容器。
   ```
   docker exec -it -u ma-user ${container-name} /bin/bash
   ```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
   ```
   npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```
   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，若希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-428 查询结果

   ```
   +------------------------------------------------------------------------------------------------+
   | npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
   +---------------------------+---------------+----------------------------------------------------+
   | NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
   | Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
   +===========================+===============+====================================================+
   | 4     910B2               | OK            | 91.4        50                0    / 0           |
   | 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
   +---------------------------+---------------+----------------------------------------------------+
   | 5     910B2               | OK            | 92.5        51                0    / 0           |
   | 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
   +---------------------------+---------------+----------------------------------------------------+
   +---------------------------+---------------+----------------------------------------------------+
   | NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
   +===========================+===============+====================================================+
   | 4       0                 | 10915         | python                   | 55400                   |
   +===========================+===============+====================================================+
   | 5       0                 | 21273         | python                   | 55388                   |
   +===========================+===============+====================================================+
   ```

4. 配置环境变量。

```
export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量。
```

5. 若要开启图模式，请配置以下5个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的档位，根据实际情况设置，另外请不要设置档位1(DeepSeek V2 236B W8A8 模型建议最大设置4个档位)
export VLLM_ENGINE_ITERATION_TIMEOUT_S=1500  # 设置vllm请求超时时间(DeepSeek V2 236B W8A8 模型建议调大为6000)
export HCCL_OP_EXPANSION_MODE=AIV  #可选
```

   设置动态分档位后，在PTA模式下不支持接收超过最大档的并发请求，超过后会导致推理服务终止。请将最大档（PTA_TORCHAIR_DECODE_GEAR_LIST参数中设置的最大值）与模型启动时的max-num-seqs保持一致来进行规避。

   在MoE模型和小模型上推荐使用图模式部署，包括mixtral-8x7B、qwen2-57B、deepseek-v2-lite-16B、deepseek-v2-236B-W8A8；和Qwen2-1.5B、Qwen2-0.5B。当前MoE模型图模式启动不支持multi step。

   MoE模型依赖MindSpeed，当使用MoE模型推理时，需提前安装：

```
git clone https://gitee.com/ascend/MindSpeed.git
cd MindSpeed
git checkout a956b907ef3b0787d2a38577eb5b702f5b7e715d #推荐commit
pip install -e .
```

   开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

6. 若要使用eagle投机，配置环境变量，使eagle投机对齐实验室版本实现。目前默认开启此模式，若不开启，目前vllm0.6.0版本与实验室版本权重无法对齐，会导致小模型精度问题。

```
export EAGLE_USE_SAFE_AI_LAB_STYLE=1  # eagle投机对基于 https://github.com/SafeAILab/EAGLE/ 版本实现
```

   如果需要使用eagle投机推理功能，需要进入 lm_tools/spec_decode文件夹，使用convert_eagle_ckpt_to_vllm_compatible.py脚本进行权重转换。转换命令为

```
python convert_eagle_ckpt_to_vllm_compatible.py --base-path 大模型权重地址 --draft-path 小模型权重地址 --base-weight-name 大模型包含lm_head的权重文件名 --draft-weight-name 小模型权重文件名
```

7. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理

8. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务【**推荐，在vllm-0.6.0之后的版本性能更好**】**

  在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

  （1）非多模态

  ```
  python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --host=${docker_ip} \
  --port=8080 \
  --gpu-memory-utilization=0.9 \
  --num-scheduler-steps=8 \
  --trust-remote-code
  ```

  （2）多模态

  当前支持Llava、InternVL、MiniCPM-v2.6模型，具体模型和权重地址参见**表4-221**，推荐显卡数量参见**表4-227**。

  ```
  export VLLM_IMAGE_FETCH_TIMEOUT=100
  export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
  # PYTORCH_NPU_ALLOC_CONF优先设置为expandable_segments:True
  # 如果有涉及虚拟显存相关的报错，可设置为expandable_segments:False
  export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

  python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --chat-template ${chat_template_path} \
  --dtype ${dtype} \
  --host=${docker_ip} \
  --port=${port} \
  --gpu-memory-utilization=0.9 \
  --trust-remote-code
  ```

  多模态推理服务启动模板参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT：图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
  - PYTORCH_NPU_ALLOC_CONF=expandable_segments:True；允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。开启时可能提升模型性能。报错则关闭。
  - --chat-template：对话构建模板，可选参数。如：

    （1）llava chat-template：${vllm_path}/examples/template_llava.jinja

- **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **方式三：多机部署vLLM服务API接口启动服务**（可选）

  当单机显存无法放下模型权重时，可选用该种方式部署；该种部署方式，需要机器在同一个集群，NPU卡之间IP能够ping通方可，具体步骤如下：

  i. 查看卡IP。

  ```
  for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
  ```

  ii. 检查卡之间的网络是否通。

  ```
  # 在另一个节点上执行，29.81.3.172是上一步输出的ipaddr的值
  hccn_tool -i 0 -ping -g address 29.81.3.172
  ```

  iii. 启动Ray集群。

  ```
  # 指定通信网卡，使用ifconfig查看，找到和主机IP一致的网卡名
  export GLOO_SOCKET_IFNAME=enp67s0f5
  export TP_SOCKET_IFNAME=enp67s0f5
  # 指定可使用的卡
  export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
  # 将其中一个节点设为头节点
  ray start --head --num-gpus=8
  # 在其他节点执行
  ray start --address='10.170.22.18:6379' --num-gpus=8
  ```

  - --num-gpus：要跟ASCEND_RT_VISIBLE_DEVICES指定的可用卡数一致。
  - --address：头节点IP+端口号，头节点创建成功后，会有打印。

  iv. 正常启服务即可。

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。若使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --num-scheduler-steps: 默认为1，推荐设置为8。用于mult-step调度。每次调度生成多个token，可以降低时延。开启multi-step后，在流式返回中，会一次返回num-scheduler-steps个token。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。

```
export USE_PREFIX_HIGH_PRECISION_MODE=1
```

- 如果需要使用multi-lora特性；需要在推理服务启动命令中额外添加如下命令。

```
--enable-lora \
--lora-modules lora1=/path/to/lora/adapter1/ lora2=/path/to/lora/adapter2/ \
--max-lora-rank=16 \
--max-loras=32 \
--max-cpu-loras=32
```

  --enable-lora表示开启lora挂载。

  --lora-modules后面添加挂载的lora列表，要求lora地址权重是huggingface格式，当前支持QKV-proj、O-proj、gate_up_proj、down_proj模块的挂载。

  --max-lora-rank表示挂载lora的最大rank数量，支持8、16、32、64。

  --max-loras 表示支持的最大lora个数，最大32。

  --max-cpu-loras要求配置和--max-loras相同。

  发请求时model指定为lora1或者lora2即为LoRA推理。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**步骤三 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --speculative-draft-tensor-parallel-size: 投机模型使用tp数，因为投机模型较小，多卡并行时通信会降低性能，故此处需要设置为1。

- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

步骤七 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-222**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
```

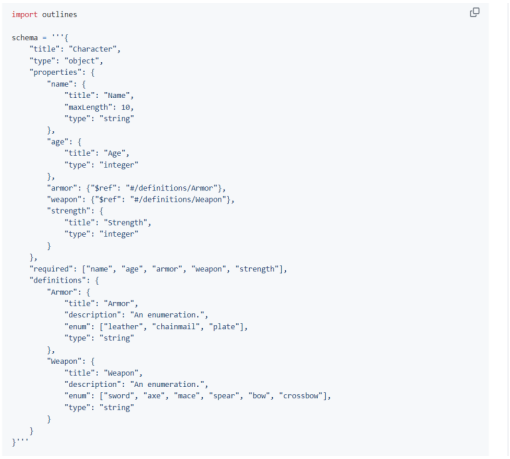
```
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-222 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-429** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

- 方式三 online_serving.py 发送请求（单图单轮对话）

由于多模态推理涉及图片的编解码，所以采用脚本方式调用服务API。脚本中需要配置的参数如表2脚本参数说明所示。

```
import base64
import requests
import argparse
# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    # 必填
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # 选填
    parser.add_argument("--temperature", type=float, default=0) # 输出结果的随机性。可选
    parser.add_argument("--ignore-eos", type=bool, default=False) # 在生成过程中是否忽略结束符号，在生成EOS token后继续生成token。可选
    parser.add_argument("--top-k", type=int, default=-1) # 参数控制着生成结果的多样性。其值越小，生成的文本就越独特，但可能缺乏连贯性。相反，其值越大，文本就越连贯，但多样性也会降低。可选
    parser.add_argument("--top-p", type=int, default=1.0) # 参数的取值范围为0到1。值越小，生成的内容就越意外，但可能牺牲连贯性。值越大，内容就越连贯，但意外性也会减弱。可选
    parser.add_argument("--stream", type=int, default=False) # 是否开启流式推理。默认为False，表示不开启流式推理。
    parser.add_argument("--max_tokens", type=int, default=16) # 生成序列的最大长度。必选
    args = parser.parse_args()
    post_img(args)
```

运行此脚本：

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text 图片内容是什么
```

表 4-223 脚本参数说明

参数	是否必须	参数类型	描述
image_path	是	str	传给模型的图片路径
payload	是	json	单图单轮对话的post请求json， 可参考表2.请求服务json参数说明
docker_ip	是	str	启动多模态openAI服务的主机ip
served_port	是	str	启动多模态openAI服务的端口号

表 4-224 请求服务 json 参数说明

参数	是否必须	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。

参数	是否必须	默认值	参数类型	描述
messages	是	-	Dict	请求输入的问题和图片。`role`: 表示消息的发送者，这里只能为用户。`content`: 表示消息的内容，类型为list。单图单轮对话content必须包含两个元素，第一个元素type字段取值为text，表示文本类型， text字段取值为输入问题的字符串。 第二个元素`type`字段取值为image_url， 表示图片类型，image_url字段取值为是输入图片的base64编码。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

4.24.2.2 分离部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

什么是分离部署

大模型推理是自回归的过程，有以下两阶段：

- Prefill阶段（全量推理）

  将用户请求的prompt传入大模型，进行计算，中间结果写入KVCache并推出第1个token，属于计算密集型。

- Decode阶段（增量推理）

  将请求的前1个token传入大模型，从显存读取前文产生的KVCache再进行计算，属于访存密集型。

分离部署场景下，全量推理和增量推理在不同的容器上进行，用于提高资源利用效率。

分离部署的实例类型启动分为以下三个阶段：

1. **步骤六 启动全量推理实例**：必须为NPU实例，用于启动全量推理服务，负责输入的全量推理。全量推理占用至少1个容器。
2. **步骤七 启动增量推理实例**：必须为NPU实例，用于启动增量推理服务，负责输入的增量推理。增量推理占用至少1个容器。
3. **步骤八 启动scheduler实例**：可为CPU实例，用于启动api-server服务，负责接收推理请求，向全量或增量推理实例分发请求，收集推理结果并向客户端返回推理结果。服务调度实例不占用显卡资源，建议增加1个容器，也可以在全量推理或增量推理的容器上启动。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-219**。

```
docker pull {image_url}
```

步骤三 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.909-xxx.zip和算子包AscendCloud-OPP-6.3.909-xxx.zip到主机中，包获取路径请参见**表4-220**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-221**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

   3.权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：

```
df -h
```

步骤四 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.909-xxx.zip和算子包AscendCloud-OPP-6.3.909-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

步骤五 生成 ranktable

介绍如何生成ranktable，以1p1d-tp2分离部署模式为例。当前1p1d分离部署模式，全量节点和增量节点分别占用2张卡，一共使用4张卡。

- 配置tools工具根目录环境变量

  使用AscendCloud-LLM发布版本进行推理，基于AscendCloud-LLM包的解压路径配置tool工具根目录环境变量：

  ```
  export LLM_TOOLS_PATH=${root_path_of_AscendCloud-LLM}/llm_tools
  ```

  其中，`${root_path_of_AscendCloud-LLM}`为AscendCloud-LLM包解压后的根路径。

  当使用昇腾云的官方指导文档制作推理镜像时，可直接基于该固定路径配置环境变量：

  ```
  export LLM_TOOLS_PATH=/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools
  ```

- 获取每台机器的rank_table

  在每个机器生成global rank_table信息与local rank_table信息。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 4,5 --decode-server-list 6,7 --api-server --save-dir ./save_dir
  ```

  执行后，会生成一个global_ranktable.json文件和使用实例个数的local_ranktable.json文件；如果指定了`--api-server`，还会生成一个local_ranktable_host.json文件用于确定服务入口实例。

  ./save_dir 生成ranktable文件如下（假设本地主机ip为10.**.**.18）。

  ```
  global_ranktable_10.**.**.18.json       # global rank_table
  local_ranktable_10.**.**.18_45.json     # 全量节点local rank_table
  local_ranktable_10.**.**.18_67.json     # 增量节点local rank_table
  local_ranktable_10.**.**.18_host.json   # api-server
  ```

- 合并不同机器的global rank_table(可选)

  如果分离部署在多台机器，获取每台机器的rank_table后，合并各个机器的global rank_table得到完整的global rank_table。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode merge --global-ranktable-list ./ranktable/global_ranktable_0.0,0,0.json ./ranktable/global_ranktable_1.1.1.1.json --save-dir ./save_dir
  ```

  pd_ranktable_tools.py的入参说明如下。

- --mode：脚本的处理模式，可选值为gen或者merge。gen模式表示生成rank_table文件，merge模式表示合并global rank_table文件。
- --save-dir：保存生成的rank_table文件的根目录，默认为当前目录。
- --api-server：仅在`gen`模式有效，可选输入，当存在该输入时，表示分离部署的服务入口在该机器。注意，在多台机器启动分离部署时，只能有一台机器存在服务入口。当存在该输入时，会生成local_ranktable_xx_host.json文件，用于在启动推理服务时确定服务入口实例。
- --prefill-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm全量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应全量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定全量实例。
- --decode-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm增量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应增量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定增量实例。
- --global-ranktable-list：仅在`merge`模式有效，必选输入，后续入参表示需要合并的global rank_table，使用空格分隔开。

执行后，会生成完成合并的global_ranktable_merge.json文件。

- global_rank_table.json格式说明

  server_group_list的长度必须为3，第一个元素(group_id="0")代表Scheduler实例的ip信息，只能有一个实例。

  第二个元素(group_id="1")代表全量实例信息，长度即为全量实例个数。其中需要配置每个全量实例的ip信息以及使用的device信息。rank_id为逻辑卡号，必然从0开始计算，device_id为物理卡号，device_ip则通过上面的hccn_tool获取。

  第三个元素(group_id="2")代表增量实例信息，长度即为增量实例个数。其余信息和全量类似。

  global_rank_table.json具体示例如下：

```
{
   "version": "1.0",
   "status": "completed",
   "server_group_list": [
      {
         "group_id": "0",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost"
            }
         ]
      },
      {
         "group_id": "1",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "4",
                     "device_ip": "10.**.**.22",
                     "rank_id": "0"
                  },
```

```
                {
                   "device_id": "5",
                   "device_ip": "10.**.**.23",
                   "rank_id": "1"
                }
             ]
          }
       ]
    },
    {
       "group_id": "2",
       "server_count": "1",
       "server_list": [
          {
             "server_id": "localhost",
             "server_ip": "localhost",
             "device": [
                {
                   "device_id": "6",
                   "device_ip": "29.**.**.56",
                   "rank_id": "0"
                },
                {
                   "device_id": "7",
                   "device_ip": "29.**.**.72",
                   "rank_id": "1"
                }
             ]
          }
       ]
    }
  ]
}
```

- local_rank_table.json格式说明

  每个全量/增量实例都需要local_rank_table.json。下面以某一个增量实例为例，需要和global_rank_table.json中的增量信息完全对应，group_id为0。

```
{
   "version": "1.0",
   "status": "completed",
   "group_id": "0",
   "server_count": "1",
   "server_list": [
      {
         "server_id": "localhost",
         "server_ip": "localhost",
         "device": [
            {
               "device_id": "6",
               "device_ip": "29.**.**.56",
               "rank_id": "0"
            },
            {
               "device_id": "7",
               "device_ip": "29.**.**.72",
               "rank_id": "1"
            }
         ]
      }
   ]
}
```

步骤六 启动全量推理实例

以下介绍如何启动全量推理实例。

1. 启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci4、davinci5。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动全量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_45.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8088 \
   --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。

- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量或增量推理实例启动的--port参数相关。--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(`--port`)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

参数定义和使用方式与vLLM0.5.0版本一致，此处介绍关键参数。详细参数解释请参见**https://github.com/vllm-project/vllm/blob/main/vllm/engine/arg_utils.py**。

步骤七 启动增量推理实例

1. 启动增量推理容器

   启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
```

```
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci6、davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动增量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_67.json

export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8089 \
   --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用,（英文逗号）分隔开作为该环境变量的输入。

- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP地址
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

步骤八 启动 scheduler 实例

建议在PD服务（即全量推理和增量推理服务）启动后，再启动scheduler服务。

1. 启动scheduler容器。启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了0张卡。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动scheduler实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=9000 \
    --served-model-name ${served-model-name}

# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。当前端口9000是对外服务端口，而8088、8089则为scheduler调度推理服务端口。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下，

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号。分离部署对外服务使用的是scheduler实例端口，在后续推理性能测试和精度测试时，服务端口需要和scheduler实例端口保持一致。
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192

- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量USE_OPENAI=1时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

说明

- 全量和增量节点的local rank table必须一一对应。
- 全量和增量节点不能使用同一个端口。
- scheduler实例中NODE_PORTS=8088,8089；端口设置顺序必须与global rank table文件中各全量和增量节点顺序一致，否则会报错。

步骤九 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-225**。

通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

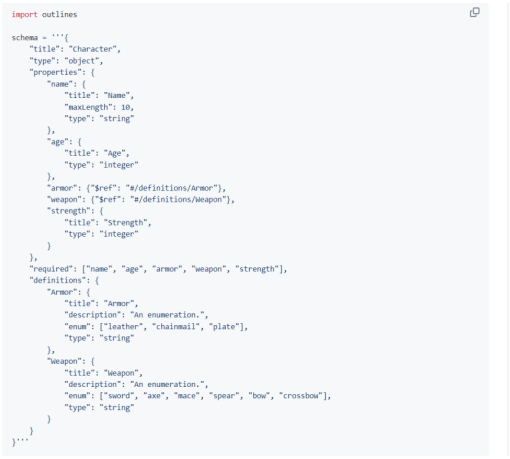
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-225 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。

参数	是否必选	默认值	参数类型	描述
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。

参数	是否必选	默认值	参数类型	描述
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必需属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-430** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.24.3 推理性能测试

4.24.3.1 语言模型推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
   |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt       # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。

   ```
   cd benchmark_tools
   ```

语言模型脚本相对路径是tools/llm_evaluation/benchmark_tools/benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_parallel.py --backend openai --host ${docker_ip} --port ${port} --tokenizer /path/to/tokenizer  --epochs 5 --num-scheduler-steps 8 \
--parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等后端。本文档使用的推理接口是openai。
- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。

3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-431 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	208.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。
   - --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

   ```
   cd benchmark_tools
   conda activate python-3.9.10
   ```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   ```
   python benchmark_serving.py --backend vllm --host ${docker_ip} --port 8080 --dataset
   custom_datasets.json --dataset-type custom \
   --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
   1000 1000 1000 \
   --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。

- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-432 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

投机推理 benchmark 验证

本章节介绍如何进行投机推理benchmark验证。

1. 已经上传投机推理benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，切换一个conda环境。
```
cd benchmark_tools
conda activate python-3.9.10
```
3. 运行验证脚本speculative_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
```
python speculative_benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --dataset human-eval-v2-20210705.jsonl \ --tokenizer /path/to/tokenizer  --num-prompts 80 \
--output_len 4096 --trust-remote-code
```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径，推荐使用human-eval-v2-20210705.jsonl数据集，数据集可从https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz下载压缩包解压获得。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。

- --num-prompts：某个频率下请求数，默认80。
- --output_len：输出长度，默认是1024。
- --trust-remote-code：是否相信远程代码。

a. 脚本运行完后，测试结果保存在终端输出。

单条请求性能测试

针对openai的/v1/completions以及/v1/chat/completions两个非流式接口，请求体中可以添加可选参数"return_latency"，默认为false，若指定该参数为true，则会在相应请求的返回体中返回字段"latency"，返回内容如下：

1. prefill_latency（首token时延）：请求从到达服务开始到生成首token的耗时
2. model_prefill_latency（模型计算首token时延）：服务从开始计算首token到生成首token的耗时
3. avg_decode_latency（平均增量token时延）：服务计算增量token的平均耗时
4. time_in_queue（请求排队时间）：请求从到达服务开始到开始被调度的耗时
5. request_latency（请求总时延）：请求从到达服务开始到结束的耗时

以上指标单位均是ms，保留2位小数。

4.24.3.2 多模态模型推理性能测试

benchmark 方法介绍

静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
|--- modal_benchmark
   |--- modal_benchmark_parallel.py  # modal 评测静态性能脚本
   |--- utils.py
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**步骤四 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，运行静态benchmark验证。
   ```
   cd benchmark_tools
   ```
3. 多模态模型脚本相对路径是llm_tools/llm_evaluation/benchmark_tools/modal_benchmark/modal_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python modal_benchmark_parallel.py \
--host ${docker_ip} \
--port ${port} \
--tokenizer /path/to/tokenizer  \
--epochs 5 \
--parallel-num 1 4 8 16 32 \
--prompt-tokens 1024 2048 \
--output-tokens 128 256 \
--height ${height} \
--width ${width} \
--benchmark-csv benchmark_parallel.csv
```

参数说明

- --host：服务部署的IP，${docker_ip}替换为宿主机实 际的IP地址。
- --port：推理服务端口。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --height: 图片长度（分辨率相关参数）。
- --width: 图片宽度（分辨率相关参数）。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

备注：当前版本仅支持语言+图片多模态性能测试。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中。

4.24.4 推理精度测试

本章节介绍如何使用opencompass工具开展语言模型的推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen。

约束限制

- 确保容器可以访问公网。
- 当前的精度测试仅适用于语言模型精度验证，不适用于多模态模型的精度验证。多模态模型的精度验证，建议使用开源MME数据集和工具（**GitHub - BradyFU/Awesome-Multimodal-Large-Language-Models at Evaluation**）。

步骤一：配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh          #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。
```
conda activate python-3.9.10
bash install.sh
```
3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。
```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
pip install huggingface-hub==0.25.1
```
4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**5**进行评测。
```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```
5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。
```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```
参数说明:
   - vllm_path：构造vllm评测配置脚本名字，默认为vllm。
   - host：与起服务的host保持一致，比如起服务为0.0.0.0，host设置也为0.0.0.0。
   - service_port：服务端口，与启动服务时的端口保持，比如8080。
   - max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
   - eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
   - model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
   - benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

   参考命令：
```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen   model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```


6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size，推理时传入的prompts数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用ppl时必须为True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

   执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

   output_path 指定保存结果的路径。

   参考模型llama3系列模型，数据集mmlu为例，配置如下：

   表 4-226 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu
```

   执行脚本如下

```
# for llama3_8b
python run.py --datasets mmlu_ppl \
--hf-type base --hf-path {hf-path} \
--max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
-w {output_path} --debug
```

参数说明如下：

- --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
- --hf-type：HuggingFace模型权重类型(base,chat)，默认为chat，依据实际的模型选择。
- --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
- --max-seq-len：模型的最大序列长度。
- --max-out-len：模型的最大输出长度。
- --hf-num-gpus：需要使用的卡数。
- --batch-size：推理每次处理的输入数目。
- -w：存放输出结果的目录。

步骤二：查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（ mmlu取值46.6 ）和GPU打分结果（ mmlu取值47 ）进行对比，误差在1以内（ 计算公式：(47-46.6) < 1 ）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远（ 小于10 ）认为分数有效。

4.24.5 推理模型量化

4.24.5.1 使用 AWQ 量化

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-221**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

步骤一 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

AutoAWQ量化工具的适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/AutoAWQ目录下。

1、在容器中使用ma-user用户， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
cd llm_tools/AutoAWQ
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

步骤二 权重格式离线转换（可选）

在GPU上AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.24.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-221**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
   - --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
   - --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**步骤六 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.24.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化和per-tensor+per-head静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-221**。

per-tensor 静态量化场景

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数。

步骤1 使用tensorRT量化工具进行模型量化。

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

步骤2 抽取kv-cache量化系数。

该步骤的目的是将步骤**步骤1**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

步骤3 启动kv-cache-int8-per-tensor量化服务。

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8_pertensor  #只支持int8，表示kvint8 per-tensor量化
--quantization-param-path kv_cache_scales.json  #输入2. 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

----结束

per-tensor+per-head 静态量化场景

如需使用该场景量化方法，请自行准备kv-cache量化系数，格式和per-tensor静态量化所需的2. 抽取kv-cache量化系数生成的json文件一致，只需把每一层的量化系数修改为列表，列表的长度为kv的头数，列表中每一个值代表每一个kv头使用的量化系数。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": [0.09965550899505615, 0.20214074850082397, 0.16
                "1": [0.07757135480642319, 0.15182086825370789, 0.13
```


在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数，启动kv-cache-int8-per-tensor+per-head量化服务。

```
--kv-cache-dtype int8_pertensor_perhead  #只支持int8，表示kvint8 per-tensor+per-head量化
--quantization-param-path kv_cache_scales.json  #输入生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.24.5.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**表4-221**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**步骤六 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.24.5.5 使用 llm-compressor 工具量化

当前版本使用llm-compressor工具量化仅支持Deepseek-v2系列模型的W8A8量化。

本章节介绍如何在GPU的机器上使用开源量化工具**llm-compressor**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   git clone https://github.com/vllm-project/llm-compressor.git
   cd llm-compressor
   pip install -e .
   ```
2. 修改examples/quantizing_moe/deepseek_moe_w8a8_int8.py中的代码：

   1）若本地已有权重，请将MODEL_ID修改为权重路径；
   ```
   MODEL_ID = "deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct"
   ```
   2）若量化Deepseek-V2-236B模型，请将num_gpus改为8；
   ```
   device_map = calculate_offload_device_map(
       MODEL_ID,
       reserve_for_hessians=True,
       num_gpus=8,
       torch_dtype=torch.bfloat16,
       trust_remote_code=True,
   )
   ```
   3）为减少量化时间，建议将以下参数设置为512；
   ```
   NUM_CALIBRATION_SAMPLES = 512
   ```
3. 执行权重量化：
   ```
   python deepseek_moe_w8a8_int8.py
   ```

说明

1、执行权重量化过程中，请保证使用的GPU卡上没有其他进程，否则可能出现OOM；

2、若量化Deepseek-v2-236b模型，大致需要10+小时。

使用量化模型

使用量化模型需要在NPU的机器上运行。

启动vLLM前，请开启图模式（参考**步骤六 启动推理服务**中的配置环境变量），启动服务的命令和启动非量化模型一致。

4.24.6 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-227 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	-	-	8	128
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32
38	falcon-11b	1	8	1	64
39	llava-1.5-7b	1	16	1	32
40	llava-1.5-13b	1	8	1	16
41	llava-v1.6-7b	1	16	1	32
42	llava-v1.6-13b	1	8	1	16
43	llava-v1.6-34b	4	32	2	64
44	internvl2-26b	2	8	1	8
45	MiniCPM-v2.6	2	4	1	32
46	llama-3.1-405B-AWQ	-	-	8	32
47	qwen2-57b-a14b	-	-	2	16
48	mixtral8x7b	4	4	2	16
49	deepseek-v2-lite-16b	2	4	1	4
50	deepseek-v2-236b	-	-	8	4

“-”表示不支持。

4.24.7 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`,

  解决方法：该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启。

4.25 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.909）

4.25.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-228**所示。

表 4-228 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**
3	llama-65b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-65b**
4	llama2-7b	√	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
5	llama2-13b	√	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
30	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
42	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
43	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
44	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
45	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
46	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
47	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
48	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)
49	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
50	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
51	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

操作流程

图 4-433 操作流程图

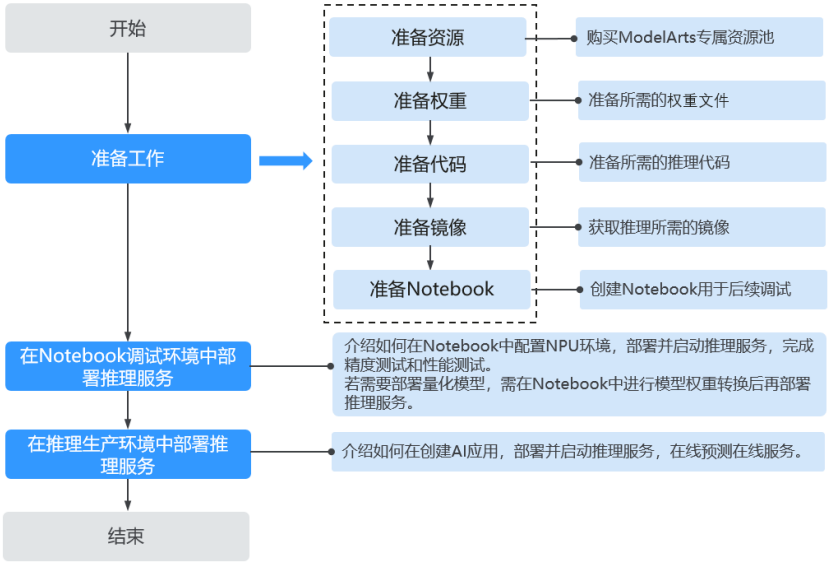


表 4-229 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.909-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 如果需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.25.2 准备工作

4.25.2.1 准备资源

创建专属资源池

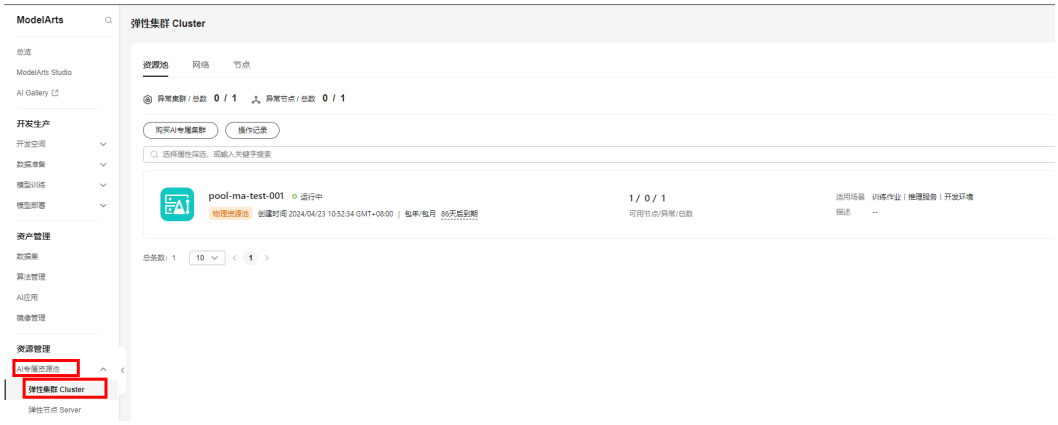
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 4-434 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 4-435 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.25.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.25.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-230**所示。

表 4-230 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
       ├── build.sh                # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile              # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
```

```
├──AutoSmoothQuant  # W8A8量化工具
    ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
    ├── autosmoothquant                 # 量化代码
    ├── build.sh                        # 安装量化模块的脚本
├──AutoAWQ              # W4A16量化工具
    ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──quantize.py   # 昇腾适配的量化转换脚本
    ├──build.sh      # 安装量化模块的脚本
├──llm_evaluation    # 推理评测代码包
  ├──benchmark_tools    #性能评测
    ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
    ├── benchmark_parallel.py   # 评测静态性能脚本
    ├── benchmark_serving.py    # 评测动态性能脚本
    ├── benchmark_utils.py      # 抽离的工具集
    ├── generate_datasets.py    # 生成自定义数据集的脚本
    ├── requirements.txt        # 第三方依赖
  ├──benchmark_eval #精度评测
    ├──opencompass.sh       #运行opencompass脚本
    ├──install.sh           #安装opencompass脚本
    ├──vllm_api.py          #启动vllm api服务器
    ├──vllm.py              #构造vllm评测配置脚本名字
```

4.25.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-231 基础容器镜像地址

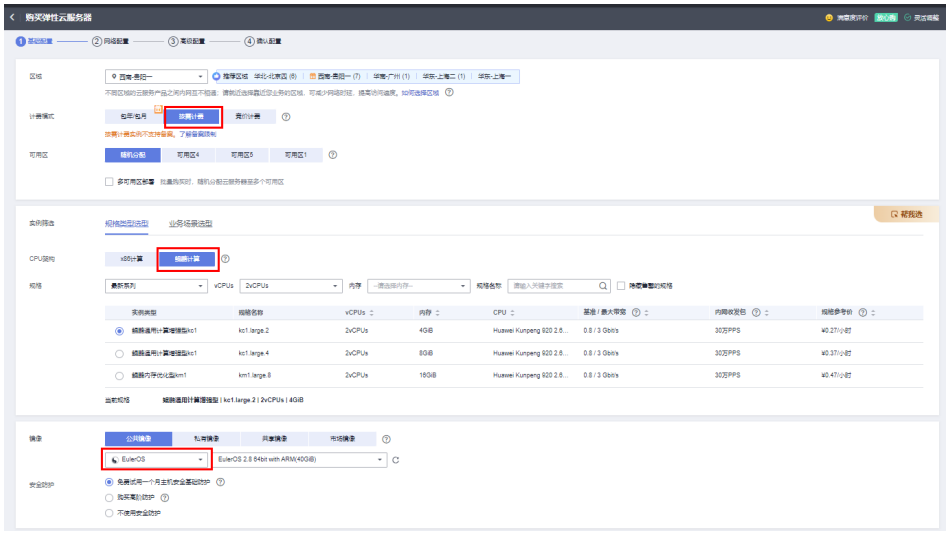
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	CANN：cann_8.0.rc3

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-436 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-437 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-230**。

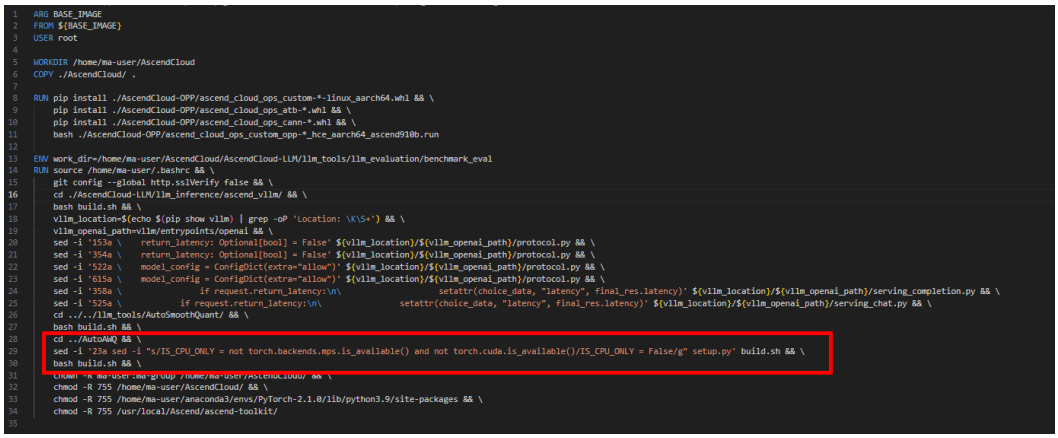
解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```

修改Dockerfile，增加如下命令：

```
RUN source /home/ma-user/.bashrc && \
    ...
    cd ../../llm_tools/AutoSmoothQuant/ && \
    bash build.sh && \
    cd ../AutoAWQ && \
    sed -i '23a sed -i "s/IS_CPU_ONLY = not torch.backends.mps.is_available() and not torch.cuda.is_available()/IS_CPU_ONLY = False/g" setup.py' build.sh && \
    bash build.sh && \
    ...
```

图 4-438 修改 dockerfile



修改build_image.sh内容，将'ENTRYPOINT ["/home/mind/model/run_vllm.sh"]'修改为'ENTRYPOINT sh /home/mind/model/run_vllm.sh'。

图 4-439 修改 build_image.sh

```
#!/bin/bash

OPTIONS=$(getopt -n "$0" -o i:e:n --long base-image:,specify-enrtypoint:,image-name: -- "$@")

if [ $? != 0 ]; then
    echo "args error"
    exit 1
fi

eval set -- "$OPTIONS"

while true; do
    case "$1" in
        -i|--base-image)
            base_image="$2"
            shift 2
            ;;
        -e|--specify-enrtypoint)
            specify_enrtypoint="$2"
            shift 2
            ;;
        -n|--image-name)
            image_name="$2"
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "unknown options"
            exit 1
            ;;
    esac
done

if [ -z "$base_image" ]; then
    echo "--base-image not specified"
    exit 1
fi

if [ -z "$image_name" ]; then
    echo "--image-name not specified"
    exit 1
fi

if [ -n "$specify_enrtypoint" ]; then
    echo 'ENTRYPOINT sh /home/mind/model/run_vllm.sh' >> Dockerfile
fi

cd ../../../../
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile ./
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/.dockerignore ./

docker build -t $image_name --build-arg BASE_IMAGE=$base_image .

rm -f Dockerfile
rm -f .dockerignore

```

执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
sh build_image.sh --base-image=${base_image} --image-name=${image_name} --specify-enrtypoint=True
```

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

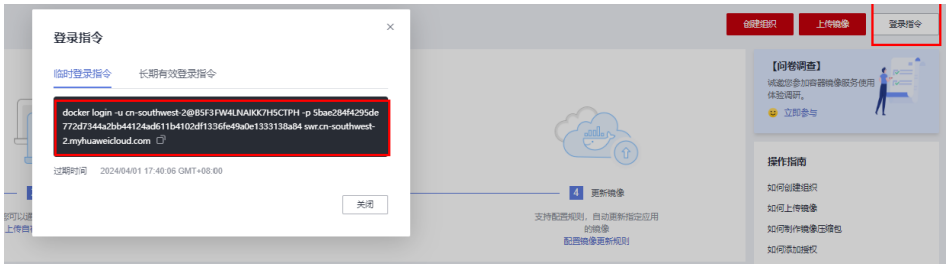
图 4-440 成功构建镜像

```
Step 12/12 : ENTRYPOINT ["/home/mind/model/run_vllm.sh"]
 ---> Running in 3183bafcdaaa
Removing intermediate container 3183bafcdaaa
 ---> 3f8a42ebda99
Successfully built 3f8a42ebda99
Successfully tagged swr.cn-nor                                                -standard
```

Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-441 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示上传镜像成功。

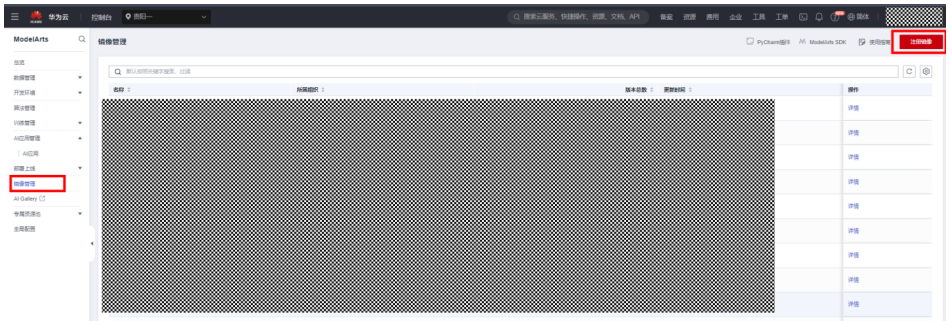
图 4-442 成功上传镜像

```
6.3.905-standard: digest: sha256:1f0b823e0fe6aa0967b                    7cf44e402d5b318b529f7 size: 12710
```

Step8 注册镜像

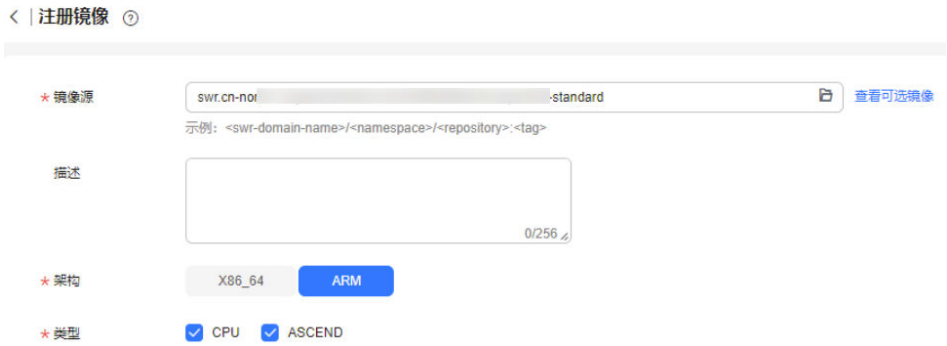
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中单击“注册镜像”。

图 4-443 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 4-444 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.25.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 4-445 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-446 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.25.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

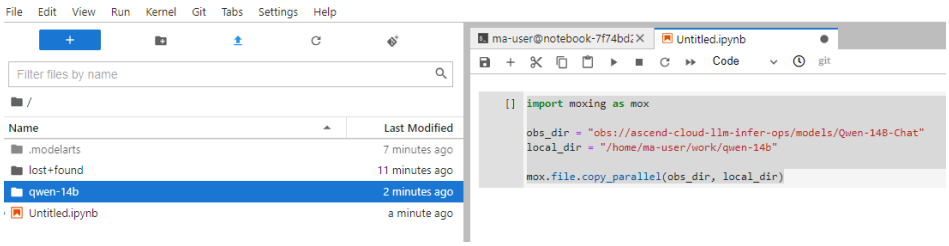
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-447 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-448 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0             |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+---------------------------+---------------+----------------------------------------------------+
| 5     910B2               | OK            | 92.5        51                0    / 0             |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+
+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+---------------------------+---------------+----------------------------------------------------+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

2. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得本次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

若要开启图模式，请配置以下4个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的挡位，根据实际情况设置，另外请不要设置挡位1
export VLLM_ENGINE_ITERATION_TIMEOUT_S=900  # 设置vllm请求超时时间
```

图模式主要针对小模型的场景，可减少算子下发的瓶颈，目前仅针对Qwen2-1.5B进行验证。

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。

4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务**

  在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

  （1）非多模态

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  （2）llava多模态

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--image-input-type pixel_values \
--image-token-id 32000 \
--image-input-shape 1,3,336,336 \
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  多模态推理服务参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
  - PYTORCH_NPU_ALLOC_CONF=expandable_segments:False；llava多卡启动时需要关闭虚拟内存扩展；开启时可能提升模型性能。允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。
  - --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
  - --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

    [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。

- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

```
最小处理单元为14*14
【llava1.5】
336*336图像 ==(336/14=24)>> 24*24=576
672*672图像 ==(672/14=48)>> 48*48=2304
【llava1.6】
336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336
==(336/14=24)>> 336/14+2*24*24=1176
672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336
==(336/14=24)>> 672/14+5*24*24=2928
```

- --chat-template：llava对话构建模板。

– **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

– -model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。

– --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。

– --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。

– --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。

– --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

– --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。

– --block-size：PagedAttention的block大小，推荐设置为128。

– --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。

- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-232**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
```

```
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

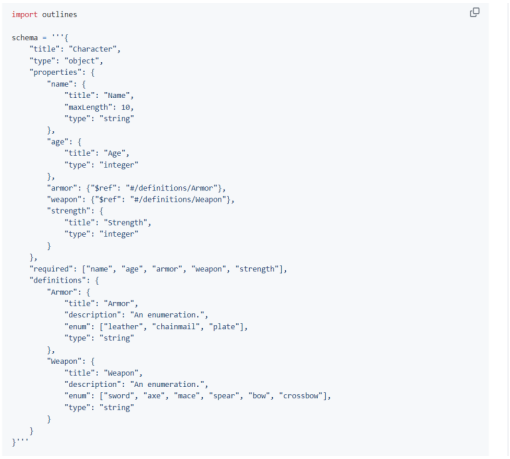
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-232 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。

参数	是否必选	默认值	参数类型	描述
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-449** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.25.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

  说明

  - 如果需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
  - 权重文件夹不要以"model"命名，如果以"model"命名会导致后续创建AI应用报错。
- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-450 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--image-input-type pixel_values \
--image-token-id 32000 \
--image-input-shape 1,3,336,336 \
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  多模态推理服务参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
  - PYTORCH_NPU_ALLOC_CONF=expandable_segments:False；llava多卡启动时需要关闭虚拟内存扩展；开启时可能提升模型性能。允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。
  - --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
  - --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

    [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。

- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

```
最小处理单元为14*14
【llava1.5】
336*336图像 ==(336/14=24)>> 24*24=576
672*672图像 ==(672/14=48)>> 48*48=2304
【llava1.6】
336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336 ==(336/14=24)>> 336/14+2*24*24=1176
672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336 ==(336/14=24)>> 672/14+5*24*24=2928
```

- --chat-template：llava对话构建模板。

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。

- --tensor-parallel-size：并行卡数。此处举例为1，表示使用单卡启动服务。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- -max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 准备模型文件和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得本次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 4-451 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-452 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

图 4-453 创建完成

说明

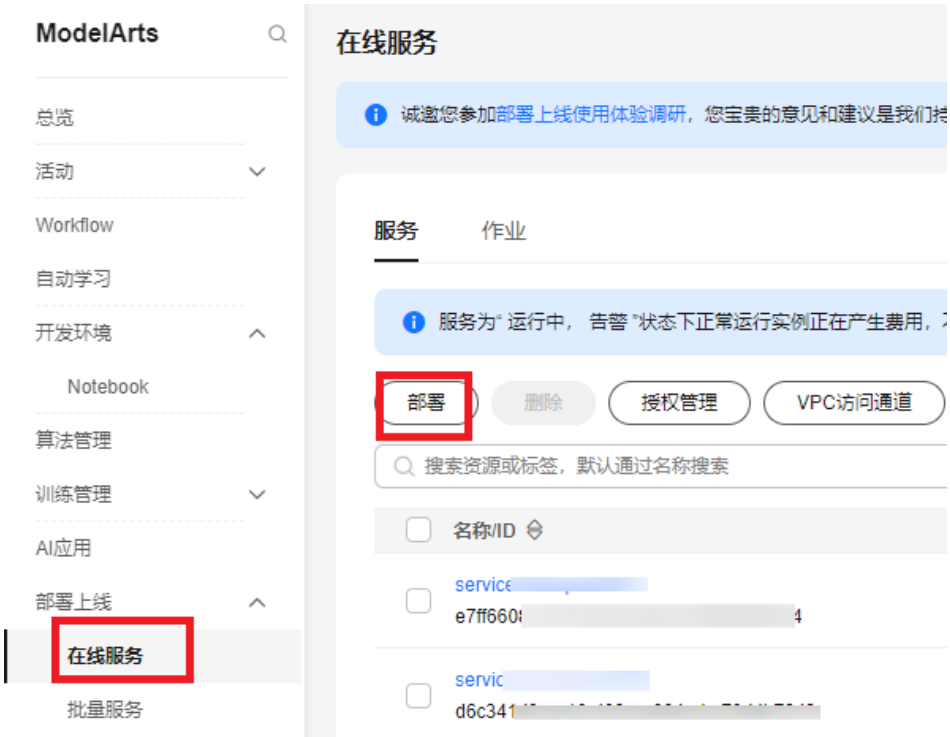
如果权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 4-454 部署在线服务

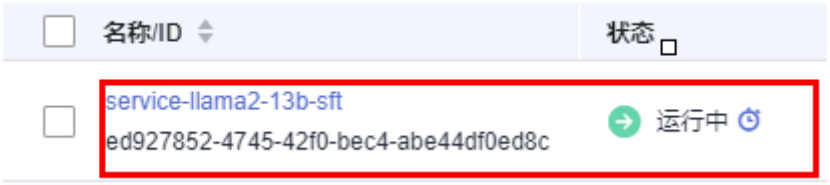


2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-455 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 4-456 服务部署完成

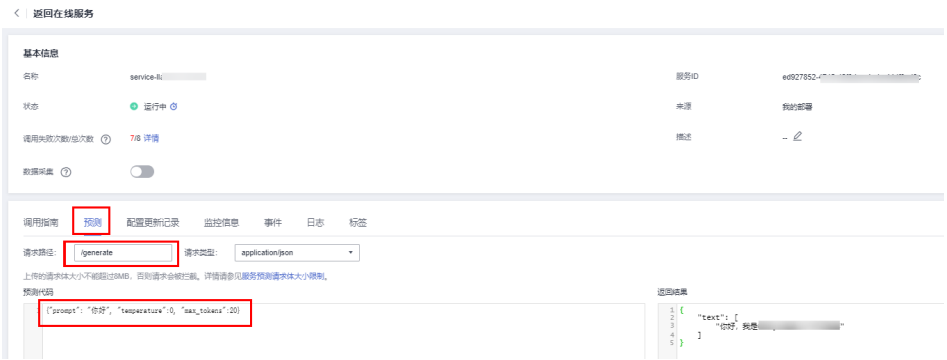


Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

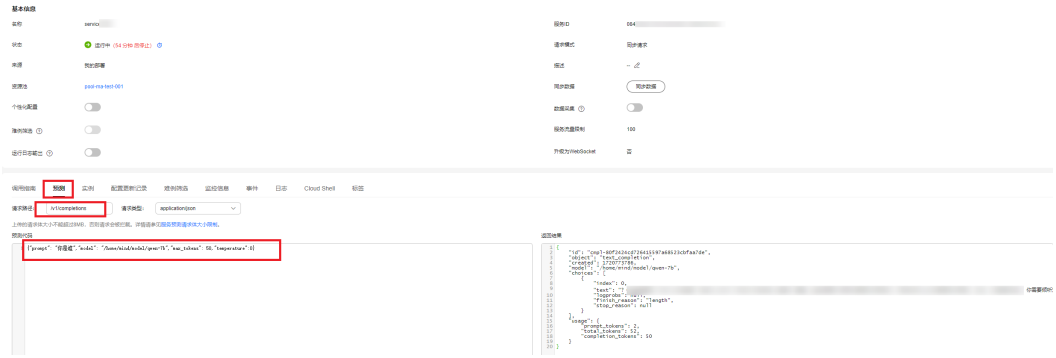
如果以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-457 预测-vllm



如果以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-458 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.25.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
bash install.sh
```

3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤5进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
```

```
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- host：与起服务的host保持一致，比如起服务为0.0.0.0,host设置也为0.0.0.0。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen    model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

fix_id_list, 将最大值适当调低。

ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 4-233 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

```
npu：
mmlu：46.6
gpu：
mmlu：47
```

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

4.25.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。如果需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，如果静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

执行性能测试脚本前，需先安装相关依赖。

```
conda activate python-3.9.10
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

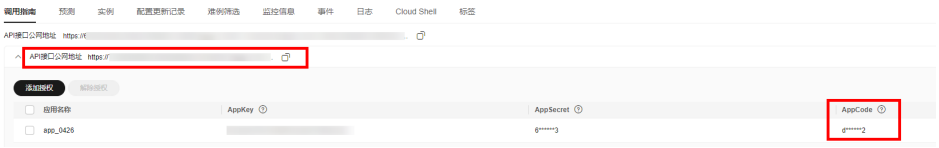
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-459 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。

- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-460 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_dataset.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。

- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

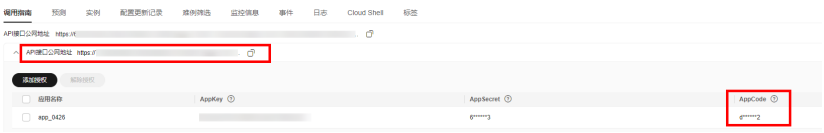
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-461 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
   - --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。

- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-462 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

投机推理 benchmark 验证

本章节介绍如何进行投机推理benchmark验证，当前投机推理benchmark仅支持在Notebook中进行测试。

1. 进入benchmark_tools目录下。
   ```
   cd benchmark_tools
   ```
2. 运行验证脚本speculative_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python speculative_benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --dataset human-eval-v2-20210705.jsonl \ --tokenizer /path/to/tokenizer  --num-prompts 80 \
   --output_len 4096 --trust-remote-code
   ```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径，推荐使用human-eval-v2-20210705.jsonl数据集，数据集可从https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz下载压缩包解压获得。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --num-prompts：某个频率下请求数，默认80。
   - --output_len：输出长度，默认是1024。
   - --trust-remote-code：是否相信远程代码。

   脚本运行完后，测试结果直接在终端输出。

4.25.7 推理模型量化

4.25.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel, W8A16 per-channel

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、在容器中使用ma-user用户， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.25.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。

- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.25.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

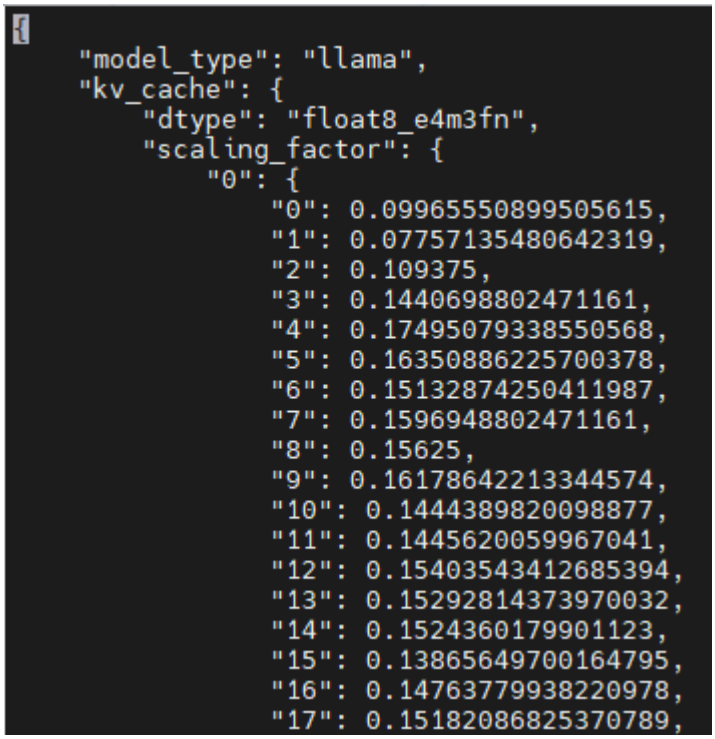
该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-463 抽取 kv-cache 量化系数



注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.25.7.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。

```
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
model_id = "meta-llama/CodeLlama-34b-hf"
tokenizer = AutoTokenizer.from_pretrained(model_id)
gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。

```
dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。

```
quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
quantization_config=gptq_config)
```

5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。

```
quantized_model.save_pretrained("CodeLlama-34b-hf")
tokenizer.save_pretrained("CodeLlama-34b-hf")

# if quantized with device_map set
quantized_model.to("cpu")
quantized_model.save_pretrained("CodeLlama-34b-hf")
```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.25.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-234 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	--		8	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
37	gemma-7b	1	8	1	32
38	falcon-11b	1	8	1	64

4.25.9 附录：Standard 大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

  config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`,

  解决方法：将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.26 主流开源大模型基于 Lite Cluster 适配 PyTorch NPU 推理指导（6.3.909）

4.26.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite k8s Cluster上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Lite k8s Cluster和昇腾Snt9B资源。
- 本文档中的CCE集群版本选择v1.27~1.28。版本使用的容器引擎为**Containerd。**
- 推理部署使用的服务框架是vLLM。vLLM支持v0.6.0版本。
- 支持FP16和BF16数据类型推理。
- Lite k8s Cluster驱动版本推荐为23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

支持的模型列表和权重文件

本方案支持vLLM的v0.6.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-235**所示。

表 4-235 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
38	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	√	√	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
42	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
43	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
44	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
45	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
46	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
47	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)
48	internvl2-26B	√	x	x	x	x	OpenGVLab/InternVL2-26B at main (huggingface.co)

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
49	MiniCPM-v2.6	√	x	x	x	x	https://huggingface.co/openbmb/MiniCPM-V-2_6/tree/main
50	deepseek-v2-236b	x	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2
51	deepseek-v2-lite-16b	√	x	√	x	x	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite

说明

各模型支持的卡数请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**章节。

4.26.2 准备工作

4.26.2.1 准备环境

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Lite k8s Cluster。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**Lite Cluster资源开通**，熟悉集群资源开通流程，再开始操作购买k8s Cluster资源。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

图 4-464 购买 Lite 专属池

k8s Cluster 资源配置

若已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

kubectl 访问集群配置

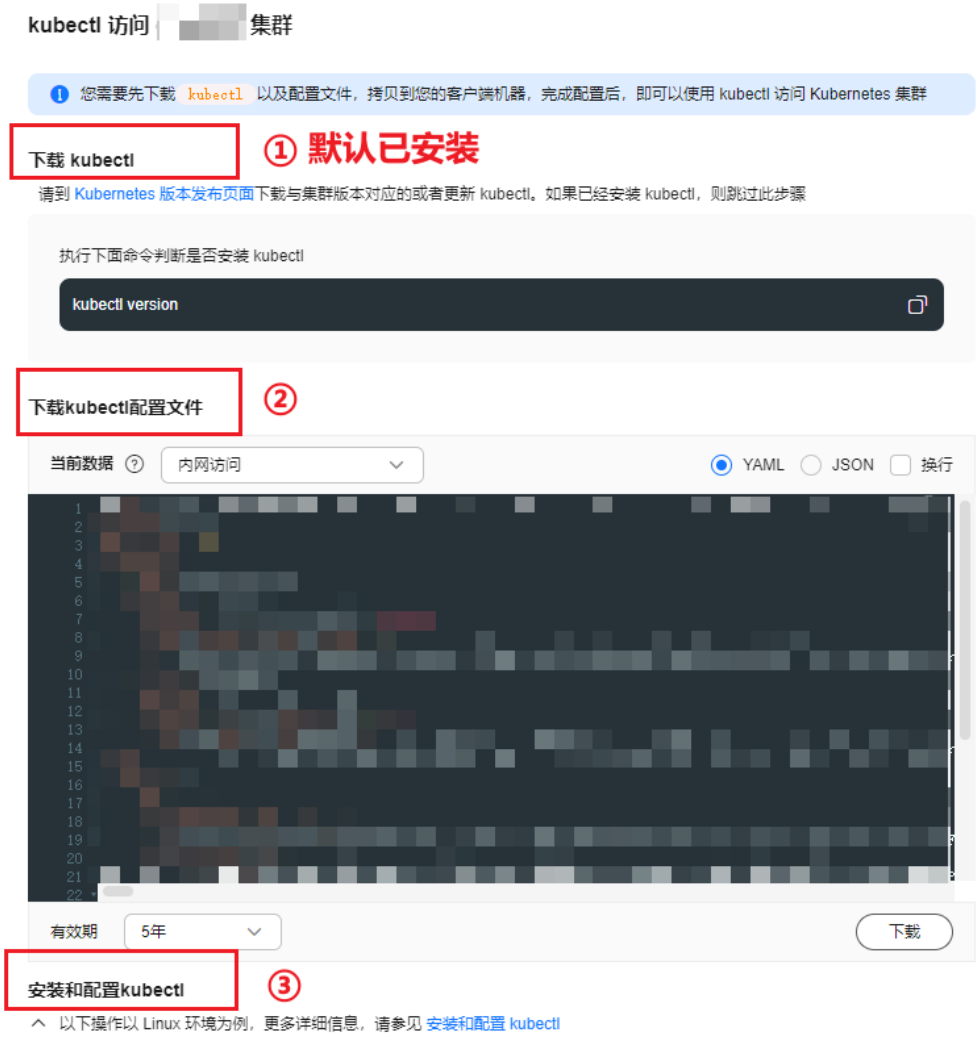
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的CCE集群控制版面中。根据**图4-465**的步骤进行操作，单击kubectl配置时，会弹出**图4-466**步骤页面。

图 4-465 配置中心

2. 根据图4-466，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-466 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。若显示如图**图4-467**的内容，则配置成功。

```
kubectl cluster-info
```

图 4-467 查看 Kubernetes 集群信息正确弹出内容

4.26.2.2 准备代码

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-236**所示。

表 4-236 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.6.0-py3-none-any.whl   # 推理安装包
       ├── build.sh               # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile              # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
   ├──AutoSmoothQuant  # W8A8量化工具
       ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
       ├── autosmoothquant                 # 量化代码
       ├── build.sh                        # 安装量化模块的脚本
   ├──AutoAWQ            # W4A16量化工具
       ├──convert_awq_to_npu.py   # awq权重转换脚本
       ├──quantize.py   # 昇腾适配的量化转换脚本
       ├──build.sh      # 安装量化模块的脚本
   ├──llm_evaluation    # 推理评测代码包
     ├──benchmark_tools    #性能评测
       ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
       ├── benchmark_parallel.py   # 评测静态性能脚本
       ├── benchmark_serving.py    # 评测动态性能脚本
       ├── benchmark_utils.py      # 抽离的工具集
       ├── generate_datasets.py    # 生成自定义数据集的脚本
       ├── requirements.txt        # 第三方依赖
     ├──benchmark_eval #精度评测
       ├──opencompass.sh       #运行opencompass脚本
       ├──install.sh           #安装opencompass脚本
       ├──vllm_api.py          #启动vllm api服务器
       ├──vllm.py            #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考《 主流开源大模型基于Lite Cluster适配PyTorch训练指导 》。

4.26.2.3 准备镜像

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-237 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	cann_8.0.rc3

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v   # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择 containerd 作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

   buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

   buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

   a. 下载并解压buildkit程序。
      ```
      # 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
      wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

      # 创建解压的目录
      mkdir /usr/local/buildkit

      # 解压到指定的目录
      tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit

      # 授予权限
      chmod -R 777 /usr/local/buildkit
      ```
   b. 添加环境变量
      ```
      echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
      # 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
      source /etc/profile  # 使刚才配置生效
      ```

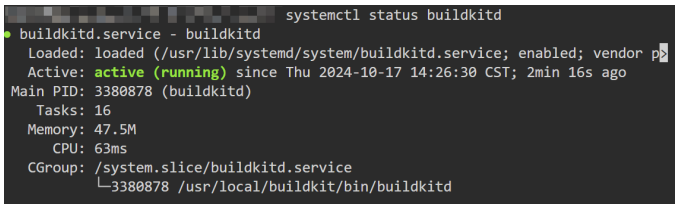
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-237**。

containerd 容器引擎有命名空间的概念。**Kubernetes 下使用的 containerd 默认命名空间是 k8s.io**。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。
  ```
  ctr -n k8s.io images pull {image_url}
  ```
- 使用 nerdctl 工具进行镜像拉取。
  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

注意：集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

Step3 制作推理镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-236**。

1. 解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.909-xxx.zip，并直接进入到llm_inference/ascend_vllm文件夹下面
```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```
2. 执行以下命令制作推理镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。
```
nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称> --build-arg BASE_IMAGE=${base_image} .
```
注意：nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以nerdctl pull拉取测试镜像是否能拉取成功。
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。
   - ${base_image}为基础镜像地址。

4.26.3 部署推理服务

本章节介绍如何使用vLLM 0.6.0框架部署并启动推理服务。

前提条件

- 已准备好Lite k8s Cluster环境，具体参考**准备环境**。推荐使用“西南-贵阳一”Region上的Cluster和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保集群可以访问公网。

Step1 上传权重文件

将权重文件上传到集群节点机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**支持的模型列表和权重文件**。

如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step2 配置 pod

在节点自定义目录${node_path}下创建config.yaml文件
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
    app: infers
spec:
  replicas: 1
  selector:
    matchLabels:
      app: infers
  template:
    metadata:
      labels:
        app: infers
    spec:
      schedulerName: volcano
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      containers:
      - image: ${image_name}                # 推理镜像名称
        imagePullPolicy: IfNotPresent
        name: ${container_name}
```

```
      securityContext:
        runAsUser: 0
      ports:
      - containerPort: 8080
      command: ["/bin/bash", "-c"]
      args: ["${node-path}/run_vllm.sh"]          # 节点自定义目录，该目录下包含pod配置文件config.yaml和推理服务启动脚本run_vllm.sh
      resources:
        requests:
          huawei.com/ascend-1980: "8"             # 需求卡数，key保持不变。
        limits:
          huawei.com/ascend-1980: "8"             # 限制卡数，key保持不变。
      volumeMounts:                               # 容器内部映射路径
      - name: ascend-driver                       #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/driver
      - name: ascend-add-ons                      #驱动挂载，保持不动
        mountPath: /usr/local/Ascend/add-ons
      - name: hccn                                #驱动hccn配置，保持不动
        mountPath: /etc/hccn.conf
      - name: localtime
        mountPath: /etc/localtime
      - name: npu-smi                             # npu-smi
        mountPath: /usr/local/sbin/npu-smi
      - name: model-path                          # 模型权重路径
        mountPath: ${model-path}
      - name: node-path
        mountPath: ${node-path}
    volumes:                                      # 物理机外部路径
    - name: ascend-driver
      hostPath:
        path: /usr/local/Ascend/driver
    - name: ascend-add-ons
      hostPath:
        path: /usr/local/Ascend/add-ons
    - name: hccn
      hostPath:
        path: /etc/hccn.conf
    - name: localtime
      hostPath:
        path: /etc/localtime
    - name: npu-smi
      hostPath:
        path: /usr/local/sbin/npu-smi
    - name: model-path
      hostPath:
        path: ${model-path}
    - name: node-path
      hostPath:
        path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。
- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml和推理服务启动脚本run_vllm.sh，run_vllm.sh内容见**Step3 创建服务启动脚本**。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

Step3 创建服务启动脚本

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

（2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--image-input-type pixel_values \
--image-token-id 32000 \
--image-input-shape 1,3,336,336 \
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
- --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

  [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。
- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

  ```
  最小处理单元为14*14
  【llava1.5】
  336*336图像 ==(336/14=24)>> 24*24=576
  672*672图像 ==(672/14=48)>> 48*48=2304
  【llava1.6】
  336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336 ==(336/14=24)>> 336/14+2*24*24=1176
  672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336 ==(336/14=24)>> 672/14+5*24*24=2928
  ```

- --chat-template：llava对话构建模板。

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：**Step1 上传权重文件**中上传的模型权重路径。
- --tensor-parallel-size：并行卡数。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**表4-240**。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 上传权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

若要开启图模式，请配置以下4个环境变量，并且启动服务时不要添加enforce-eager参数。

```
export INFER_MODE=PTA  # 开启PTA模式，若不使用图模式，请关闭该环境变量
export PTA_TORCHAIR_DECODE_GEAR_ENABLE=1  # 开启动态分档功能
export PTA_TORCHAIR_DECODE_GEAR_LIST=2,4,6,8,16,32  # 设置动态分档的挡位，根据实际情况设置，另外请不要设置挡位1
export VLLM_ENGINE_ITERATION_TIMEOUT_S=900  # 设置vllm请求超时时间
```

图模式主要针对小模型的场景，可减少算子下发的瓶颈，目前仅针对Qwen2-1.5B进行验证。

开启图模式后，服务第一次响应请求时会有一个较长时间的图编译过程，并且会在当前目录下生成.torchair_cache文件夹来保存图编译的缓存文件。当服务第二次启动时，可通过缓存文件来快速完成图编译的过程，避免长时间的等待，并且基于图编译缓存文件来启动服务可获得更优的推理性能，因此请在有图编译缓存文件的前提下启动服务。另外，当启动服务时的模型或者参数发生改变时，请删除.torchair_cache文件夹，避免由于缓存文件与实际推理不匹配而报错。

Step4 创建 pod

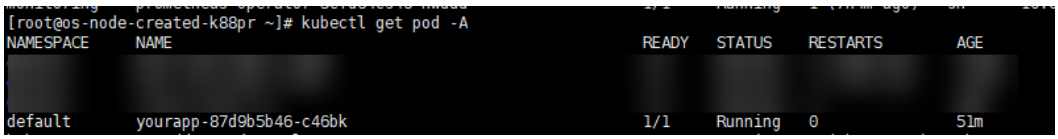
在节点自定义目录${node_path}下执行如下命令创建pod。

```
kubectl apply -f config.yaml
```

检查pod启动情况，执行下述命令。如果显示“1/1 running”状态代表启动成功。

```
kubectl get pod -A
```

图 4-468 启动 pod 成功



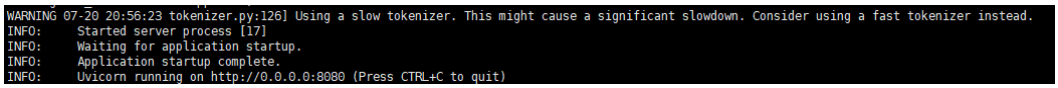
执行如下命令查看pod日志，若打印类似下图信息表示服务启动成功。

```
kubectl logs -f ${pod_name}
```

参数说明：

- ${pod_name}：pod名，例如图4-468${pod_name}为yourapp-87d9b5b46-c46bk。

图 4-469 启动服务成功



Step5 推理请求

执行如下命令进入容器。

```
kubectl exec -it {pod_name} bash
```

参数说明：

- ${pod_name}：pod名，例如图4-468${pod_name}为yourapp-87d9b5b46-c46bk。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见表4-238。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和Step3 创建服务启动脚本中设置的宿主机端口保持一致。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
```

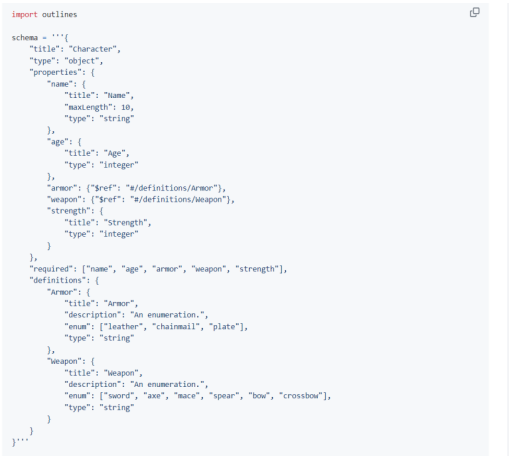
```
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-238 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-470** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.26.4 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step3 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 执行如下命令进入容器。
   ```
   kubectl exec -it {pod_name} bash
   ```
   ${pod_name}：pod名，例如**图4-468**${pod_name}为yourapp-87d9b5b46-c46bk。
3. 进入benchmark_tools目录下，切换conda环境并安装依赖。
   ```
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
   conda activate python-3.9.10
   pip install -r requirements.txt
   ```

4. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
--parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务部署的IP。
- --port：推理服务端口8080。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。
- --enable-prefix-caching：服务端是否启用enable-prefix-caching特性，默认为false。

5. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-471 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	208.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 执行如下命令进入容器。
```
kubectl exec -it {pod_name} bash
```
${pod_name}：pod名，例如图4-468${pod_name}为yourapp-87d9b5b46-c46bk。

2. 进入benchmark_tools目录下，切换conda环境。
```
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
conda activate python-3.9.10
```

3. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   generate_dataset.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。
```
python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```
   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。
   - --num-requests：输出数据集的数量，可以根据实际需求设置。

4. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。
```
python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。

- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。
- --num-scheduler-steps: 需和服务启动时配置的num-scheduler-steps一致。默认为1。

5. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-472 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

投机推理 benchmark 验证

本章节介绍如何进行投机推理benchmark验证。

1. 进入benchmark_tools目录下。
   ```
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_tools
   ```
2. 运行验证脚本speculative_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python speculative_benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --dataset human-eval-v2-20210705.jsonl \ --tokenizer /path/to/tokenizer  --num-prompts 80 \
   --output_len 4096 --trust-remote-code
   ```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径，推荐使用human-eval-v2-20210705.jsonl数据集，数据集可从https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz下载压缩包解压获得。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。

- --num-prompts：某个频率下请求数，默认80。
- --output_len：输出长度，默认是1024。
- --trust-remote-code：是否相信远程代码。

脚本运行完后，测试结果直接在终端输出。

4.26.5 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6。

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 执行如下命令进入容器。

```
kubectl exec -it {pod_name} bash
```

${pod_name}：pod名，例如图4-468${pod_name}为yourapp-87d9b5b46-c46bk。

3. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
bash install.sh
```

4. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下（可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

5. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

6. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- host：与起服务的host保持一致，比如起服务为0.0.0.0,host设置也为0.0.0.0。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen   model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

7. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

8. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应

的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 4-239 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

9. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat)，默认为chat，依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成

的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

```
npu：
mmlu：46.6
gpu：
mmlu：47
```

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

4.26.6 推理模型量化

4.26.6.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel，W8A16 per-channel

Step1 环境准备

1. 在节点自定义目录${node_path}下创建config.yaml文件

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yourapp
  labels:
     app: infers
spec:
  replicas: 1
  selector:
    matchLabels:
      app: infers
  template:
    metadata:
      labels:
        app: infers
    spec:
      schedulerName: volcano
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      containers:
      - image: ${image_name}                # 推理镜像名称
        imagePullPolicy: IfNotPresent
        name: ${container_name}
        securityContext:
          runAsUser: 0
        ports:
        - containerPort: 8080
        command:
        - "sleep"
        - "10000000000000000000"
        resources:
          requests:
            huawei.com/ascend-1980: "8"            # 需求卡数，key保持不变。
          limits:
            huawei.com/ascend-1980: "8"            # 限制卡数，key保持不变。
        volumeMounts:                          # 容器内部映射路径
        - name: ascend-driver                #驱动挂载，保持不动
```

```
      mountPath: /usr/local/Ascend/driver
    - name: ascend-add-ons          #驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: hccn                    #驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: localtime
      mountPath: /etc/localtime
    - name: npu-smi                 # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: model-path              # 模型权重路径
      mountPath: ${model-path}
    - name: node-path               # 节点自定义目录，该目录下包含pod配置文件config.yaml
      mountPath: ${node-path}
  volumes:                          # 物理机外部路径
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: model-path
    hostPath:
      path: ${model-path}
  - name: node-path
    hostPath:
      path: ${node-path}
```

参数说明：

- ${container_name}：容器名称，此处可以自己定义一个容器名称，例如ascend-vllm。
- ${image_name}：**Step3 制作推理镜像**构建的推理镜像名称。
- ${node-path}：节点自定义目录，该目录下包含pod配置文件config.yaml。
- ${model-path}：**Step1 上传权重文件**中上传的模型权重路径。

2. 参考**Step4 创建pod**创建pod以用于后续进行模型量化

Step2 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 执行如下命令进入容器，并进入AutoAWQ目录下， vLLM使用transformers版本与awq冲突，需要切换conda环境，运行以下命令下载并安装AutoAWQ源码。

```
kubectl exec -it {pod_name} bash
conda create --name awq --clone PyTorch-2.1.0
conda activate awq
pip uninstall ascend-vllm vllm transformers
cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoAWQ
bash build.sh
```

2. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step3 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step4 启动 AWQ 量化服务

参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q awq 或者--quantization awq
```

4.26.6.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
    ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
```

```
├── autosmoothquant              # 量化代码
├── build.sh                     # 安装量化模块的脚本
...
```

具体操作如下：

1. 参考**Step1 环境准备**创建pod准备量化环境。
2. 执行如下命令进入容器，并进入AutoSmoothQuant目录下
   ```
   kubectl exec -it {pod_name} bash
   cd /home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/AutoSmoothQuant/autosmoothquant/examples
   ```
3. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，若希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-473 查询结果

   ```
   +------------------------------------------------------------------------------------------------+
   | npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
   +---------------------------+---------------+----------------------------------------------------+
   | NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
   | Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
   +===========================+===============+====================================================+
   | 4     910B2               | OK            | 91.4        50                0    / 0           |
   | 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
   +===========================+===============+====================================================+
   | 5     910B2               | OK            | 92.5        51                0    / 0           |
   | 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
   +===========================+===============+====================================================+
   +---------------------------+---------------+----------------------------------------------------+
   | NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
   +===========================+===============+====================================================+
   | 4       0                 | 10915         | python                   | 55400                   |
   +===========================+===============+====================================================+
   | 5       0                 | 21273         | python                   | 55388                   |
   +===========================+===============+====================================================+
   ```

4. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。

- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

5. 启动smoothQuant量化服务。

   参考**部署推理服务**，使用量化后权重部署AWQ量化服务。

   注：**Step3 创建服务启动脚本**启动脚本中，服务启动命令需添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.26.6.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf    \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

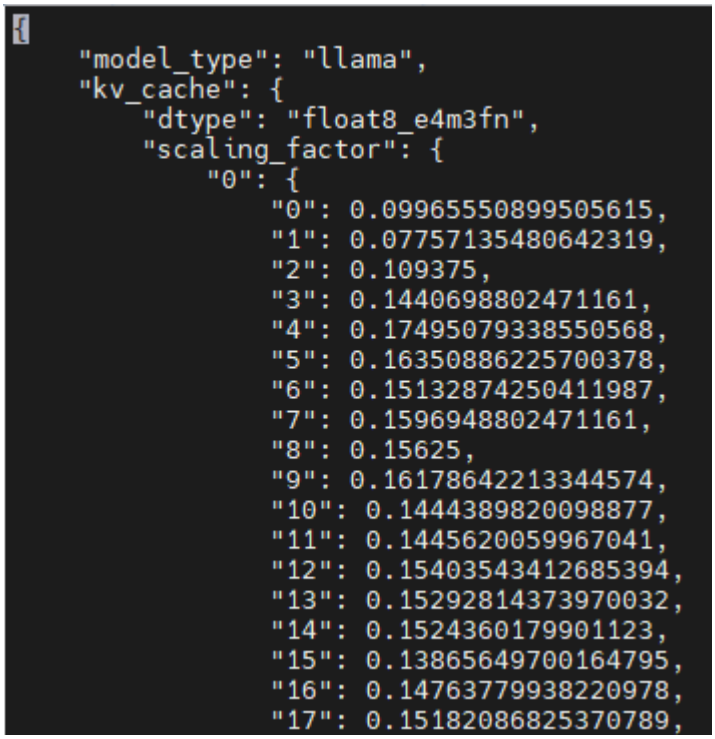
Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：



注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.26.6.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   ```

```
tokenizer = AutoTokenizer.from_pretrained(model_id)
gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。

```
dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```

4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。

```
quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
quantization_config=gptq_config)
```

5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。

```
quantized_model.save_pretrained("CodeLlama-34b-hf")
tokenizer.save_pretrained("CodeLlama-34b-hf")

# if quantized with device_map set
quantized_model.to("cpu")
quantized_model.save_pretrained("CodeLlama-34b-hf")
```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 创建服务启动脚本**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
        max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.26.7 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.6.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-240 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	--		8	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
37	gemma-7b	1	8	1	32
38	falcon-11b	1	8	1	64

4.26.8 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`,

  解决方法：将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.26.9 附录：工作负载 Pod 异常问题和解决方法

Pod 状态为 Pending

当Pod状态长时间为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。

图 4-474 pod 状态 pending

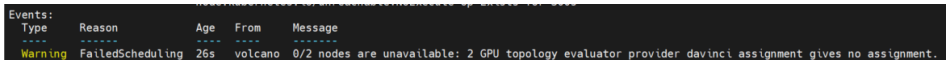
通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现以下图中的问题。

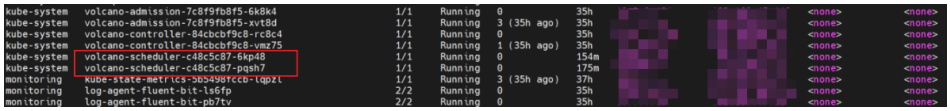
图 4-475 volcano 资源争抢



解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```

   图 4-476 scheduler

3. 若重启后，还是会Pending，建议多重复重启几次。

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

4.27 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.909）

4.27.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**表4-243**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。
- Lite Server驱动版本要求23.0.6
- PyTorch版本：2.1.0
- 确保容器可以访问公网。

文档更新内容

6.3.909版本相对于6.3.908版本新增如下内容：

- 文档中新增对Llama3.1的适配。
- ModelLink框架和MindSpeed已升级到最新版本。

训练支持的模型列表

本方案支持以下模型的训练，如**表4-241**所示。

表 4-241 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce

序号	支持模型	支持模型参数量	权重文件获取地址
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

操作流程

图 4-477 操作流程图

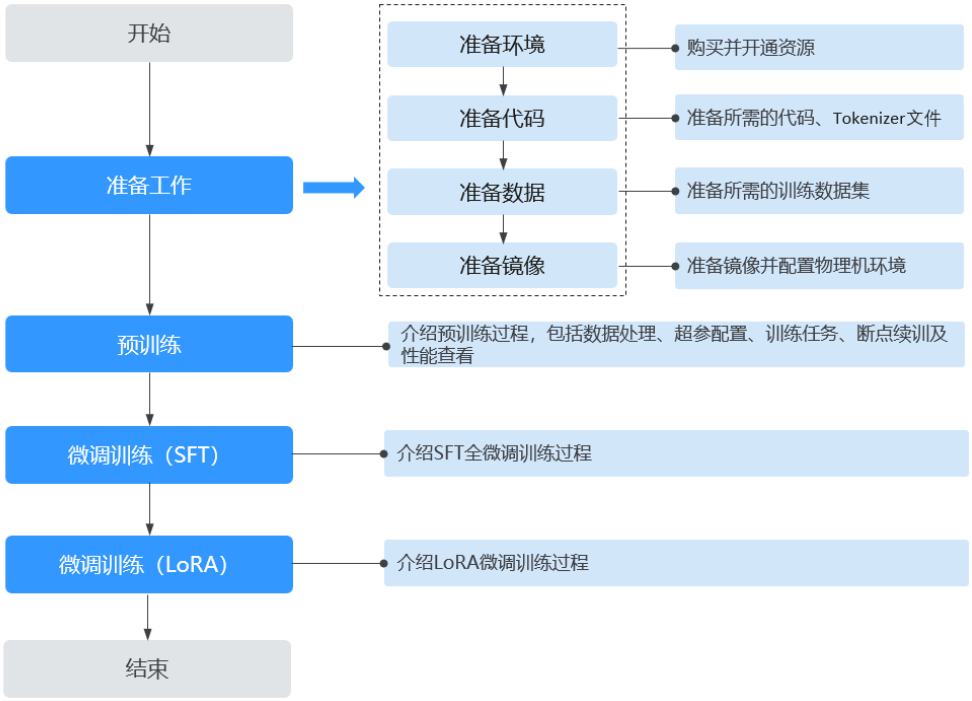


表 4-242 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。

阶段	任务	说明
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.27.2 准备工作

4.27.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-251**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.27.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如表4-243所示，模型列表、对应的开源权重获取地址如表4-244所示。

表 4-243 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

表 4-244 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了git clone repo_url 的方式下载，但是不支持断点续传，并且clone会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
  |——llm_train                # 模型训练代码包
     |——AscendSpeed           # 基于AscendSpeed的训练代码
        |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
        |——scripts/             # 训练需要的启动脚本
           |——llama2            # llama2系列模型执行脚本的文件夹
           |——llama3            # llama3系列模型执行脚本的文件夹
           |——qwen              # Qwen系列模型执行脚本的文件夹
           |——qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
           |—— ...
           |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
           |—— install.sh       # 环境部署脚本
        |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——llm_inference              # 推理代码包
  |——llm_tools                  # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws）
|——llm_train                #解压代码包后自动生成的代码目录，无需用户创建
   |—— AscendSpeed           # 代码目录
        |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |——scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
   # 自动生成数据目录结构
   |—— processed_for_input        #目录结构会自动生成，无需用户创建
        |—— ${model_name}           # 模型名称
           |—— data              # 预处理后数据
```

```
            |── pretrain   # 预训练加载的数据
            |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
   |── saved_dir_for_output          # 训练输出保存权重，目录结构会自动生成，无需用户创建
      |── ${model_name}              # 模型名称
         |── logs            # 训练过程中日志（loss、吞吐性能）
            |── saved_models
         |── lora            # lora微调输出权重
         |── sft             # 增量训练输出权重
         |── pretrain        # 预训练输出权重
|── tokenizers                 #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
   |── Llama2-70B
|── models                     #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
   |── Llama2-70B
|── training_data              #原始数据目录，需要用户手动创建，后续操作步骤中会提示
   |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
   |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.27.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下this格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

  如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：
  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
  - MOSS数据集的Excel中需要有三个列名称：conversation_id, Human, assistant

- conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
- Human：数据集中每条数据的输入。
- assistant：数据集中每条数据的输出。

– 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

- user_id：用户的唯一不重复的ID值，必选。
- excel_addr：待处理的excel文件的地址，必选。
- dataset_name：处理后的数据集名称，必选。
- proportion：测试集所占份数，范围[1,9]，可选。
- test_count：测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
- proportion和test_count二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.27.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-245 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3

表 4-246 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
   ```
   docker exec -it ${container_name} bash
   ```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
4. 使用ma-user用户安装依赖包。
   ```
   #进入scripts目录换
   cd /home/ma-user/ws/llm_train/AscendSpeed
   #执行安装命令
   sh scripts/install.sh
   ```

⚠ 注意

- 若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 install.sh 脚本中的 transformers 的版本。
  - 由默认 transformers==4.45.0 修改为：transformers==4.44.2
- 为了避免因使用不同版本的 transformers 库进行训练和推理而导致冲突的问题，建议用户分别为训练和推理过程创建独立的容器环境。

通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/            # 训练需要的启动脚本
  |——src/                # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/        # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/          # MindSpeed昇腾大模型加速库
  |——ModelLink/          # ModelLink端到端的大语言模型方案
      |——megatron/      # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

如果git下载代码时报错，请参见**Git下载代码时报错**解决。

4.27.3 执行预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-247**所示。其他超参均有默认值，可以参考**表4-250**按照实际需求修改。

表 4-247 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**步骤二 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
  <NNODES=4> <NODE_RANK=0>
  示例:
  # 第一台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
  <NODE_RANK=0>
  示例:
  sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
  ```

注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/
llama2/0_pl_pretrain_7b.sh
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-478 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

4.27.4 执行 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和**0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-247**所示。其他超参均有默认值，可以参考**表4-250**按照实际需求修改。

表 4-248 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:
  # 第一台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  示例:
  sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  示例:
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
  ```

  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_sft_7b.sh
  ```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.27.5 执行 LoRA 微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-247**所示。其他超参均有默认值，可以参考**表4-250**按照实际需求修改。

表 4-249 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-251**所示。

步骤三 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:
  #第一台节点

```
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
```

- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。

  示例:

  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。

  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
  ```

- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
  ```

  如果单机运行需要指定使用NPU卡的数量，可提前定义变量 NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令。

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_lora_7b.sh
  ```

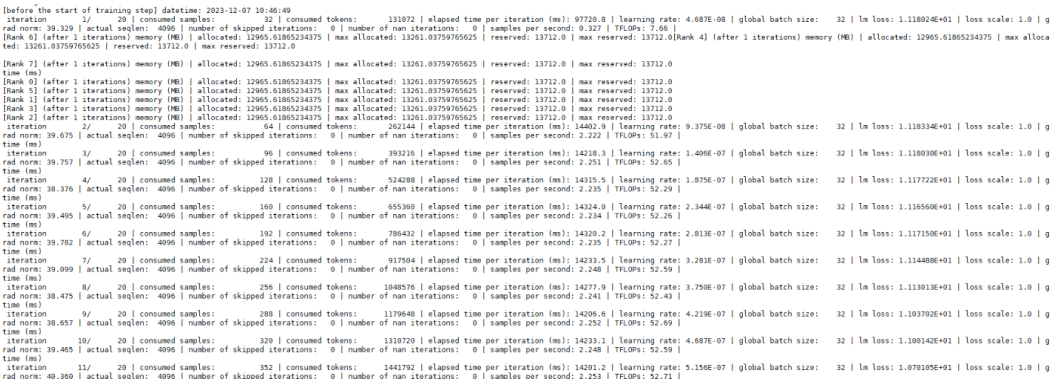
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

4.27.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-479 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

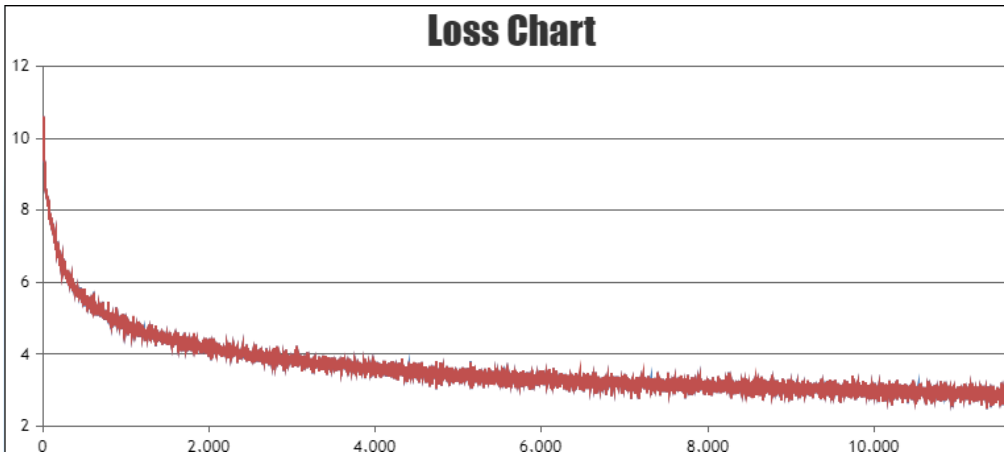
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-250**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-480**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-480 Loss 收敛情况（示意图）



4.27.7 训练脚本说明参考

4.27.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以下参数取值主要以**llama2-70b预训练**为例，请根据实际模型修改。

表 4-250 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。

参数	示例值	参数说明
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-251**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-251 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
24	llama3.1	llama3.1-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
25		llama3.1-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

4.27.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。

- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

  - 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}
```

```
"### Response:"
{output}
```

– 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

     iii. input_ids是user_ids和assistant_ids的拼接

     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

  d. attention_mask是和input_ids等长的全1序列

  e. 返回input_ids\attention_mask\labels的字典

  f. 处理完单一sample

  注：labels中用-100填充的地方，表示会被loss_mask给mask掉

  – 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。


```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-252 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.27.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

⚠ 注意

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-253 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.27.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-481所示。

图 4-481 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-482所示。

图 4-482 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-483 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-484所示。

图 4-484 修改 ChatGLMv4-9B tokenizer 文件

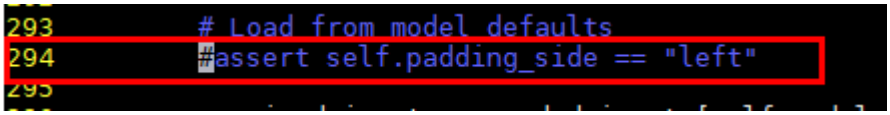
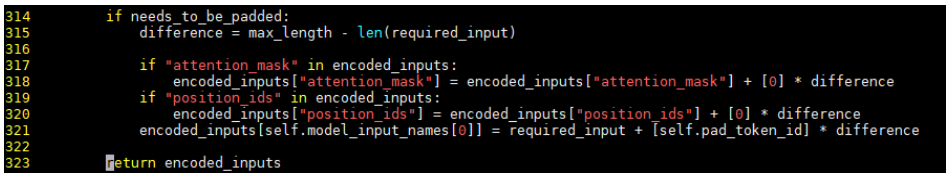


图 4-485 修改 ChatGLMv4-9B tokenizer 文件

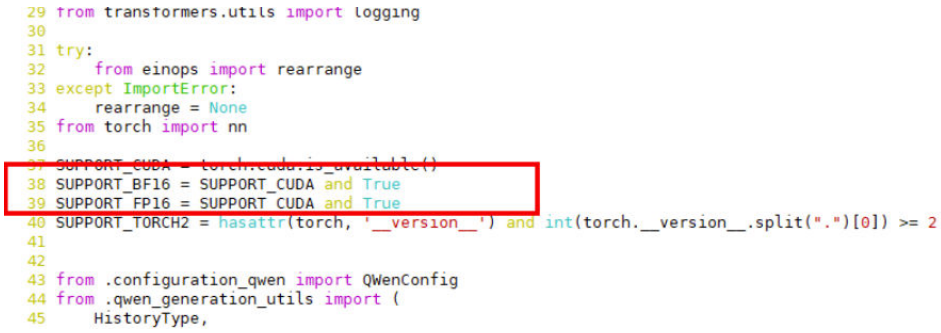


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-486所示。

图 4-486 修改 Qwen tokenizer 文件



4.27.8 常见错误原因和解决方法

4.27.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-251**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

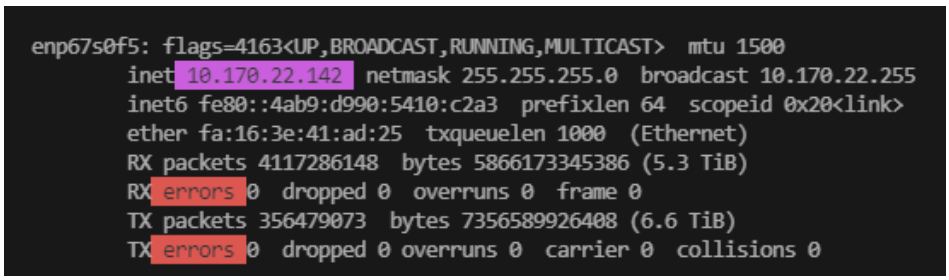
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.27.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-487 网卡名称错误



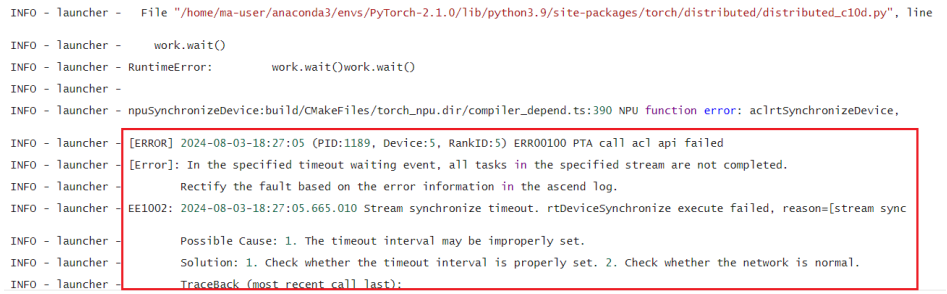

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.27.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-488 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.27.8.4 Git 下载代码时报错

在执行scripts/install.sh安装命令或使用Dockerfile构建镜像时，如遇到git下载代码出现以下类似的报错信息，关闭git验证即可。

报错信息：

```
fatal: unable to access 'https://gitee.com/ascend/ModelLink.git/': error setting certificate verify locations:
CAfile: /etc/pki/tls/certs/ca-bundle.crt CApath: none
```

关闭git验证命令如下：

```
git config --global http.sslverify false
```

4.28 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.909）

4.28.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的不同训练阶段方案，包括指令监督微调、DPO偏好训练、RM奖励模型训练、PPO强化训练方案。

- DPO(Direct Preference Optimization)：直接偏好优化方法，通过直接优化语言模型来实现对大模型输出的精确把控，不用进行强化学习，也可以准确判断和学习到使用者的偏好，最后，DPO算法还可以与其他优化算法相结合，进一步提高深度学习模型的性能。
- RM奖励模型(Reward Model)：是强化学习过程中一个关键的组成部分。它的主要任务是根据给定的输入和反馈来预测奖励值，从而指导学习算法的方向，帮助强化学习算法更有效地优化策略
- PPO强化学习(Proximal Policy Optimization)：是一种在强化学习中广泛使用的策略优化算法。它属于策略梯度方法的一种，旨在通过限制新策略和旧策略之间

的差异来稳定训练过程。PPO通过引入一个称为“近端策略优化”的技巧来避免过大的策略更新，从而减少了训练过程中的不稳定性和样本复杂性。

- 指令监督式微调(Self-training Fine-tuning)：是一种利用有标签数据进行模型训练的方法。 它基于一个预先训练好的模型，通过调整模型的参数，使其能够更好地拟合特定任务的数据分布。 与从头开始训练模型相比，监督式微调能够充分利用预训练模型的知识和特征表示，从而加速训练过程并提高模型的性能。

训练阶段下有不同的训练策略，分为全参数训练、部分参数训练、LoRA、QLoRA，本文档主要支持全参数（Full）和LoRA。

- LoRA(Low-Rank Adaptation)： 这种策略主要针对如何在保持模型大部分参数固定的同时，通过引入少量可训练参数来调整模型以适应特定任务。
- 全参训练（Full）：这种策略主要对整个模型进行微调。这意味着在任务过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**表4-256**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.RC3。
- Server驱动版本要求23.0.6
- PyTorch版本：2.2.0
- Python版本：3.10
- 确保容器可以访问公网。
- 仅支持313T、376T、400T

训练支持的模型列表

本方案支持以下模型的训练，如**表4-254**所示。

表 4-254 支持的模型列表及权重文件地址

支持模型	支持模型参数量	权重文件获取地址
Llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
	llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
	llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-chat-hf**
Llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**

支持模型	支持模型参数量	权重文件获取地址
	llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
Llama3.1	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct/tree/main**
	llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct/tree/main**
Qwen1.5	qwen1.5-0.5b	**https://huggingface.co/Qwen/Qwen1.5-0.5B**
	qwen1.5-1.8b	**https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat**
	qwen1.5-4b	**https://huggingface.co/Qwen/Qwen1.5-4B**
	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
	qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
	yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
	qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
	qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**
	qwen2-72b	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
Qwen2_VL（支持多模态数据集）	qwen2_vl-2b	**https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main**
	qwen2_vl-7b	**https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main**
Falcon2	falcon-11B	**https://huggingface.co/tiiuae/falcon-11B**
GLM-4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本 4b556ad4d70c38924cb8c120adbf21a0012de6ce

表 4-255 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
训练	启动训练	介绍各个训练阶段：指令微调、PPO强化训练、RM奖励模型、DPO偏好训练使用全参/lora训练策略进行训练任务、性能查看。

4.28.2 准备工作

4.28.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-261**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite

Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.28.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-256**所示，模型列表、对应的开源权重获取地址如**表4-254**所示。

表 4-256 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                    # 模型训练代码包
      |——LLaMAFactory              # 基于LLaMAFactory的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——demo.yaml            # 样例yaml配置文件
         |——demo.sh              # 指令微调启动shell脚本
         |——intall.sh            # 需要的依赖包
         |——LLaMA-Factory        # LLaMAFactory的代码目录
      |——AscendSpeed              # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
   |—— LLaMAFactory             # 代码目录
      |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
      |——demo.sh              # 指令微调启动shell脚本
      |——demo.yaml            # 样例yaml配置文件
      |——intall.sh            # 需要的依赖包
      |——LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考步骤三 启动容器镜像
         |——data              # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|—— tokenizers                  #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
   |—— Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表4-259表格中output_dir参数值
|—— saved_dir_for_output_lf      # 训练输出保存权重，目录结构会自动生成，无需用户创建
   |—— ${model_name}             # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。
   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```
3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。
   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen2-72B
   ```

4.28.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-257 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_2_ascend:pytorch_2.2.0-cann_8.0.rc3-py_3.10-hce_2.0.2406-aarch64-snt9b-20240910150953-6faa0ed

表 4-258 模型镜像版本

模型	版本
CANN	cann_8.0.RC3
驱动	23.0.6
PyTorch	2.2.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。

2. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤三 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```
参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

– ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

– --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如：  chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

4.28.2.4 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前支持alpaca格式和sharegpt格式的微调数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加数据集描述；具体示例如下。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。

```
cd  /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
```

2. 将自定义原始数据（指令监督微调样例数据集：alpaca_gpt4_data.json.json）按照下面的数据存放目录要求放置。

说明

指令微调样例数据集alpaca_gpt4_data.json.json的下载链接：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**

数据存放参考目录结构如下：

```
${workdir}（例如/home/ma-user/ws/llm_train ）
 |── LLaMAFactory/data
     |── alpaca_en_demo.json                # 代码原有数据集
     |── identity.json                      # 代码原有数据集
     ...
     |── alpaca_gpt4_data.json              # 自定义数据集
```

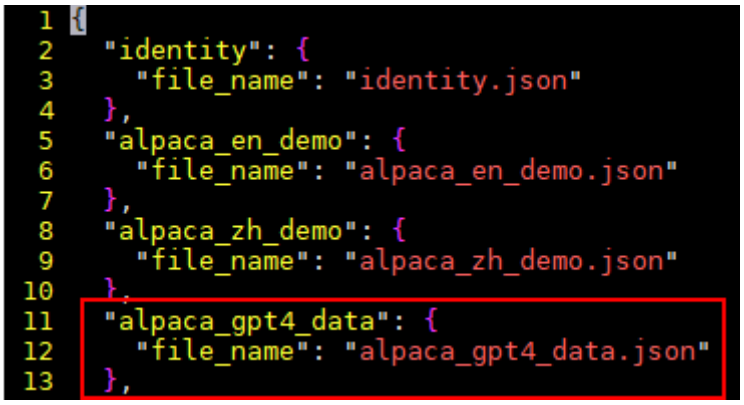
3. 更新代码目录下data/dataset_info.json文件。如使用以下示例数据集则命令如下。关于数据集文件格式及配置，更多样例格式信息请参考**data/README_zh.md** 的内容。

```
vim dataset_info.json
```

新加配置参数如下：

```
"alpaca_gpt4_data": {
  "file_name": "alpaca_gpt4_data.json"
 },
```

样例截图：



4.28.3 执行训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

步骤二 修改训练 yaml 文件配置

LlamaFactroy配置文件为Yaml文件，启动训练前需修改Yaml配置文件，Yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示。

1. 选择训练阶段类型。
   - 指令监督微调,复制**tune_yaml样例模板**内容覆盖demo.yaml文件内容。
   - DPO偏好训练，复制**dpo_yaml样例模板**内容覆盖demo.yaml文件内容。
   - PPO强化训练，先进行**RM奖励训练**任务后，复制**ppo_yaml样例模板**内容覆盖demo.yaml内容。
   - RM奖励训练，复制**rm_yaml样例模板**内容覆盖demo.yaml文件内容。

   说明

   1、DPO偏好训练、Reward奖励模型训练、PPO强化学习目前仅限制支持于llama3系列
   2、PPO训练暂不支持 ZeRO-3存在通信问题，如llama3-70B使用ZeRO-3暂不支持

2. 训练策略类型
   - 全参full，配置如下：

```
finetuning_type: full
```

- lora，如dpo仅支持此策略；配置如下：
  ```
  finetuning_type: lora
  lora_target: all
  ```

3. 修改**yaml文件**(demo.yaml)的参数如**表4-259**所示。

表 4-259 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-261**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	**可修改**。指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配。取值可参考**表4-261**中**梯度累积值**列。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

4. 是否选择加速深度学习训练框架Deepspeed，可参考**表4-261**选择不同的框架。

- 是，选用ZeRO (Zero Redundancy Optimizer)优化器。
  - ZeRO-0，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z0_config.json
    ```
  - ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
    ```
    deepspeed: examples/deepspeed/ds_z1_config.json
    ```
  - ZeRO-2，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z2_config.json
    ```
  - ZeRO-3，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z3_config.json
    ```
  - ZeRO-3-Offload，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z3_offload_config.json
    ```
- 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。

5. 是否开启NPU FlashAttention融合算子，具体约束详见**NPU_Flash_Attn融合算子约束**
   - 是，配置以下参数。
     ```
     flash_attn: sdpa
     ```
   - 否，配置以下参数关闭。
     ```
     flash_attn: disabled
     ```
6. 是否使用固定句长。
   - 是，配置以下参数
     ```
     packing: true
     ```
   - 否，默认使用动态句长，**注释**掉packing参数。
7. 选用数据精度格式bf16或fp16二者选一，两者区别可查看**BF16和FP16说明**。
   - bf16，配置以下参数。
     ```
     bf16: true
     ```
   - fp16，相比bf16还需配置loss scale参数，配置如下。
     - 设置fp16为True。
       ```
       fp16: true
       ```
     - 修改deepspeed的"loss_scale"参数，配置如下。
       - 修改**ZeRO优化器**配置文件，如ZeRO2命令如下。
         ```
         cd /home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
         vim ds_z2_config.json
         ```
       - 使用fp16容易出现数值溢出，因此配置loss scale建议配置4096或4096以上：
         ```
         "loss_scale": 4096,
         ```
8. 是否使用自定义数据集。
   - 是，参考**准备数据（可选）**后，以指令监督微调数据集为例，配置以下参数：参考**表4-259**dataset_dir和dataset参数说明；如alpaca_gpt4_data.json数据集前缀则为alpaca_gpt4_data。
     ```
     dataset: alpaca_gpt4_data
     dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
     ```
   - 否，使用代码包自带数据集，注释掉dataset_dir参数，配置参数如下。
     - 指令监督微调/PPO数据集
       ```
       dataset: identity,alpaca_en_demo
       ```

- 多模态数据集，如qwen2_vl系列模型
  dataset: mllm_demo,identity
- RM/DPO，目前仅支持llama3系列模型
  dataset: dpo_en_demo

9. 是否使用falcon-11b、qwen2_vl系列、glm4-9b模型。
   - 是，更新配置或命令。
     - falcon-11b，参考**falcon-11B模型**替换文件。
     - glm4-9b，参考**glm4-9b模型**修改文件内容。
     - qwen2_vl系列，数据集为**多模态数据集**，若前面步骤已配置请忽略。具体配置如下：
       数据集dataset配置：
       dataset: mllm_demo,identity
   - 否，忽略此步骤，执行下一步。
10. 如需其他配置参数，可参考**表4-260**按照实际需求修改。

步骤三 启动训练脚本

修改完yaml配置文件后，启动训练脚本。模型不同最少NPU卡数不同，NPU卡数建议值可参考**表4-261**。

1. **修改启动脚本demo.sh**

   进入代码目录{work_dir}/llm_train/LLaMAFactory下**修改启动脚本**，其中{work_dir}为容器挂载路径

   ①是否为PPO强化训练；

   - 是，demo.sh添加变量；
     export PYTORCH_NPU_ALLOC_CONF = expandable_segments:False
   - 否，忽略此步骤，执行下一步；

   ②修改yaml文件路径：修改demo.sh最后一行代码，将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，例如将以下命令

   - 修改前

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
   ```

   - 修改后：

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
   ```

2. **执行多机启动命令**（可选）

   多台机器执行训练启动命令如下。

   ```
   多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
   示例:
   #第一台节点
   sh demo.sh  xx.xx.xx.xx  4  0
   # 第二台节点
   sh demo.sh  xx.xx.xx.xx  4  1
   # 第三台节点
   sh demo.sh  xx.xx.xx.xx  4  2
   # 第四台节点
   sh demo.sh  xx.xx.xx.xx  4  3
   ```

   以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

3. **执行单机启动命令**（可选）

   一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

   进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

   ```
   # 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
   sh demo.sh localhost 1 0
   ```

   单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
   ```

   其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务。

训练成功标志

“***** train metrics *****”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：训练常见问题**解决。

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：训练常见问题**问题3小节。

3、大模型参数如（qwen2-72B、llama2-70B）等sft训练完成后多线程退出时报“torch.distributed.DistStoreError: Socket Timeout”时请参考**问题4：Error waiting on exit barrier错误**

4、需要开启profiling功能进行性能数据采集和解析请参考**录制Profiling**

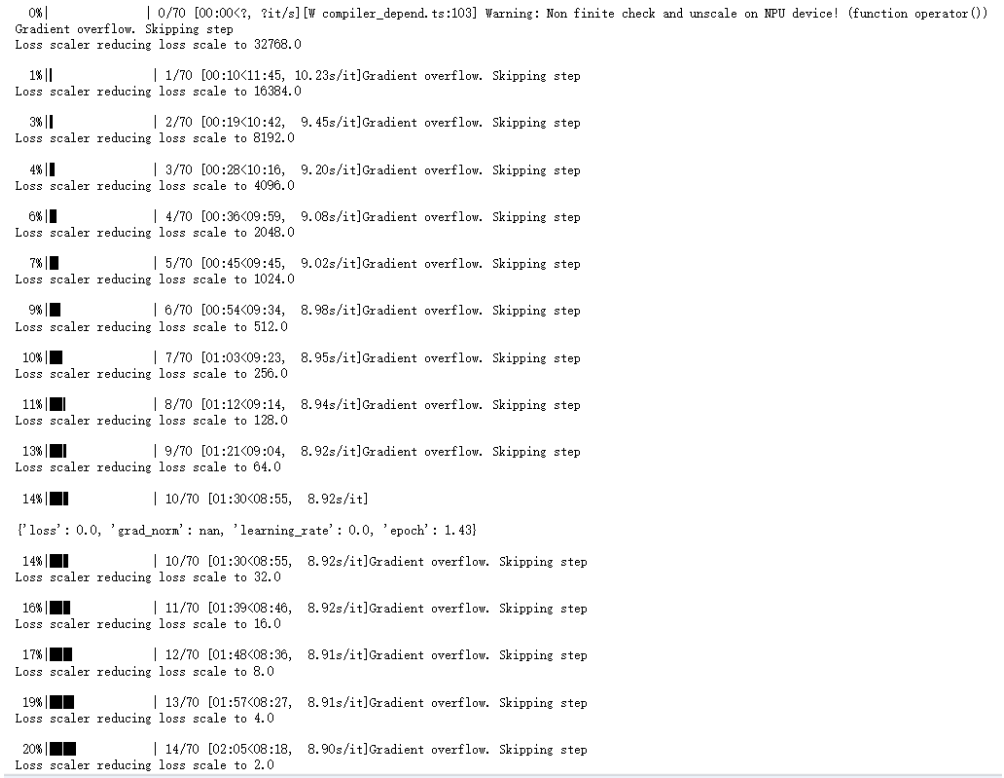
5、训练过程中报"ModuleNotFoundError: No module named 'multipart'"关键字异常，可更新python-multipart为0.0.12版本，具体请参考**6-问题6：No module named 'multipart'"报错：** 。

4.28.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-489 打印训练日志



训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表4-259**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：可通过**表4-259**表格中**output_dir**参数值路径下的**trainer_log.jsonl**计算性能。取中间过程多steps平均值吞吐计算公式为：

  delta_tokens = end_total_tokens-start_ total_tokens

  delta_time = end_elapsed_time - start_elapsed_time

  吞吐值(tps) = delta_tokens / delta_time / 训练卡数

  如图所示：

{"current_steps": 35, "total_steps": 54, "loss": 0.983,
6.785605346968387e-06, "epoch": 0.6466512702078522, "pe
"elapsed_time": "0:13:54", "remaining_time": "0:07:32",
2750.19, "total_tokens": 2293760}
{"current_steps": 36, "total_steps": 54, "loss": 0.988
6.173165676349103e-06, "epoch": 0.6651270207852193, "pe
"elapsed_time": "0:14:16", "remaining_time": "0:07:08",
2756.06, "total_tokens": 2359296}
{"current_steps": 37, "total_steps": 54, "loss": 0.974
5.5771130978099896e-06, "epoch": 0.6836027713625866, "
"elapsed_time": "0:14:38", "remaining_time": "0:06:43",
2761.64, "total_tokens": 2424832}
{"current_steps": 38, "total_steps": 54, "loss": 1.031
5.000000000000003e-06, "epoch": 0.7020785219399538, "pe
"elapsed_time": "0:15:00", "remaining_time": "0:06:18",
2766.96, "total_tokens": 2490368}

start

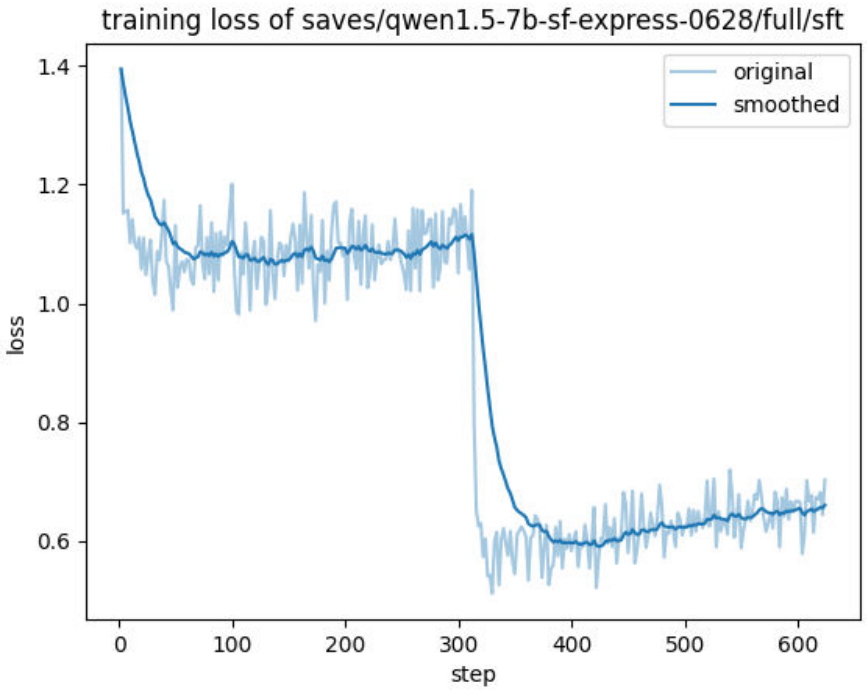
end

- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-259**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，将**trainer_log.jsonl**文件长传至可视化工具页面，如**图4-490**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

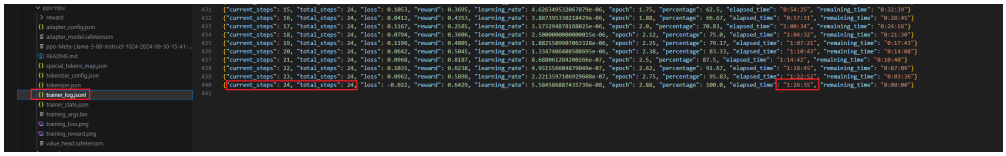
  多节点训练：训练过程中的loss打印在第一个节点上。

  图 4-490 Loss 收敛情况（示意图）



ppo训练结束不会打印性能。建议根据保存路径下的trainer_log.jsonl文件的最后一行总的训练steps和时间来判断性能。

图 4-491 trainer_log.jsonl 文件



4.28.5 训练脚本说明

4.28.5.1 Yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-260 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示当前的训练阶段。可选择值：sft、rm、ppo、dpo。 • sft代表指令监督微调； • rm代表奖励模型训练； • ppo代表PPO训练； • dpo代表DPO训练。

参数	示例值	参数说明
finetuning_type	full	用于指定微调策略类型，可选择值full、lora。 如果设置为full，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
lora_target	all	采取lora策略方法的目标模块，默认为all
dataset	• 指令监督微调/ppo：alpaca_en_demo • rm/dpo:dpo_en_demo • 多模态数据集(图像)：mllm_demo,identity	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表4-261**中的**template列**
max_samples	50000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。
per_device_train_batch_size	1	指定每个设备的训练批次大小。
gradient_accumulation_steps	8	**必须修改**，指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可参考**表4-261**

参数	示例值	参数说明
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
save_steps	5000	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	2.0e-5	指定学习率
disable_gradient_checkpointing	true	关闭重计算，用于禁用梯度检查点，**默认开启**梯度检查点;在深度学习模型训练中用于保存模型的状态，以便在需要时恢复。这种技术可以帮助减少内存使用，特别是在训练大型模型时，但同时影响性能。True表示关闭重计算功能。
include_tokens_per_second include_num_input_tokens_seen	true	用于在训练过程中包含每秒处理的tokens和已经看到的输入tokens，方便计算性能。
reward_mode	/home/ma-user/ws/saves/rm/llama3-8b/lora	**PPO强化必修改**；指定Reward奖励任务完成时output_dir目录，PPO强化训练前提为完成Reward奖励学习；请根据实际规划修改。

tune_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
```

```
# 全参
finetuning_type: full

# lora
# finetuning_type: lora
# lora_target: all

deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/tune/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

dpo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: dpo
do_train: true

# lora
finetuning_type: lora
lora_target: all

pref_beta: 0.1
pref_loss: sigmoid
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: dpo_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/saves/dpo/llama3-8b/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
```

```
gradient_accumulation_steps: 8
learning_rate: 5.0e-6
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ppo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
reward_model: /home/ma-user/ws/saves/rm/llama3-8b/lora
### method
stage: ppo
do_train: true

# 全参
# finetuning_type: full
# reward_model_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: identity,alpaca_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/ppo/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0
bf16: true
ddp_timeout: 180000000
flash_attn: sdpa
include_tokens_per_second: true
include_num_input_tokens_seen: true
### generate
max_new_tokens: 512
top_k: 0
top_p: 0.9
```

rm_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/llama3-8b
### method
stage: rm
do_train: true

# 全参
```

```
# finetuning_type: full

# lora
finetuning_type: lora
lora_target: all

deepspeed: examples/deepspeed/ds_z0_config.json
### dataset
dataset: dpo_en_demo
template: llama3
cutoff_len: 4096
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
packing: true
### output
output_dir: /home/ma-user/ws/saves/rm/llama3-8b/lora
logging_steps: 1
save_steps: 500
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-4
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0
bf16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
  }
}
```

4.28.5.2 模型 NPU 卡数、梯度累积值取值表

不同模型推荐的训练参数和计算规格要求如**表4-261**所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推。

表 4-261 NPU 卡数、加速框架、梯度配置取值表

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
llama2	llama2	7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		13B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
llama3	llama3	70B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full		gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
llama3.1	llama3	8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		70B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
Qwen2	qwen	72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		0.5/1.5B	lora/full	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
Qwen2_vl	qwen2_vl	2B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 2*Ascend
		7B	lora	4096	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
				8192	gradient_accumulation_steps: 8	ZeRO-2-Offload	1*节点 & 8*Ascend
Qwen1.5	qwen	0.5/1.8B	lora/full	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
		4B	lora	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
		7B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
		14B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
		32B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
			full	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			full	8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	2*节点 & 8*Ascend
		72B	lora	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			lora	8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
			full	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
falcon2	falcon	11B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
GLM4	glm4	9B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
Yi	yi	6B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
			full	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
		34B	full	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 2*Ascend

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
			full	8192	gradient_accumulation_steps: 8	ZeRO-3	4*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend

说明

以上参数为开启NPU FlashAttention融合算子，上述参数值仅供参考，请根据自己实际要求合理配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器、NPU节点数及其他配置。

具体优化工具使用说明可参考**如何选择最佳性能的zero-stage和-offloads**。

4.28.5.3 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

glm4-9b 模型

在训练开始前，需要修改glm4-9b模型中的tokenizer文件modeling_chatglm.py内容，具体步骤如下：

- 进入到tokenizer目录下{work_dir}/tokenizers/glm4-9B/，命令如下：

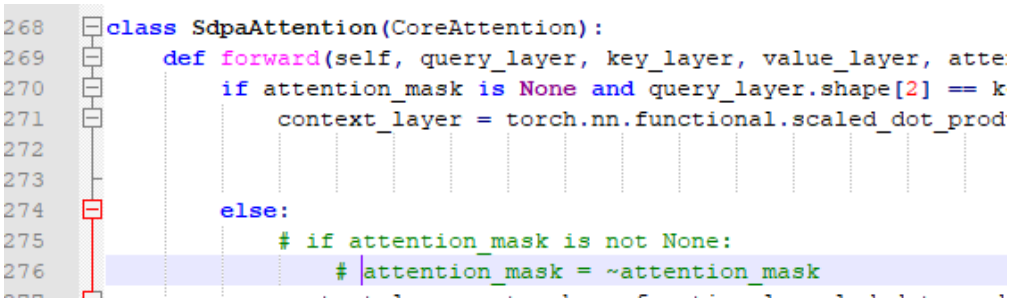
```
cd /home/ma-user/ws/tokenizers/glm4-9B
```

- 修改modeling_chatglm.py文件内容：

```
vim modeling_chatglm.py
# 注释掉以下两行内容
```

```
# if attention_mask is not None
    # attention_mask = ~attention_mask
```

样例图：



4.28.5.4 NPU_Flash_Attn 融合算子约束

1. query、key、value都需要梯度。默认开启重计算，则前向时qkv没有梯度，如果需要关闭重计算，可以在yaml配置 `disable_gradient_checkpointing: true` 关闭，但显存占用会直线上升。
2. attn_mask 只支持布尔（bool）数据类型，或者为None。
3. query的shape仅支持 [B, N1, S1, D]，其中N1≤ 2048，D≤ 512并且dim== 4。
4. 对于GQA，key的shape是 [B, N2, S2, D]，其中 N2 ≤ 2048，并且N1是N2的正整数倍。

不满足以上场景，则不能实现NPU_Flash_Attn功能。

4.28.5.5 BF16 和 FP16 说明

在大模型训练中，BF16（Brain Floating Point）和FP16（Float16）都是使用的半精度浮点数格式，但它们在结构和适用性上有一些重要的区别。

BF16：具有8个指数位和7个小数位。在处理大模型时有优势，能够避免在训练过程中数值的上溢或下溢，从而提供更好的稳定性和可靠性，在大模型训练和推理以及权重存储方面更受欢迎。

FP16：用于深度学习训练和推理过程中，可以加速计算并减少内存的占用，对模型准确性的影响在大多数情况下较小。与BF16相比在处理非常大或非常小的数值时遇到困难，导致数值的精度损失。

综上所述，BF16因其与FP32相似的数值范围和稳定性，在大模型训练中提供了优势。而FP16则在计算效率和内存使用方面有其独特的优点，但可能在数值范围和稳定性方面略逊一筹。因此，选择哪种格式往往取决于具体的应用场景和训练需求。

4.28.5.6 录制 Profiling

Ascend PyTorch Profiler是针对PyTorch框架开发的性能数据采集和解析工具，通过在PyTorch训练脚本中插入Ascend PyTorch Profiler接口，执行训练的同时采集性能数据，完成训练后直接输出可视化的性能数据文件，提升了性能分析效率。

Ascend PyTorch Profiler接口可全面采集PyTorch训练场景下的性能数据，主要包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等，可以全方位分析PyTorch训练时的性能状态。

录制命令如下：

在启动训练脚本基础：**步骤三 启动训练脚本** 新加DO_PROFILER=1和PROF_SAVE_PATH=/save_path参数，单机启动举例说明：

```
DO_PROFILER=1 PROF_SAVE_PATH=/save_path sh demo.sh localhost 1 0
```

- PROF_SAVE_PATH：Profiling录制结果存放路径
- DO_PROFILER：是否开启Profiling录制功能

4.28.6 附录：训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

1. 容器内执行以下命令，指定NPU内存分配策略的环境变量，开启动态内存分配，即在需要时动态分配内存，可以提高内存利用率，减少OOM错误的发生。
   ```
   export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True
   ```
2. 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。
3. 替换深度学习训练加速的工具或增加zero等级，可参考**模型NPU卡数、梯度累积值取值表**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。
   a. - ZeRO-0 数据分布到不同的NPU
   b. - ZeRO-1 Optimizer States分布到不同的NPU
   c. - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
   d. - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU
4. 增加卡数重新训练，未解决找相关人员定位。

问题 2：访问容器目录时提示 Permission denied

解决方法：

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: XXX not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

问题 4：Error waiting on exit barrier 错误

错误截图：


```
[ERROR] Error waiting on exit barrier. Elapsed: 300.1057639122009 seconds
[ERROR] Traceback (most recent call last):
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py

[ERROR]     store_util.barrier(
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     synchronize(store, data, rank, world_size, key_prefix, barrier_timeout)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li

[ERROR]     agent_data = get_all(store, rank, key_prefix, world_size)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     store_get(f"{prefix}{node_rank}_FIN")
[ERROR] torch.distributed.DistStoreError: Socket Timeout
```

报错原因：多线程退出各个节点间超时时间默认为300s，时间设置过短。

解决措施：

修改容器内torch/distributed/elastic/agent/server/api.py文件参数：

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py
```

修改def _exit_barrier(self)方法中的barrier_timeout参数，修改后如图4-492所示。

```
#修改前
barrier_timeout=self._exit_barrier_timeout
#修改后
barrier_timeout=3000
```

图 4-492 修改后的 barrier_timeout 参数

```
913     def _exit_barrier(self):
914         """
915         Define a barrier that keeps the agent process alive until all workers finish.
916
917         Wait for ``exit_barrier_timeout`` seconds for all agents to finish
918         executing their local workers (either successfully or not). This
919         acts as a safety guard against user scripts that terminate at different
920         times.
921         """
922         log.info(
923             "Local worker group finished (%s). "
924             "Waiting %s seconds for other agents to finish",
925             self._worker_group.state, self._exit_barrier_timeout
926         )
927         start = time.time()
928         try:
929             store_util.barrier(
930                 self._store,
931                 self._worker_group.group_rank,
932                 self._worker_group.group_world_size,
933                 key_prefix= TERMINAL_STATE_SYNC_ID,
934                 barrier_timeout=3000,
935             )
936             log.info(
937                 "Done waiting for other agents. Elapsed: %s seconds", time.time() - start
938             )
939         except SignalException as e:
940             log.warning("Got termination signal: %s", e.sigval)
941             raise
942         except Exception:
943             log.exception(
944                 "Error waiting on exit barrier. Elapsed: %s seconds",
945                 time.time() - start
946             )
```

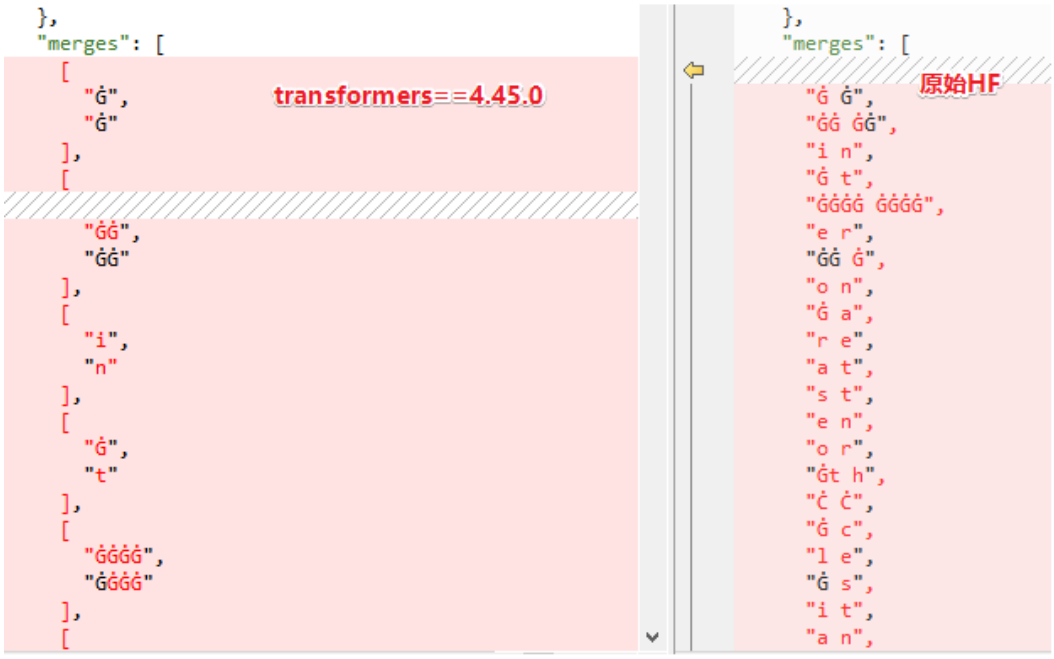
问题 5：训练完成使用 vllm0.6.0 框架推理失败：

错误截图：

```
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/tokenization_utils_fast.py", line 115, in __init__
    fast_tokenizer = TokenizerFast.from_file(fast_tokenizer_file)
Exception: data did not match any variant of untagged enum ModelWrapper at line 757272 column 1
```

报错原因：

训练时transformers版本要求为4.45.0，训练完成后保存的tokenizer.json文件中的“merges”时保存的是拆开的列表不是字符串，导致推理异常



解决措施，以下两种方法任选其一：

①更新transformes和tokenizers版本

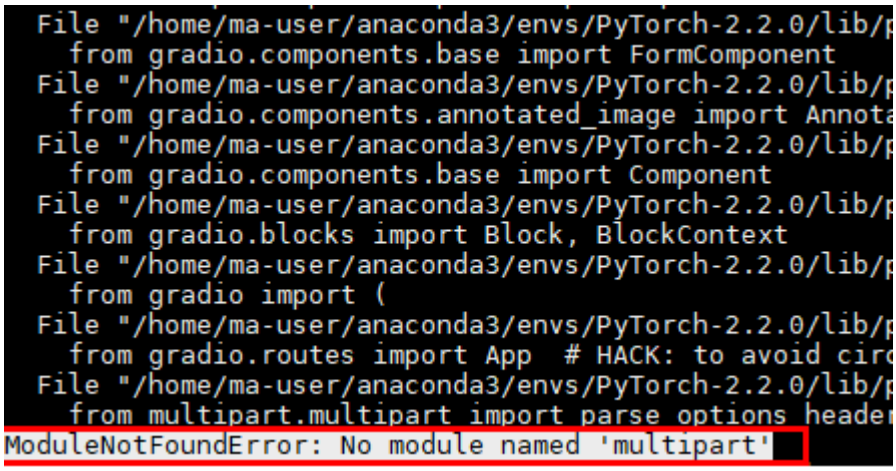
- GLM4-9B模型，容器内执行以下步骤：
  ```
  pip install transformers==4.43.2
  ```
- 其它模型，容器内执行以下步骤：
  ```
  pip install transformers==4.45.0
  pip install tokenizers==0.20.0
  ```

②使用原始hf权重的tokenizer.json覆盖保存的tokenizer.json即可，如llama3-8b_lora具体过程如下：

```
# 进入模型tokenizer目录
cd /home/ma-user/ws/tokenizers/llama3-8b/
# 替换tokenizer.json文件
cp -f tokenizer.json /home/ma-user/ws/saves/rm/llama3-8b/lora/tokenizer.json
```

问题 6：训练过程中报"ModuleNotFoundError: No module named 'multipart'"报错：

截图如下：



解决措施：可更新python-multipart为0.0.12版本，具体步骤如下：

- 启动训练任务前更新python-multipart版本：

```
pip install python-multipart==0.0.12
```

4.29 主流开源大模型基于 Standard+OBS 适配 ModelLink PyTorch NPU 训练指导（6.3.909）

4.29.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.909版本相对于6.3.908版本新增如下内容：

- 文档中新增对Llama3.1的适配。

- ModelLink框架和MindSpeed已升级到最新版本。

支持的模型列表

本方案支持以下模型的训练，如表4-262所示。

表 4-262 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b

序号	支持模型	支持模型参数量	权重文件获取地址
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
19		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**
20		qwen2-72b	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**
24	llama3.1	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct**
25		llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct**

操作流程

图 4-493 操作流程图

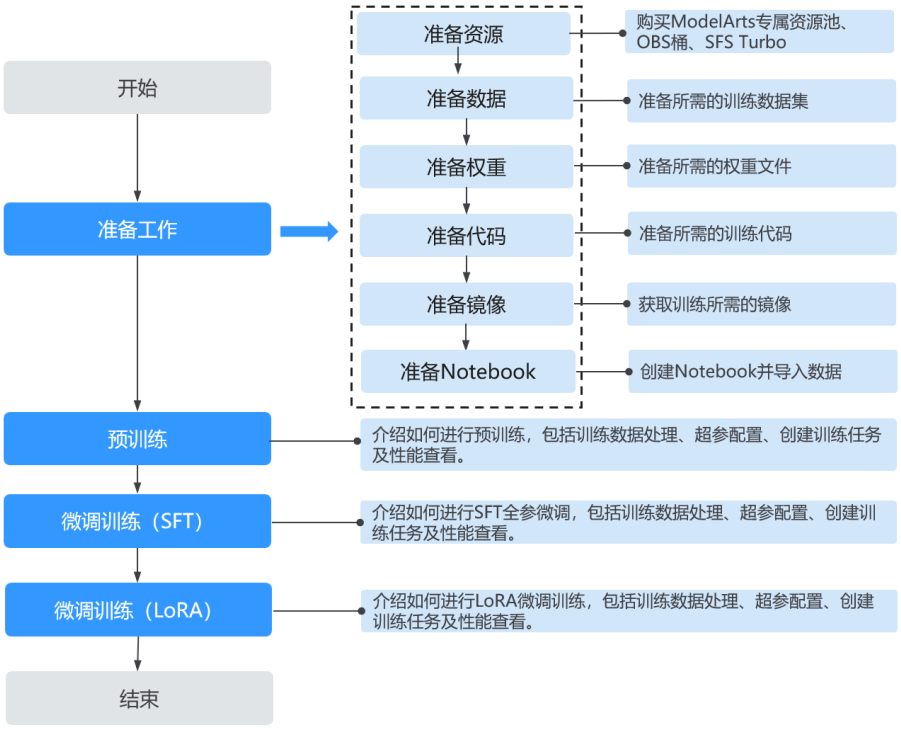


表 4-263 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.29.2 准备工作

4.29.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-269**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.29.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
     'id': '1',
     'url': 'https://simple.wikipedia.org/wiki/April',
     'title': 'April',
     'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

  如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：
  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst

- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id：指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human：数据集中每条数据的输入。
  - assistant：数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.29.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-262**。

   权重文件下载有如下几种方式，但不仅限于以下方式：
   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
```

如果要下载指定版本的模型文件，则命令如下：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.29.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-264**所示。

表 4-264 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.909代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
├──llm_train                # 模型训练代码包
      ├──AscendSpeed        # 基于AscendSpeed的训练代码
```

```
            |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |──scripts/             # 训练需要的启动脚本
                |──llama2           # llama2系列模型执行脚本的文件夹
                |──llama3           # llama3系列模型执行脚本的文件夹
                |──qwen             # Qwen系列模型执行脚本的文件夹
                |──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |── ...
                |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |── install.sh       # 环境部署脚本
            |──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|──llm_inference                  # 推理代码包
|──llm_tools                      # 推理工具
```

代码上传至 OBS

将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed             # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
|── models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.29.2.5 准备镜像

4.29.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-265 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行install.sh文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案**：在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

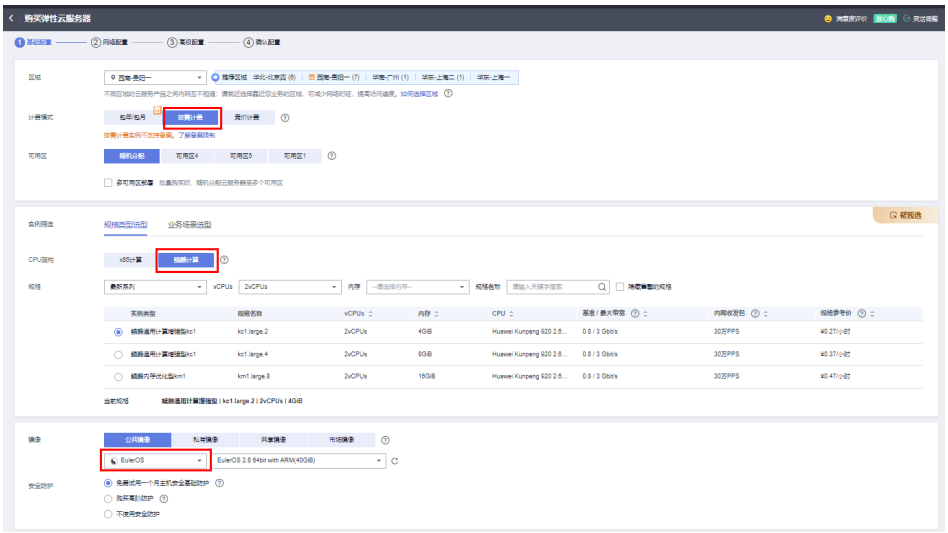
4.29.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

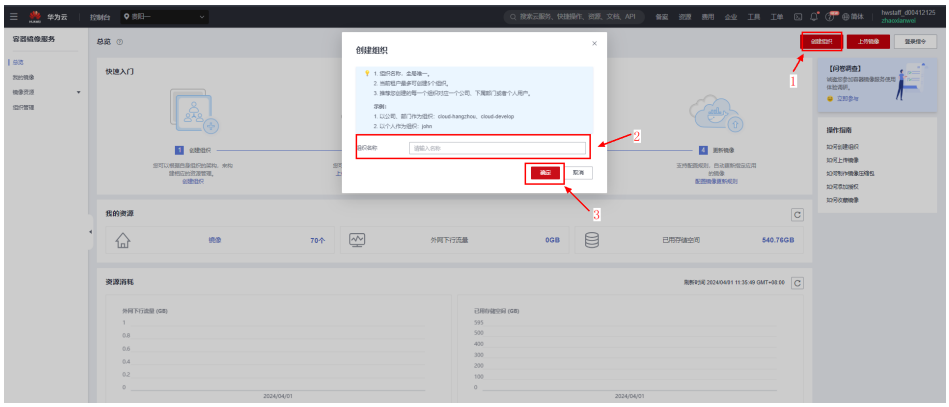
图 4-494 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-495 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

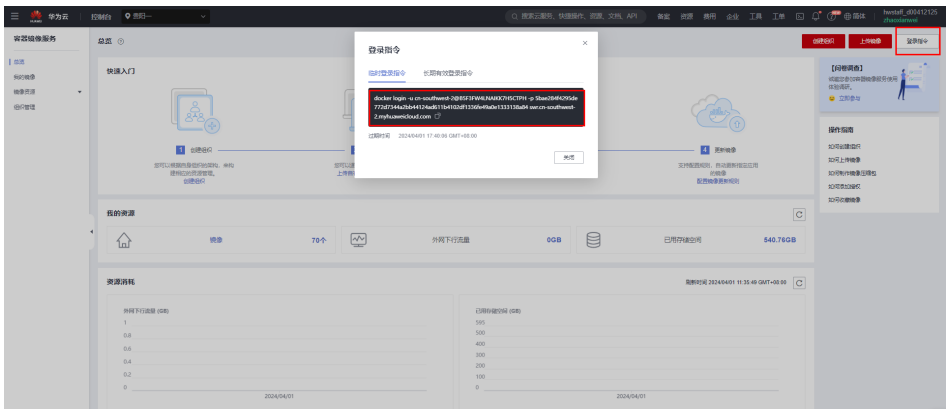
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-265**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-496 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.29.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-497**中都需要执行install.sh文件，来安装依赖以及下载完整代码。命令如下：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-497 训练作业启动命令

4.29.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-264**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.909-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。

   ```
   RUN git config --global http.sslVerify false && \
       git config --global user.email "you@example.com" &&  \
       git config --global user.name "Your Name" &&  \
       git clone https://gitee.com/ascend/ModelLink.git && cd ModelLink && git checkout 8f50777 && cd .. && \
       git clone https://gitee.com/lmzwhu/Megatron-LM.git  && \
       cd Megatron-LM  && \
       git checkout -f core_r0.6.0  && \
       cp -r megatron ../ModelLink/  && \
       cd ..  && \
       cd ModelLink && \
       pip install -e . && \
       cd .. && \
       git clone https://gitee.com/ascend/MindSpeed.git && \
       cd MindSpeed && \
       git checkout 4ea42a23 && \
       pip3 install -e . && cd .. && \
       pip install -e . && \
       pip install transformers==4.45.0
       chown -R ma-user:ma-group /home/ma-user/AscendSpeed && \
       chown -R ma-user:ma-group /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages && \
   ```

   注意

   若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

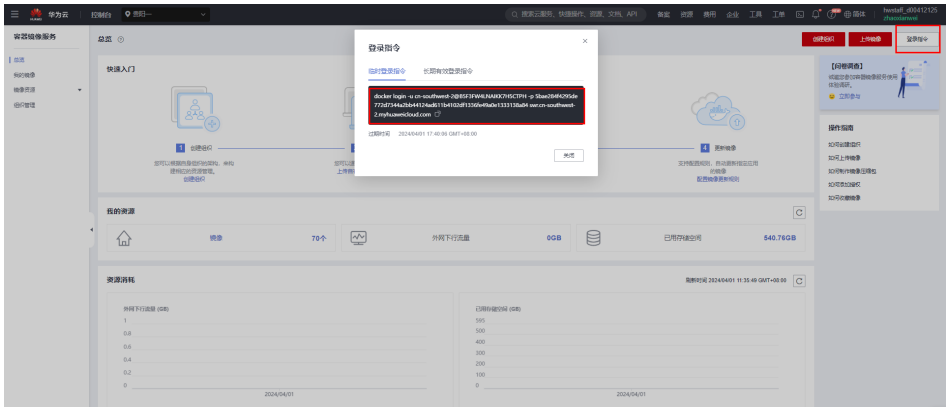
   由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-498 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.29.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中，如果用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-499 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，如果需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-500 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

4.29.3 预训练

前提条件

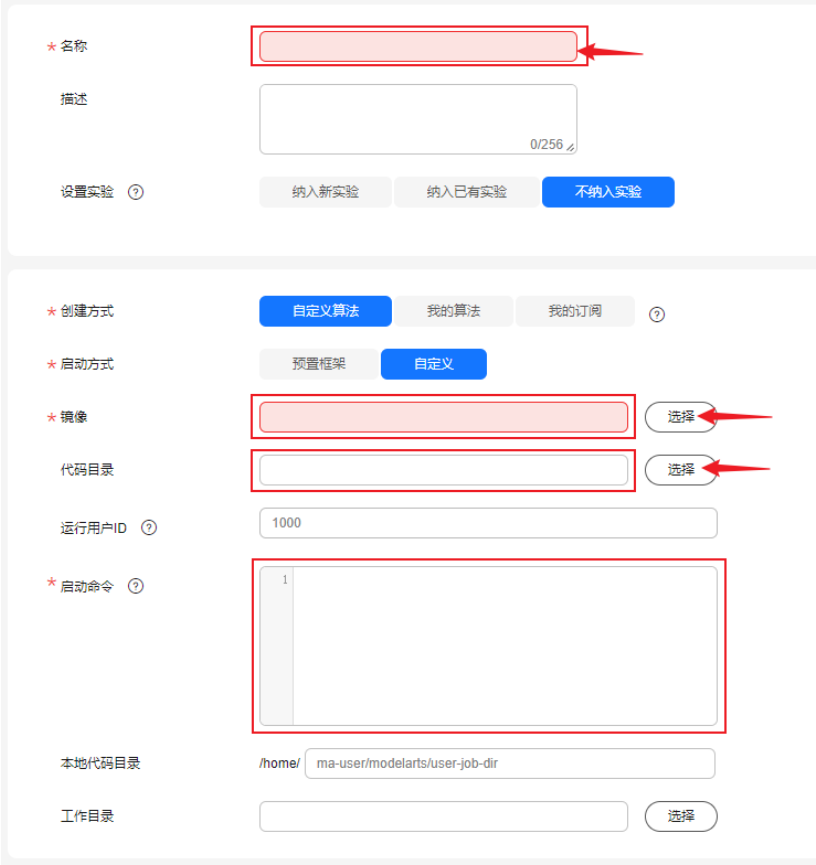
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-501 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-266表格中的配置进行填写。

表 4-266 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。

环境变量	示例值	参数说明
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

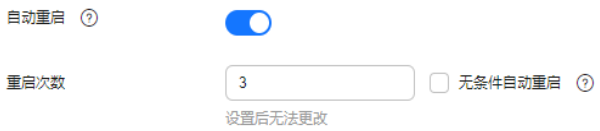
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-502 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-269**进行配置。

图 4-503 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.29.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-504 创建训练作业

如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-266表格中的配置进行填写。

图 4-505 环境变量

表 4-267 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。

环境变量	示例值	参数说明
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

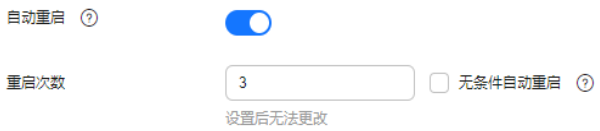
模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-506 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-269**进行配置。

图 4-507 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.29.5 LoRA 微调训练

前提条件

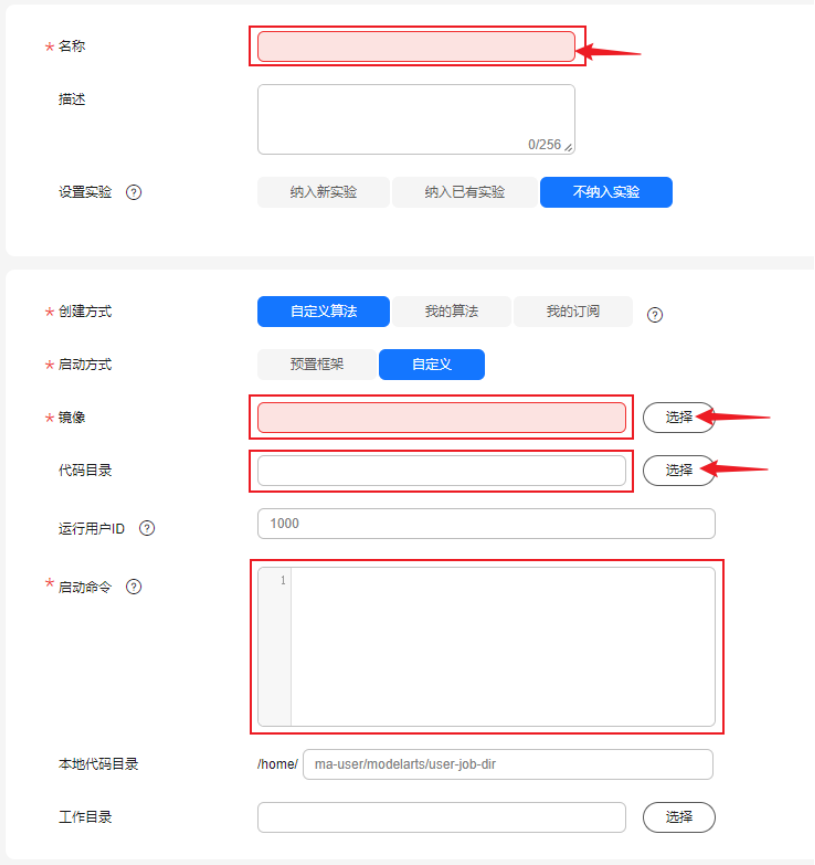
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-508 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-266**表格中的配置进行填写。

表 4-268 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

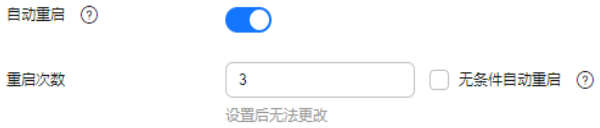
- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。

- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-509 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-269**进行配置。

图 4-510 选择资源池规格

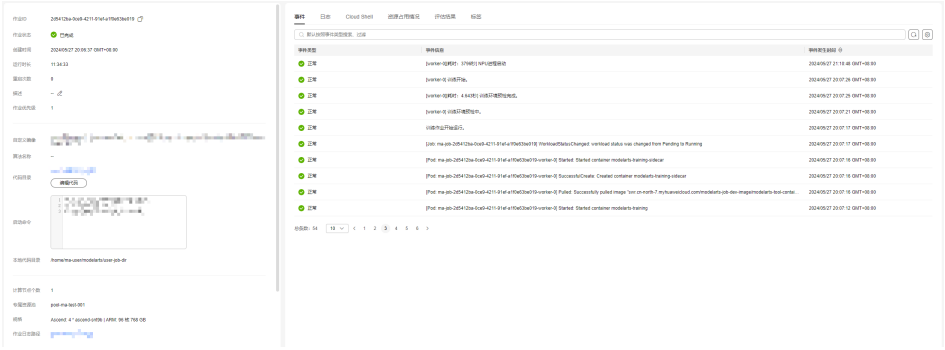
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.29.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

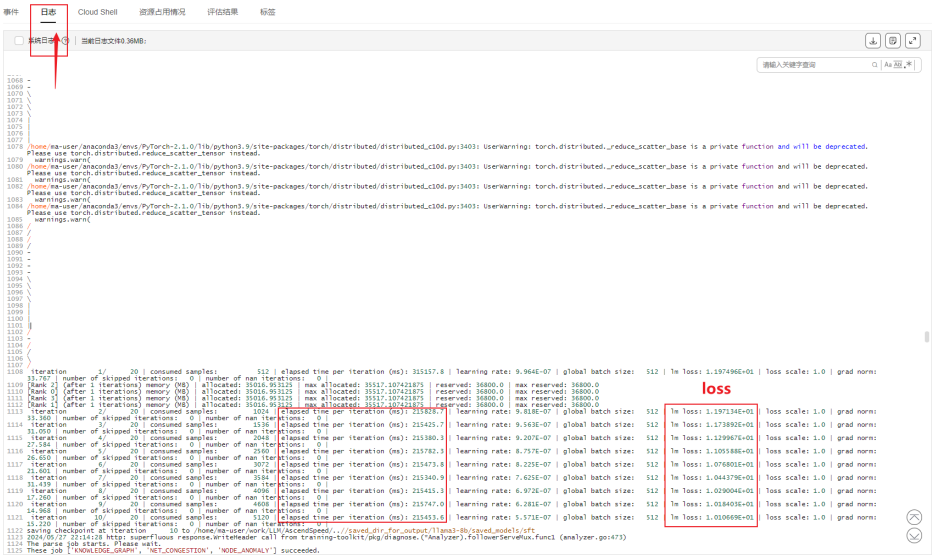
图 4-511 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-512 查看日志和性能



4.29.7 训练脚本说明

4.29.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 ……）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据

和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-269所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-269 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
24	llama3.1	llama3.1-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
25		llama3.1-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

4.29.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。如果未进行数据集预处理，则会自动执行**scripts/llama2/1_preprocess_data.sh**。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。

- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

  ```
  [
    {"text": "document"},
    {"other keys": "optional content"}
  ]
  ```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

  ```
  doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
  ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

  ```
  def _filter(self, sample):
      messages = self._format_msg(sample)
      full_prompt = self.prompter.generate_training_prompt(messages)
      tokenized_full_prompt = self._tokenize(full_prompt)
      if self.args.append_eod:
          tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
          tokenized_full_prompt["attention_mask"].append(1)
          tokenized_full_prompt["labels"].append(self.tokenizer.eod)
      if not self.train_on_inputs:
          user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
              self.prompter.template.assistant_token + "\n"
          tokenized_user_prompt = self._tokenize(user_prompt)
          user_prompt_len = len(tokenized_user_prompt["input_ids"])
          tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
      for key in self.args.json_keys:
          tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
      return tokenized_full_prompt
  ```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

iii. input_ids是user_ids和assistant_ids的拼接

iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-270 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。

环境变量	示例	参数说明
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.29.7.3 训练的权重转换说明

以**llama2-13b**举例，使用训练作业运行**obs_pipeline.sh**脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

如果用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。如果用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-271 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.29.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-513所示。

图 4-513 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-514所示。

图 4-514 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297             """
298             # Load from model defaults
299             # assert self.padding_side == "left"
300
301             required_input = encoded_inputs[self.model_input_names[0]]
302             seq_length = len(required_input)
303
```

图 4-515 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-516所示。

图 4-516 修改 ChatGLMv4-9B tokenizer 文件

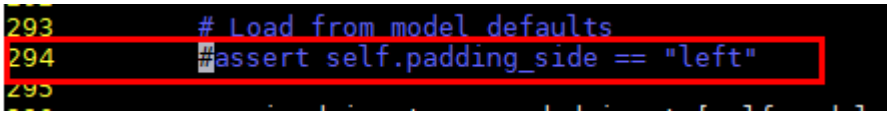
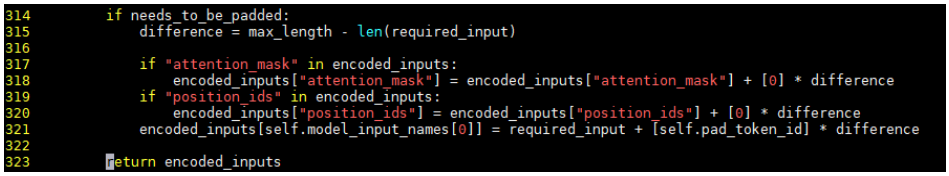


图 4-517 修改 ChatGLMv4-9B tokenizer 文件

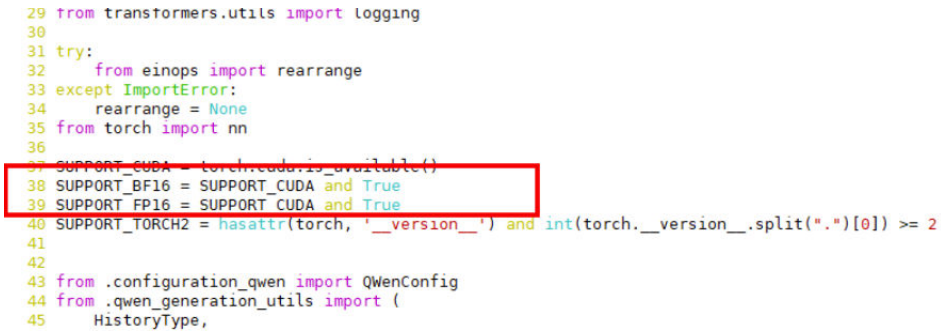


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-518所示。

图 4-518 修改 Qwen tokenizer 文件



4.29.8 常见错误原因和解决方法

4.29.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-269**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.29.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.30 主流开源大模型基于 Standard+OBS+SFS 适配 ModelLink PyTorch NPU 训练指导（6.3.909）

4.30.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.909版本相对于6.3.908版本新增如下内容：

- 文档中新增对Llama3.1的适配。
- ModelLink框架和MindSpeed已升级到最新版本。

支持的模型列表

本方案支持以下模型的训练，如表4-272所示。

表 4-272 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

序号	支持模型	支持模型参数量	权重文件获取地址
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

操作流程

图 4-519 操作流程图

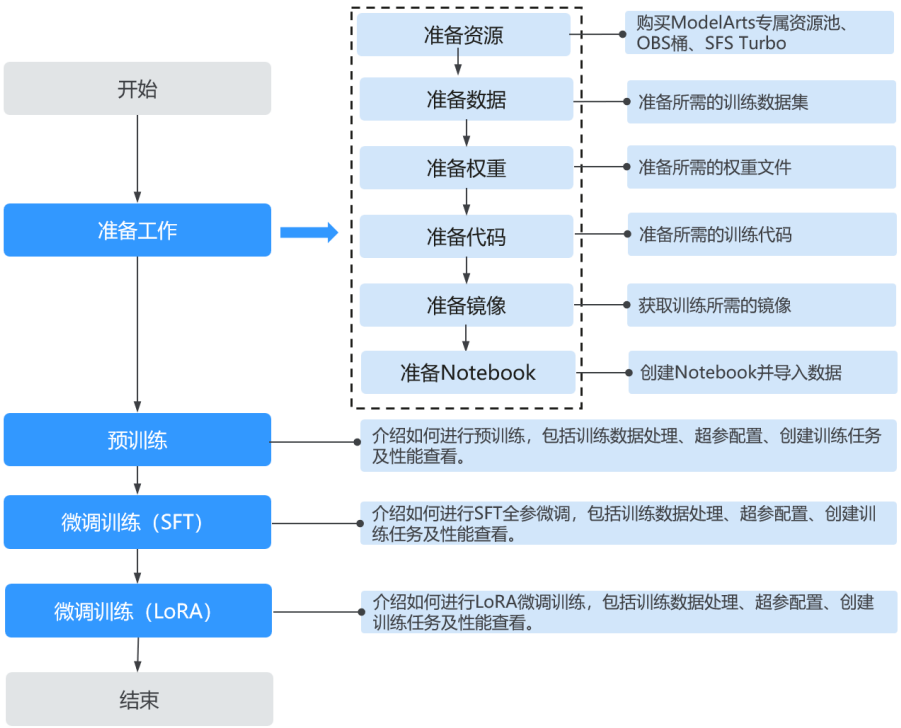


表 4-273 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。

阶段	任务	说明
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.30.2 准备工作

4.30.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-280**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。具体过程请参考**创建OBS桶**，例如桶名：standard-llama2-13b。

由于ModelArts创建训练作业时，需要将作业日志输出至OBS桶中，因此创建OBS桶为必选项。用户可通过**OBS Browser+**、**obsutil**等工具访问和管理OBS桶，将代码、模型文件、数据集等数据上传或下载进行备份。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

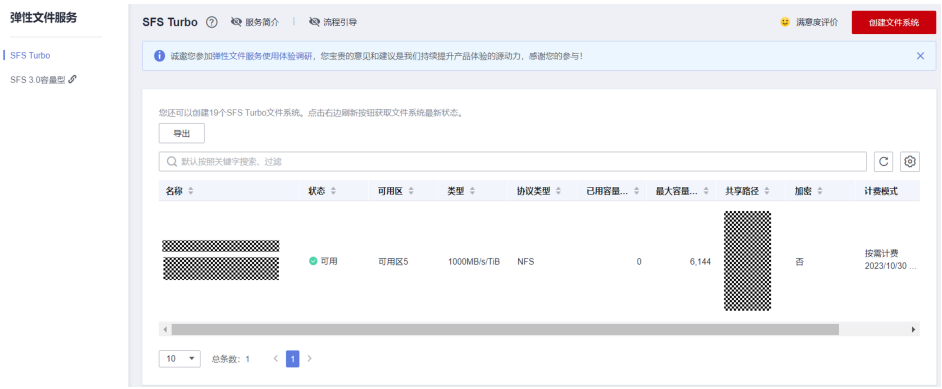
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-520 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

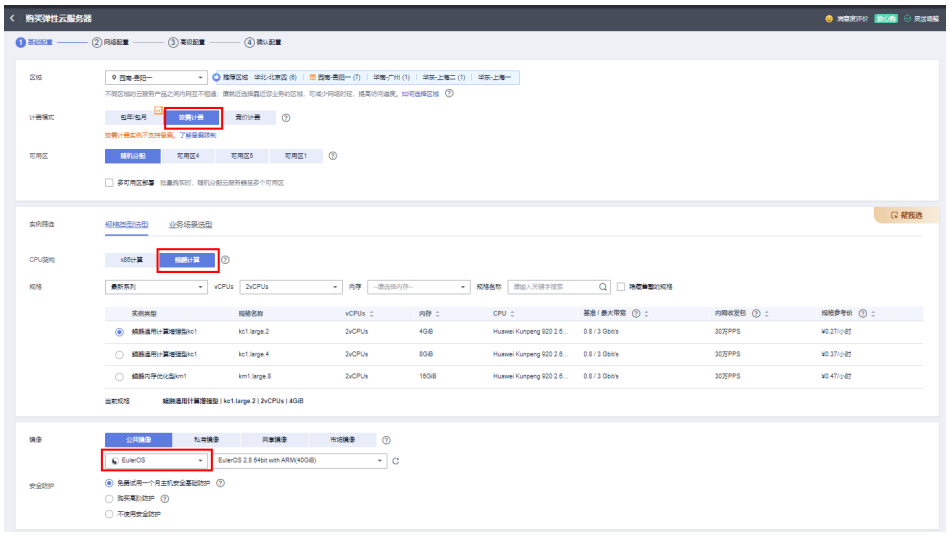
图 4-521 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 4-522 购买 ECS

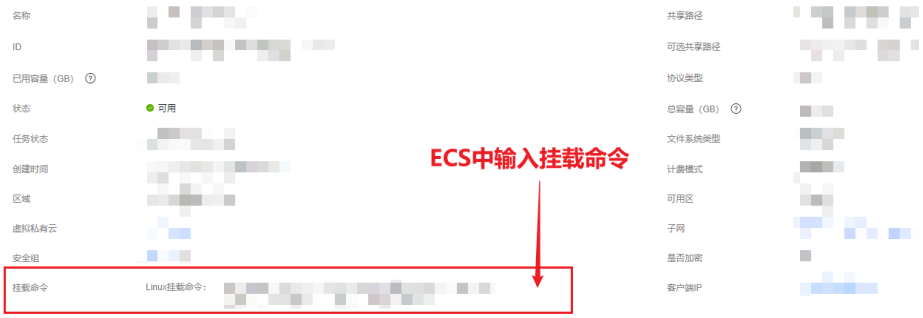


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看基本信息图4-523，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 4-523 SFS Turbo 基本信息



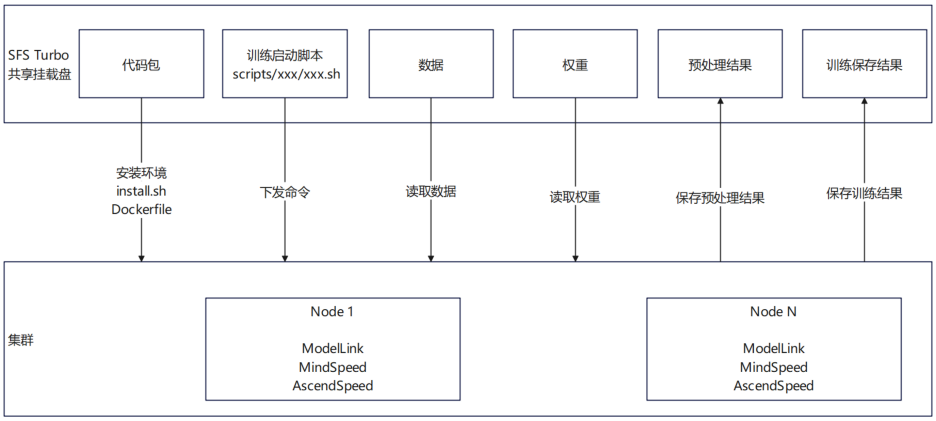
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-524 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表4-275**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.30.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据集至 SFS Turbo

准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。可通过两种方式，将数据集上传至SFS Turbo中。

方式一：将下载的原始数据通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/training_data目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具。数据存放参考目录：

```
/mnt/sfs_turbo/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
2. 利用**OBS Browser+工具**将下载的数据集上传至创建的文件夹目录下。得到OBS下数据集结构：
```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```
3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.30.2.3 准备权重

获取对应模型的权重文件，获取链接参考**表4-272**。权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。文件会直接下载用户本地，需要再上传至SFS Turbo中。
- **方法二：huggingface-cli**：**huggingface-cli**是Hugging Face官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
```
  如果要下载指定版本的模型文件，则命令如下：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```
- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

随后可通过以下两种方式，将下载到本地的模型文件上传至SFS Turbo中。

本地上传权重文件至 SFS Turbo

通过以下两种方式将下载到本地的模型文件上传至SFS Turbo中。方式一操作简单，但是数据传输速度比较慢，费时间。方式二操作相对方式一复杂一些，但是数据传输速度较快。

方式一：将已下载的模型文件通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/model目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放模型，例如在桶standard-llama2-13b中创建文件夹model/llama-2-13b-hf。
2. 利用**OBS Browser+工具**将下载的模型文件上传至创建的文件夹目录下。

3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.30.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-274**所示。

表 4-274 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.909代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
                |——llama2           # llama2系列模型执行脚本的文件夹
                |——llama3           # llama3系列模型执行脚本的文件夹
                |——qwen             # Qwen系列模型执行脚本的文件夹
                |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |—— ...
                |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |—— install.sh       # 环境部署脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

代码上传至 SFS Turbo

将AscendSpeed代码包AscendCloud-LLM-xxx.zip直接上传至ECS服务器中的SFS Turbo中，例如存放在/mnt/sfs_turbo/AscendCloud-LLM-xxx.zip目录下并解压缩。

```
unzip AscendCloud-*.zip
```

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至SFS Turbo后，目录结构如下。

```
/mnt/sfs_turbo/
|——llm_train                  # 解压代码包后自动生成的代码目录，无需用户创建
```

```
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          # 目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain    # 预训练加载的数据
          |── finetune    # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |─── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                          #微调数据文件
|── tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
|── models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.30.2.5 准备镜像

4.30.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-275 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**可二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。

- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

使用以上方案时，都会下载Megatron-LM、MindSpeed、ModelLink源码至AscendSpeed文件夹中。下载后的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/            # 训练需要的启动脚本
  |——src/                # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/        # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/          # MindSpeed昇腾大模型加速库
  |——ModelLink/          # ModelLink端到端的大语言模型方案
      |——megatron/      # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

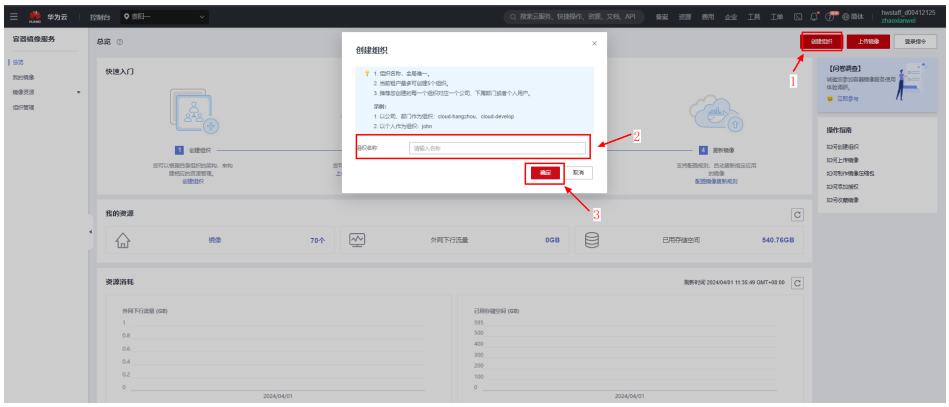
训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

4.30.2.5.2 ECS 获取和上传基础镜像

Step1 创建镜像组织

在SWR服务页面创建镜像组织。

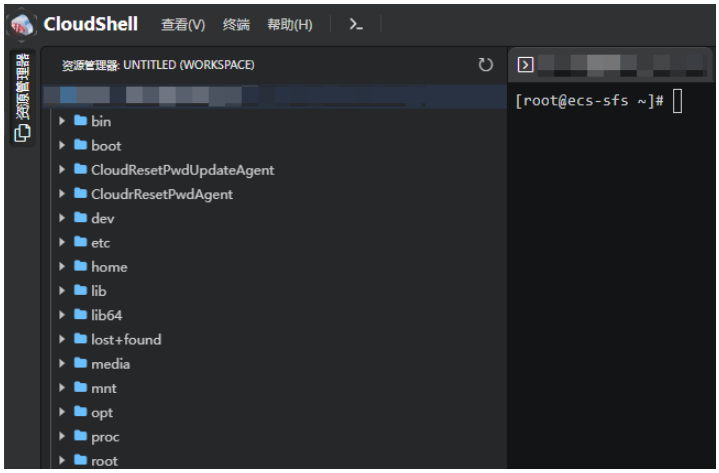
图 4-525 创建镜像组织



Step2 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录如图所示。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

图 4-526 CloudShell 远程登录界面



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

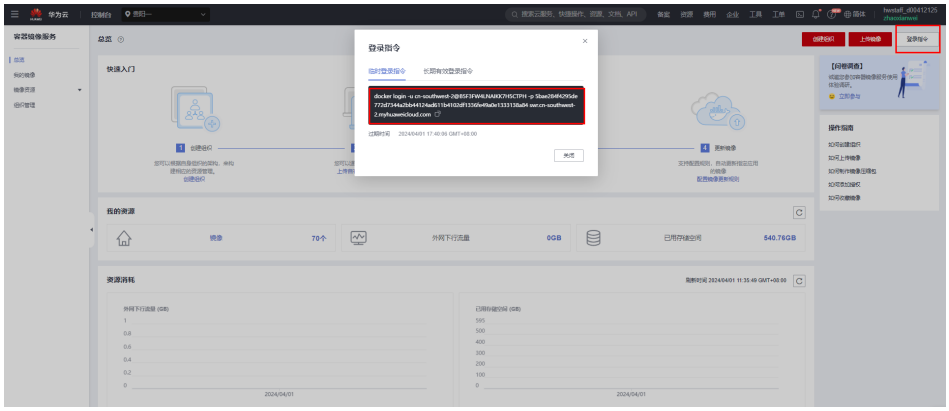
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-275**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-527 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.30.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

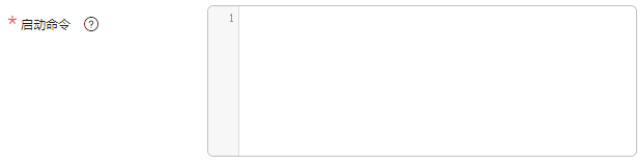
由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-528**中都需要执行 install.sh文件，来安装依赖以及下载完整代码。

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-528 训练作业启动命令



4.30.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-274**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   cd ./llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。
   ```
   FROM {image_url}
   ```
3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。

   ```
   RUN git config --global http.sslVerify false && \
       git config --global user.email "you@example.com" &&  \
       git config --global user.name "Your Name" &&  \
       git clone https://gitee.com/ascend/ModelLink.git && cd ModelLink && git checkout 8f50777 && cd .. && \
       git clone https://gitee.com/lmzwhu/Megatron-LM.git  && \
       cd Megatron-LM  && \
       git checkout -f core_r0.6.0  && \
       cp -r megatron ../ModelLink/  && \
       cd ..  && \
       cd ModelLink && \
       pip install -e . && \
       cd .. && \
       git clone https://gitee.com/ascend/MindSpeed.git && \
       cd MindSpeed && \
       git checkout 4ea42a23 && \
       pip3 install -e . && cd .. && \
       pip install -e . && \
       pip install transformers==4.45.0
       chown -R ma-user:ma-group /home/ma-user/AscendSpeed && \
       chown -R ma-user:ma-group /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages && \
   ```

注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

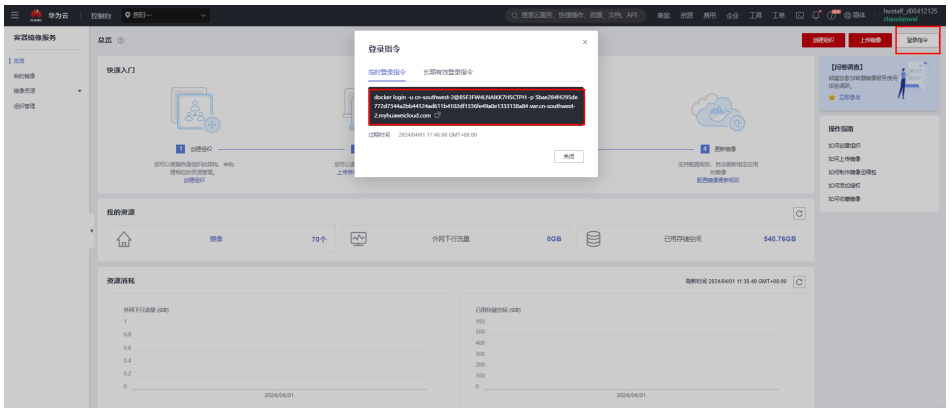
由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-529 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.30.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-276**所示。其他超参均有默认值，可以参考**表4-279**按照实际需求修改。

表 4-276 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	**可添加**。该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

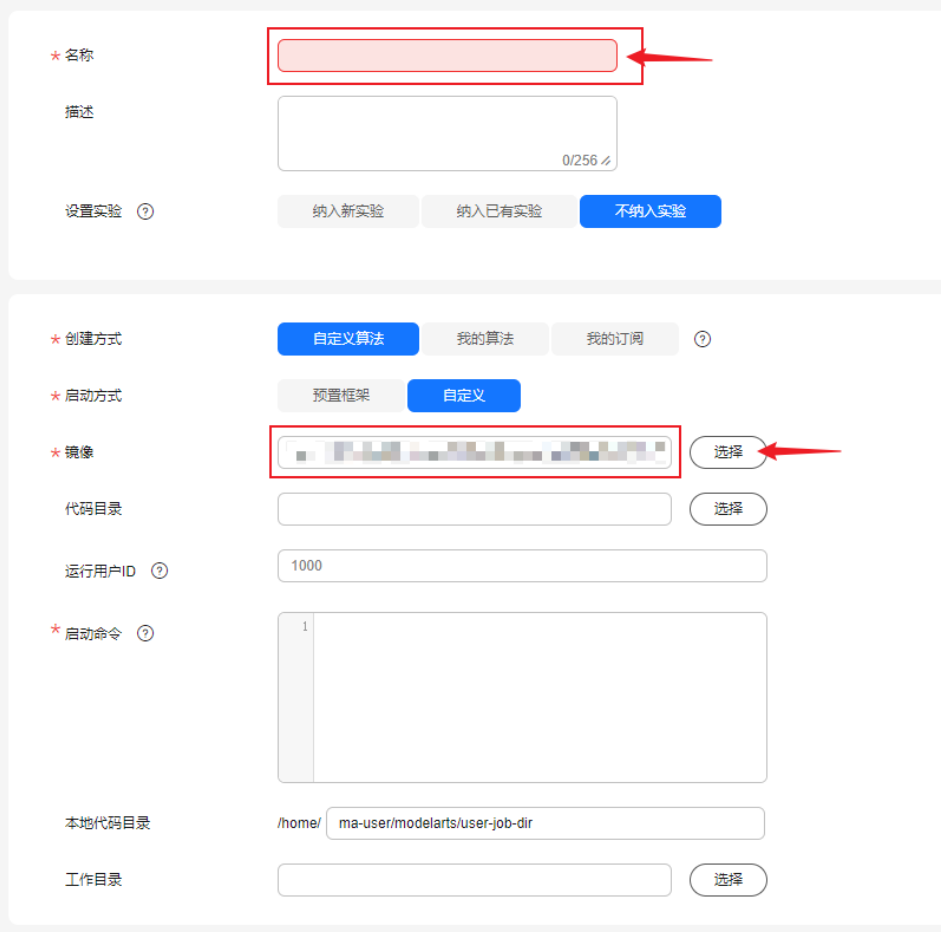
参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-530 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

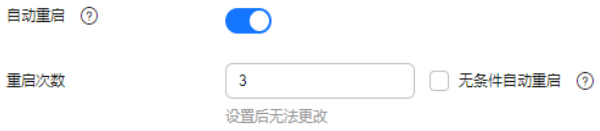
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-531 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

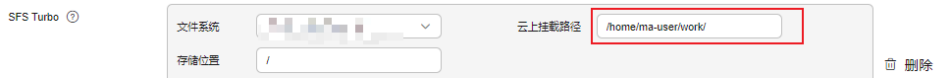
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-280**进行配置。

图 4-532 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-533 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.30.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-277**所示。其他超参均有默认值，可以参考**表4-279**按照实际需求修改。

表 4-277 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

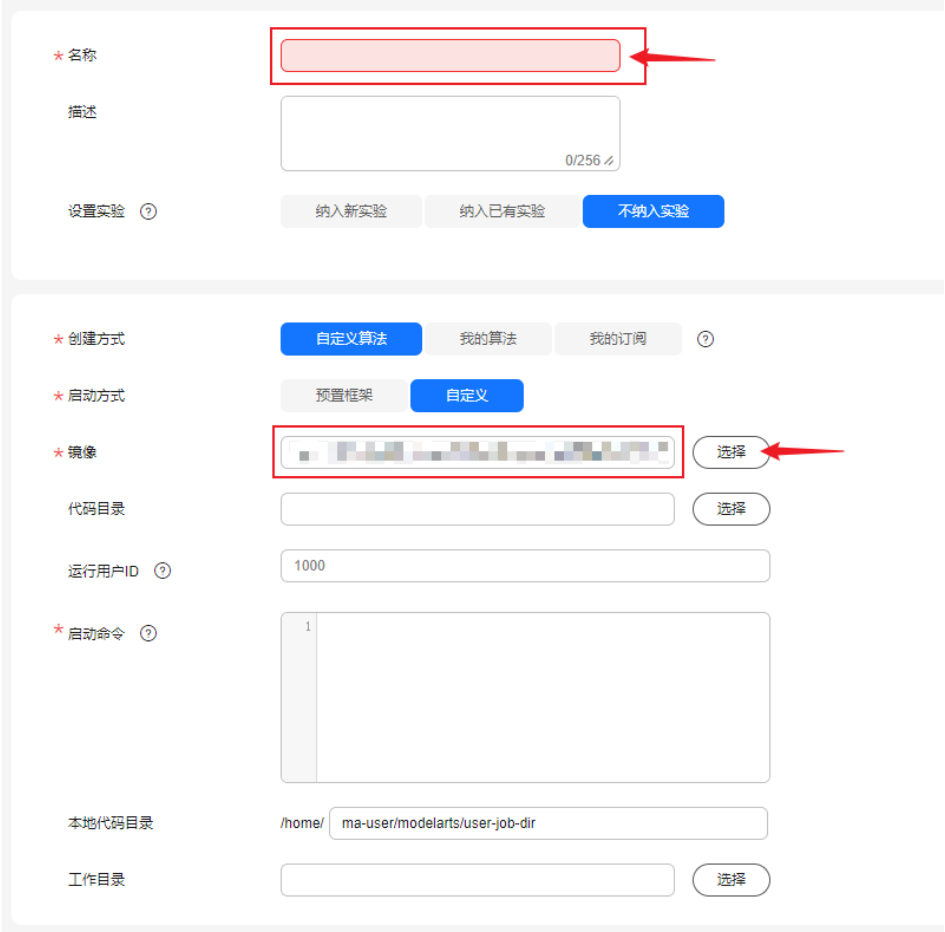
参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-534 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

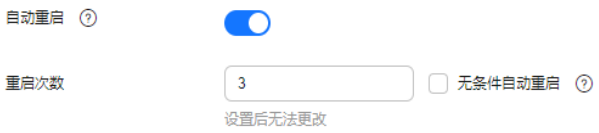
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-535 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-280**进行配置。

图 4-536 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-537 选择 SFS Turbo

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.30.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-278**所示。其他超参均有默认值，可以参考**表4-279**按照实际需求修改。

表 4-278 训练超参配置说明

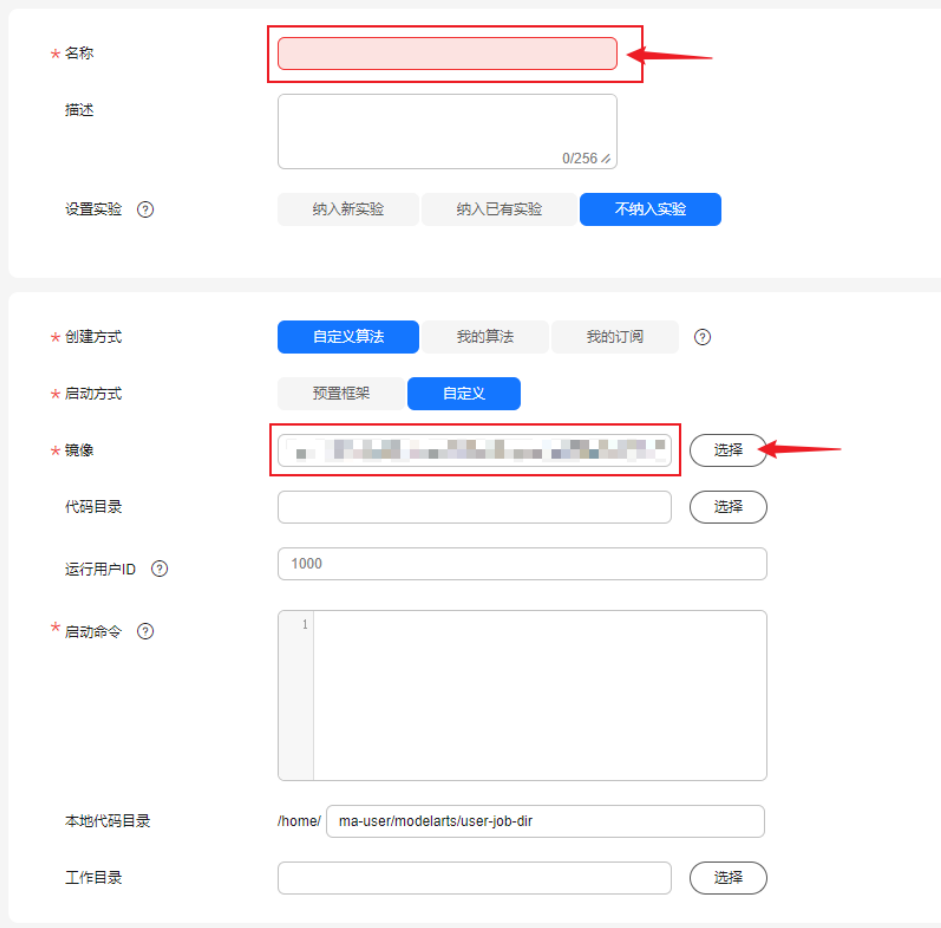
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-538 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

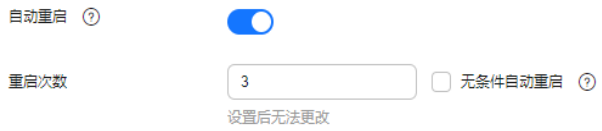
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-539 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。可以通过训练脚本中的SAVE_INTERVAL参数来指定间隔多少step保存checkpoint。

说明

如果要使用自动重启功能，资源规格必须选择八卡规格。

训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

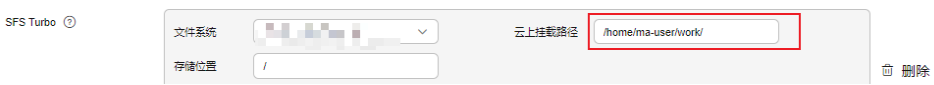
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-280**进行配置。

图 4-540 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-541 选择 SFS Turbo



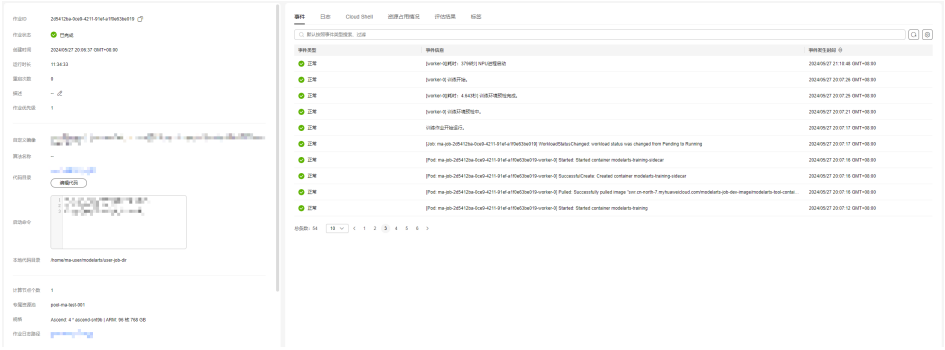
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.30.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

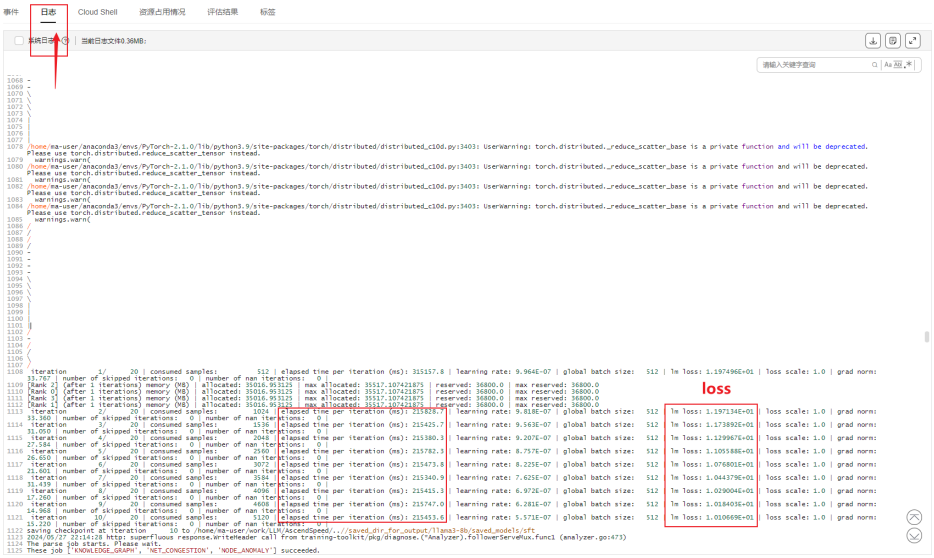
图 4-542 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-543 查看日志和性能



4.30.7 训练脚本说明

4.30.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后

的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-279 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。

参数	示例值	参数说明
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-280**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-280 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
24	llama3.1	llama3.1-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
25		llama3.1-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

4.30.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**0_pl_pretrain_13b.sh**训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。支持 .parquet \ .csv \ .json \ .jsonl \ .txt \ .arrow 格式。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。

  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
```

```
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

```
[
  {"text": "document"},
  {"other keys": "optional content"}
]
```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

```
doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

  - 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
```

```
{instruction} + "\n" + {input}

"### Response:"
{output}
```

– 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行
  b. 循环处理其中的单轮对话
  c. 在单轮对话中
     i. 对user和assistant的文本进行清洗
     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
     iii. input_ids是user_ids和assistant_ids的拼接
     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
  d. attention_mask是和input_ids等长的全1序列
  e. 返回input_ids\attention_mask\labels的字典
  f. 处理完单一sample

  注：labels中用-100填充的地方，表示会被loss_mask给mask掉

  – 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以llama2为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-281 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.30.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-282 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.30.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-544所示。

图 4-544 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-545 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-546 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-547 修改 ChatGLMv4-9B tokenizer 文件

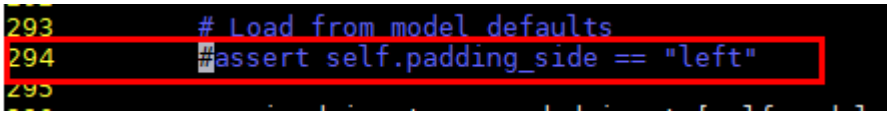
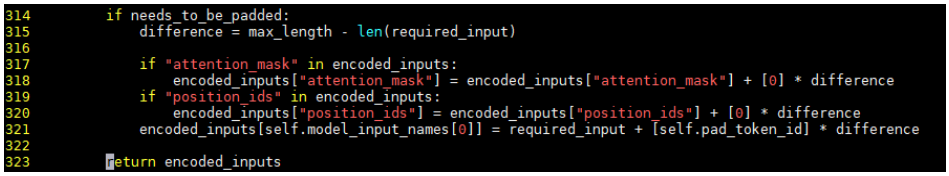


图 4-548 修改 ChatGLMv4-9B tokenizer 文件



Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-549 修改 Qwen tokenizer 文件

4.30.8 常见错误原因和解决方法

4.30.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-280**进行设置。

- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.30.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.31 主流开源大模型基于 Lite Cluster 适配 ModelLink PyTorch NPU 训练指导（6.3.909）

4.31.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Cluster上的训练方案。训练框架使用的是ModelLink。

本方案目前仅适用于企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.909版本，请参考表4-285获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Cluster。
- 本文档中的**CCE集群版本选择v1.27~1.28**。版本使用的容器引擎为**Containerd**。
- 镜像适配的Cann版本是cann_8.0.rc3，驱动版本是23.0.6。
- 确保集群可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如表4-283所示。

表 4-283 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
24	llama3.1	llama3.1-8b	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
25		llama3.1-70b	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

操作流程

图 4-550 操作流程图

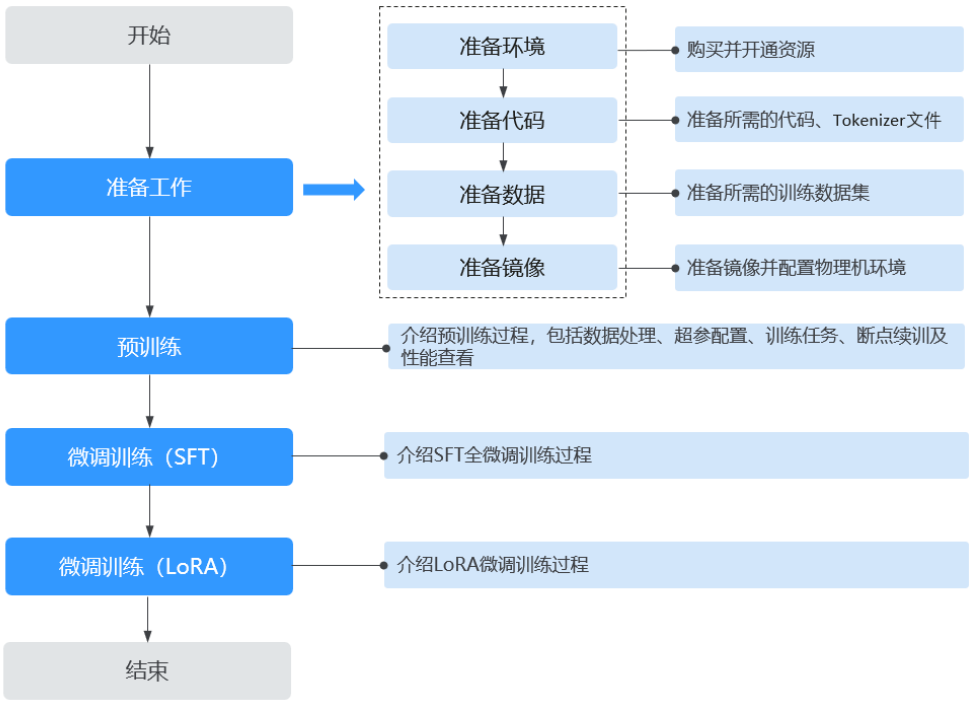


表 4-284 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite k8s Cluster运行的，需要购买并开通k8s Cluster资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.31.2 准备工作

4.31.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Cluster。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-292**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Cluster资源，请先阅读**k8s Cluster资源购买**，熟悉集群资源开通流程，再开始操作购买Cluster资源。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买专属资源池注意事项

- 使用场景需要选择ModelArts Lite。
- CCE集群已完成创建。
- 节点数量可自定义选择使用多少节点。
- 开启高级选项：输入容器引擎空间大小（推荐输入最大空间），容器引擎选择**Containerd**。

k8s Cluster 资源配置

若已完成集群资源购买和开通，则需要对网络、存储、容器镜像等内容进行配置。请参考**k8s Cluster环境配置详细流程**。

其中k8s Cluster的容器中挂载存储支持OBS、SFS Turbo等方案进行挂载。例如OBS支持静态挂载和动态挂载，而SFS Turbo仅支持静态挂载，详细的挂载操作流程可阅读**通过静态存储卷使用已有极速文件存储**和**通过动态存储卷使用对象存储**。

kubectl 访问集群配置

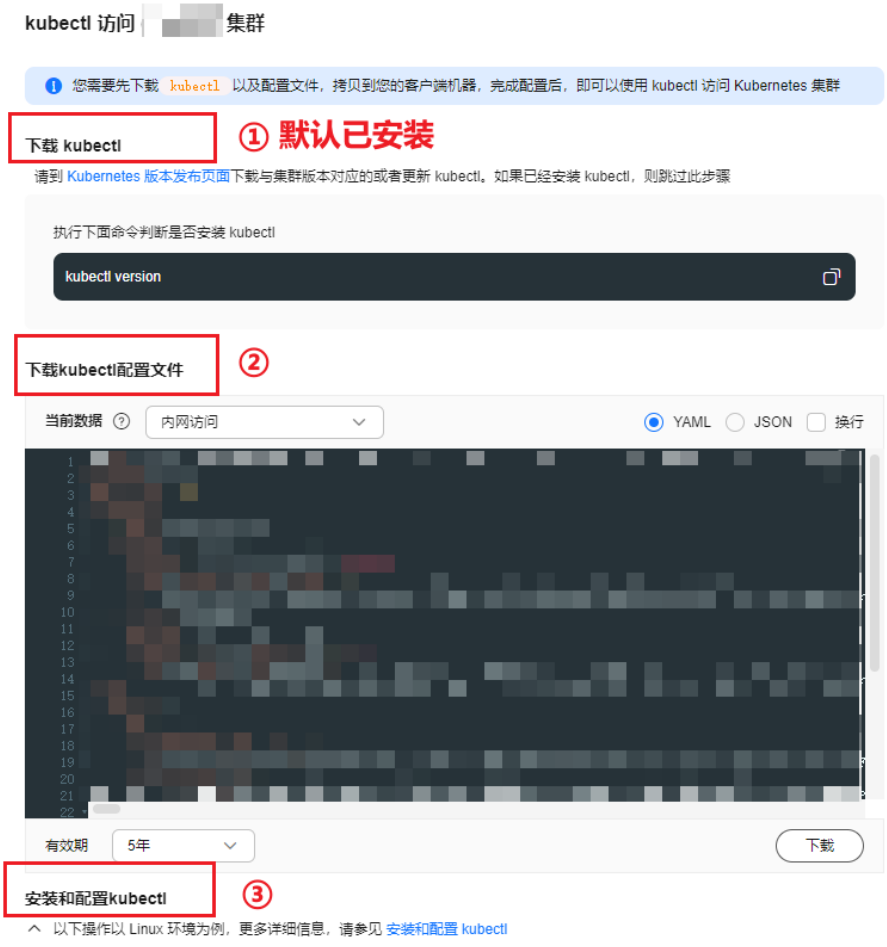
本步骤需要在节点机器，对kubectl进行集群访问配置。

1. 首先进入已创建的 CCE 集群控制版面中。根据**图4-551**的步骤进行操作，单击kubectl配置时，会弹出**图4-552**步骤页面。

图 4-551 配置中心

2. 根据**图4-552**，按步骤进行：判断是否安装 kubectl、下载kubectl配置文件、在机器中安装和配置kubectl。

图 4-552 kubectl 访问集群配置



3. 在节点机器中，输入命令，查看Kubernetes集群信息。若显示如图图4-553的内容，则配置成功。

```
kubectl cluster-info
```

图 4-553 查看 Kubernetes 集群信息正确弹出内容

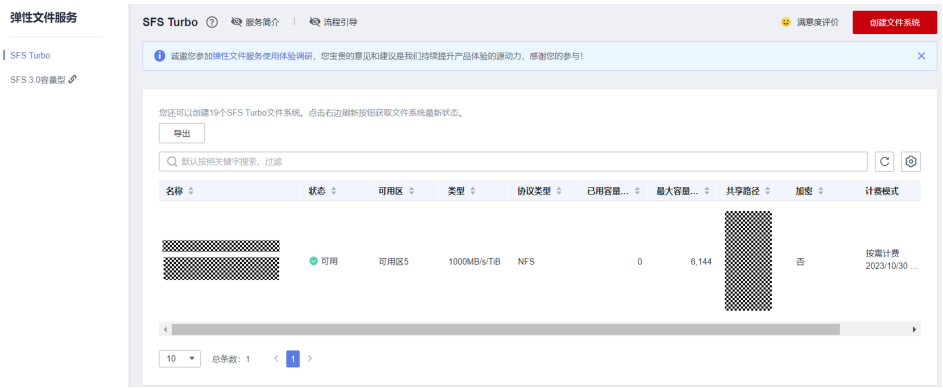
创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。

图 4-554 创建 SFS Turbo



2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

   详细操作指导请参考**创建SFS Turbo文件系统**。

   其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-555 SFS 类型和容量选择

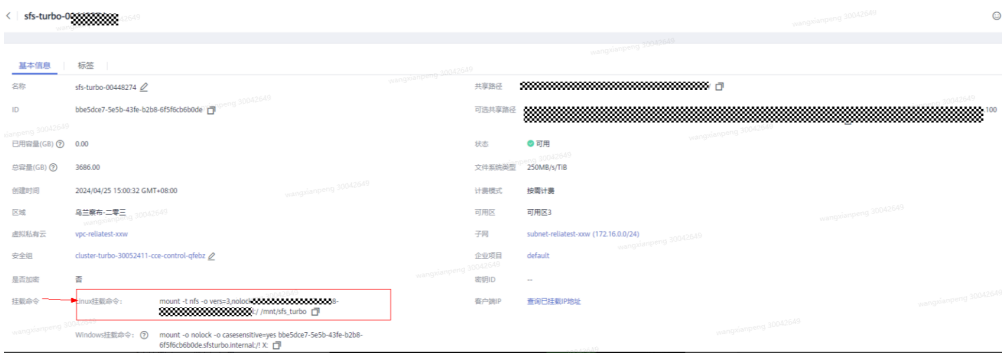
CCE 集群关联 SFS Turbo

进入已购买创建的CCE集群，选择存储，随后单击“创建存储卷声明PVC”。

- 选择“极速文件存储”，随后输入PVC名称。
- 选择“新建存储卷PV”，并单击“选择极速文件存储”。
- 进入选择页面，选择已经创建好的SFS Turbo，最后输入PV名称。

接下来需要通过访问集群节点，挂载SFS Turbo。

- 可通过ssh登录CCE集群中的某个节点（ssh使用的是eip地址）。
- 创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo
- SFS Turbo存储手动挂载到安装节点中，挂载命令如下截图：
- 挂载完成后，可通过以下步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。



4.31.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-285**所示，模型列表、对应的开源权重获取地址如**表4-283**所示。

表 4-285 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.909-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

获取对应模型的权重文件，获取链接参考**表4-283**。

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  若要下载指定版本的模型文件，则命令如下：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
  ```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.909中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
      |——AscendSpeed           # 基于AscendSpeed的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——scripts/             # 训练需要的启动脚本
            |——llama2            # llama2系列模型执行脚本的文件夹
            |——llama3            # llama3系列模型执行脚本的文件夹
            |——qwen              # Qwen系列模型执行脚本的文件夹
            |——qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
            |—— ...
            |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
            |—— install.sh       # 环境部署脚本
         |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
   |——llm_inference                # 推理代码包
   |——llm_tools                    # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}（例如使用SFS Turbo的路径：/mnt/sfs_turbo/）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}            # 模型名称
            |── data               # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}            # 模型名称
            |── logs               # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora               # lora微调输出权重
            |── sft                # 增量训练输出权重
            |── pretrain           # 预训练输出权重
|── tokenizers                    #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                        #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                  #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

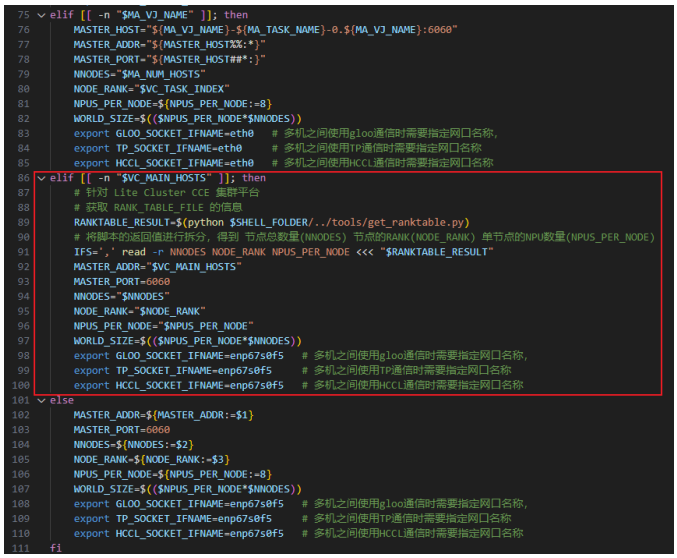
修改代码

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后。在上传代码前，需要对解压后的训练脚本代码进行修改。具体文件为：修改llm_train/AscendSpeed/scripts/dev_pipeline.sh以及新建文件llm_train/AscendSpeed/scripts/tools/get_rank_table.py 。

1. dev_pipeline.sh 具体添加代码内容以及位置，如下所示。

```
elif [[ -n "$VC_MAIN_HOSTS" ]]; then
    # 针对 Lite Cluster CCE 集群平台
    # 获取 RANK_TABLE_FILE 的信息
    RANKTABLE_RESULT=$(python $SHELL_FOLDER/../tools/get_ranktable.py)
    # 将脚本的返回值进行拆分，得到 节点总数量(NNODES) 节点的RANK(NODE_RANK) 单节点的NPU数量(NPUS_PER_NODE)
    IFS=',' read -r NNODES NODE_RANK NPUS_PER_NODE <<< "$RANKTABLE_RESULT"
    MASTER_ADDR="$VC_MAIN_HOSTS"
    MASTER_PORT=6060
    NNODES="$NNODES"
    NODE_RANK="$NODE_RANK"
    NPUS_PER_NODE="$NPUS_PER_NODE"
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

图 4-556 dev_pipeline.sh 添加代码位置和内容



```
elif [[ -n "$MA_VJ_NAME" ]]; then
    MASTER_HOST="${MA_VJ_NAME}-${MA_TASK_NAME}-0.${MA_VJ_NAME}:6060"
    MASTER_ADDR="${MASTER_HOST%%:*}"
    MASTER_PORT="${MASTER_HOST##*:}"
    NNODES="$MA_NUM_HOSTS"
    NODE_RANK="$VC_TASK_INDEX"
    NPUS_PER_NODE=${NPUS_PER_NODE:=8}
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=eth0   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=eth0     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=eth0   # 多机之间使用HCCL通信时需要指定网口名称
elif [[ -n "$VC_MAIN_HOSTS" ]]; then
    # 针对 Lite Cluster CCE 集群平台
    # 获取 RANK_TABLE_FILE 的信息
    RANKTABLE_RESULT=$(python $SHELL_FOLDER/../tools/get_ranktable.py)
    # 将脚本的返回值进行拆分，得到 节点总数量(NNODES) 节点的RANK(NODE_RANK) 单节点的NPU数量(NPUS_PER_NODE)
    IFS=',' read -r NNODES NODE_RANK NPUS_PER_NODE <<< "$RANKTABLE_RESULT"
    MASTER_ADDR="$VC_MAIN_HOSTS"
    MASTER_PORT=6060
    NNODES="$NNODES"
    NODE_RANK="$NODE_RANK"
    NPUS_PER_NODE="$NPUS_PER_NODE"
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
else
    MASTER_ADDR=${MASTER_ADDR:=$1}
    MASTER_PORT=6060
    NNODES=${NNODES:=$2}
    NODE_RANK=${NODE_RANK:=$3}
    NPUS_PER_NODE=${NPUS_PER_NODE:=8}
    WORLD_SIZE=$(($NPUS_PER_NODE*$NNODES))
    export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称,
    export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
    export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
fi
```

2. 在llm_train/AscendSpeed/scripts/tools的路径下，新建脚本文件get_rank_table.py ，具体代码如下所示。

```
import os
import re
import sys
import json

from extras.logging import get_logger, set_file_handler

logger = get_logger(__name__)


def get_rank_table():
    rank_table_file_path = os.getenv("RANK_TABLE_FILE")
    env_ip = os.getenv("ip")

    # Lite Cluster中的RANK_TABLE_FILE实际名称为 jobstart_hccl.json
    job_start_file = "jobstart_hccl.json"
    # job_start_file_path 路径默认为 "/user/config/jobstart_hccl.json"
    job_start_file_path = rank_table_file_path.rsplit("/", 1)[0] + "/" + job_start_file

    # 读取RANK_TABLE_FILE文件
    with open(job_start_file_path, 'r', encoding='utf-8') as file:
        data = json.load(file)

    # RANK_TABLE_FILE文件缺少字段，表示文件缺少关键信息
    if "status" not in data.keys() and "server_count" not in data.keys() and "server_list" not in
data.keys():
        logger.error(f"Get RANK_TABLE Error: RANK_TABLE_FILE missing key value.")
        sys.exit(1)

    # RANK_TABLE_FILE文件status不是completed，表示训练作业未创建成功
    if data["status"] != "completed":
        logger.error(f"Get RANK_TABLE Error: RANK_TABLE_FILE is incomplete, and the training job
creation was not successful.")
        sys.exit(1)

    # 获取：节点总数量:server_count 节点的RANK:server_index 单节点的NPU数量:device_count
    server_count = data["server_count"]
    server_list = data["server_list"]
    server_index = -1
    device_count = 0
    for index, server in enumerate(server_list):
        # RANK_TABLE_FILE文件中，节点ip为空
        if server["server_id"] == "":
```

```
            logger.error(f"Get RANK_TABLE Error: the IP address of the server is null.")
            sys.exit(1)

        if server["server_id"] == env_ip:
            server_index = index
            if server["device"]:
                device_count = len(server["device"])

    # RANK_TABLE_FILE文件中，节点总数量为0，表示未获取到节点
    if server_count == 0:
        logger.error(f"Get RANK_TABLE Error: the server does not exist.")
        sys.exit(1)

    # RANK_TABLE_FILE文件中，未找到对应ip的节点
    if server_index == -1:
        logger.error(f"Get RANK_TABLE Error:  the IP address {env_ip} was not found.")
        sys.exit(1)

    # RANK_TABLE_FILE文件中，NPU卡数为0，表示未获取到NPU
    if device_count == 0:
        logger.error(f"Get RANK_TABLE Error: NPU does not exist.")
        sys.exit(1)

    return server_count, server_index, device_count


if __name__ == '__main__':
    result = get_rank_table()
    print(','.join(map(str, result)))
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录服务器。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如SFS Turbo的路径：/mnt/sfs_turbo目录下，以下都以/mnt/sfs_turbo为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/mnt/sfs_turbo/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/mnt/sfs_turbo，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /mnt/sfs_turbo
   mkdir -p tokenizers/Llama2-70B
   ```

4.31.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
     'id': '1',
     'url': 'https://simple.wikipedia.org/wiki/April',
     'title': 'April',
     'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注他们自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

  若用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：
  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst

- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1，处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，若同时输入，则优先使用test_count，若都未输入，则返回处理失败 False。

上传数据到指定目录

将下载的原始数据存放在/mnt/sfs_turbo/training_data目录下。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

```
mkdir training_data
```

数据存放参考目录结构如下：

```
${workdir}
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.31.2.4 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-286 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3

表 4-287 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查containerd是否安装。
   ```
   containerd -v  # 检查containerd是否安装
   ```
   在创建CCE集群时，会选择containerd作为容器引擎，并默认给机器安装。**如尚未安装，说明机器操作系统安装错误。需要重新纳管机器，重新安装操作系统。**
3. 安装nerdctl工具。nerdctl是containerd的一个客户端命令行工具，使用方式和docker命令基本一致，可用于后续镜像构建步骤中。
   ```
   # 下载 nerdctl 工具，注意使用的是1.7.6 arm64版本
   wget https://github.com/containerd/nerdctl/releases/download/v1.7.6/nerdctl-1.7.6-linux-arm64.tar.gz

   # 将程序解压至运行目录中
   tar -zxf nerdctl-1.7.6-linux-arm64.tar.gz -C /usr/bin/

   # 查看是否安装成功
   nerdctl -v
   ```
4. 安装buildkit工具。buildkit是从Docker从公司开源出来的下一代镜像构建工具，支持OCI标准的镜像构建，nerdctl需要结合buildkit一起使用。buildkit由两部分组成：

   buildkitd（服务端）：负责镜像构建，目前支持runc和containerd作为镜像构建环境，默认是runc。

   buildkitctl（客户端）：负责解析Dockerfile文件，并向服务端buildkitd发出构建请求。

   a. 下载并解压buildkit程序。
      ```
      # 下载 buildkit 工具，注意使用的是0.15.1 arm64版本
      wget https://github.com/moby/buildkit/releases/download/v0.15.1/buildkit-v0.15.1.linux-arm64.tar.gz

      # 创建解压的目录
      ```

```
mkdir /usr/local/buildkit

# 解压到指定的目录
tar -zxf buildkit-v0.15.1.linux-arm64.tar.gz -C /usr/local/buildkit

# 授予权限
chmod -R 777 /usr/local/buildkit
```

b. 添加环境变量

```
echo 'export PATH=/usr/local/buildkit/bin:$PATH' >> /etc/profile
# 注意这里的echo 要使用单引号，单引号会原样输出，双引号会解析变量
source /etc/profile  # 使刚才配置生效
```

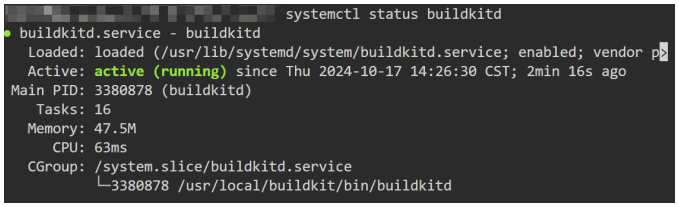
c. 创建buildkitd的启动服务。其中都是buildkitd.service的内容。复制以下全部命令并运行即可。

```
cat <<EOF > /usr/lib/systemd/system/buildkitd.service
[Unit]
Description=buildkitd
After=network.target

[Service]
ExecStart=/usr/local/buildkit/bin/buildkitd

[Install]
WantedBy=multi-user.target
EOF
```

d. 启动buildkitd的服务

```
# 重新加载Unit file
systemctl daemon-reload
# 启动服务
systemctl start buildkitd
# 开机自启动
systemctl enable buildkitd
# 查看状态
systemctl status buildkitd
```

e. 若buildkitd的服务运行状态如下图所示，则表示服务运行成功。使用Ctrl+C即可退出查看状态。



步骤二 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

containerd 容器引擎有命名空间的概念。Kubernetes 下使用的 containerd 默认命名空间是 k8s.io。所以在导入镜像时需要指定命令空间为 k8s.io，否则使用 crictl images 无法查询到。以下命令可选其一进行镜像拉取：

- 使用 containerd 自带的工具 ctr 进行镜像拉取。

  ```
  ctr -n k8s.io pull {image_url}
  ```

- 使用nerdctl工具拉取镜像。

  ```
  nerdctl --namespace k8s.io pull {image_url}
  ```

⚠ 注意

集群有多个节点，要确保每个节点都拥有镜像。

镜像获取完成后可通过如下其中一个命令进行查看：

```
# ctr 工具查看
ctr -n k8s.io image list
# 或
crictl image

# nerdctl 工具查看
nerdctl --namespace k8s.io image list
```

步骤三 构建 ModelArts Lite 训练镜像

获取模型软件包，并上传到机器SFS Turbo的目录下（可自定义路径），获取地址参考**表4-285**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.908-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面

   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```

2. 编辑llm_train/AscendSpeed中的Dockerfile文件第一行镜像地址，修改为本文档中的基础镜像地址。

   ```
   FROM {image_url}
   ```

3. （选填）编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。

   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```

4. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改安装transformers库代码的位置，放置在 chown -R ma-user:ma-group 代码的上面。避免transformers安装后由于权限问题无法访问。

   ```
   RUN git config --global http.sslVerify false && \
       git config --global user.email "you@example.com" &&  \
       git config --global user.name "Your Name" &&  \
       git clone https://gitee.com/ascend/ModelLink.git && cd ModelLink && git checkout 8f50777 && cd .. && \
       git clone https://gitee.com/lmzwhu/Megatron-LM.git  && \
       cd Megatron-LM  && \
       git checkout -f core_r0.6.0  && \
       cp -r megatron ../ModelLink/  && \
       cd ..  && \
       cd ModelLink && \
       pip install -e . && \
       cd .. && \
       git clone https://gitee.com/ascend/MindSpeed.git && \
       cd MindSpeed && \
       git checkout 4ea42a23 && \
       pip3 install -e . && cd .. && \
       pip install -e . && \
       pip install transformers==4.45.0
       chown -R ma-user:ma-group /home/ma-user/AscendSpeed && \
       chown -R ma-user:ma-group /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages && \
   ```

⚠ 注意

若要对ChatCLMv3、GLMv4系列模型进行训练时，需要修改 Dockerfile 中的 transformers 的版本。

由默认 transformers==4.45.0 修改为：transformers==4.44.2

5. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保机器可以访问公网。

   ```
   nerdctl --namespace k8s.io build -t <镜像名称>:<版本名称>  .
   ```

   nerdctl build 会去镜像仓库拉取镜像，**不会直接使用本地镜像**。构建前可以nerdctl pull 拉取测试以下镜像是否能拉取成功。

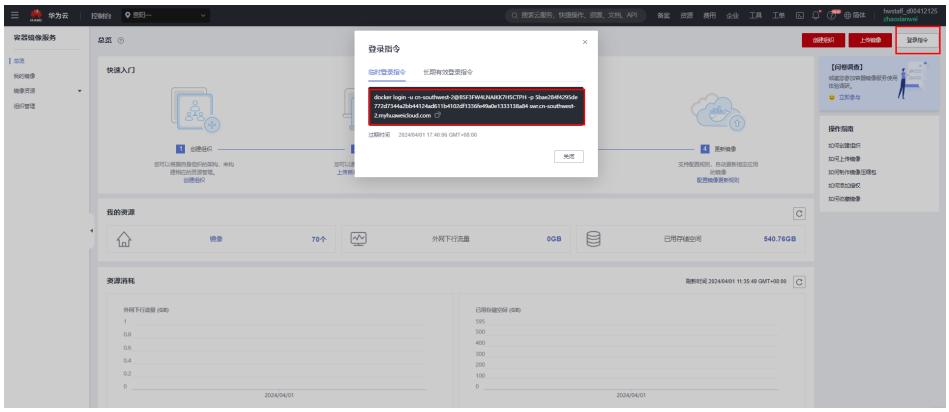
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606。

- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤四 在节点机器中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。

图 4-557 复制登录指令



由于使用的容器引擎是containerd，不再是docker，因此需要改写复制的登录指令，将docker进行替换，使用nerdctl工具。

```
# docker login 替换为：
nerdctl login
```

步骤五 修改并上传镜像

1. 在机器中输入Step4登录指令后，使用下列示例命令将镜像上传至SWR：

```
nerdctl --namespace k8s.io tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在**步骤三 构建ModelArts Lite训练镜像**中使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

nerdctl --namespace k8s.io tag ${dockerfile_image_name} swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
nerdctl --namespace k8s.io push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

nerdctl --namespace k8s.io push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240606

步骤六 编写 Config.yaml 文件

k8s有两种方式来管理对象：

- 命令式，即通过Kubectl指令直接操作对象。
- 声明式，通过定义资源YAML格式的文件来操作对象。

首先给出单个节点训练的config.yaml文件模板，用于配置pod。而在训练中，需要按照参数说明修改**${}**中的参数值。该模板使用SFS Turbo挂载方案。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob              # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                     # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980     # 保持不动
data:                                    # data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                            # job名字，需要和configmap中名字保持联系
  namespace: default                     # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano                 # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}           # 镜像地址
          imagePullPolicy: IfNotPresent  # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每
```

```
次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
        name: ${container_name}
        securityContext:                        # 容器内 root 权限
          allowPrivilegeEscalation: false
          runAsUser: 0
        env:
        - name: name
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ip
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: framework
          value: "PyTorch"
        command: ["/bin/sh", "-c"]
        args:
        - ${command}
        resources:
          requests:
            huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
            memory: ${requests_memory}                # 容器请求的最小内存
            cpu: ${requests_cpu}                      # 容器请求的最小 CPU
          limits:
            huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
            memory: ${limits_memory}                  # 容器可使用的最大内存
            cpu: ${limits_cpu}                        # 容器可使用的最大 CPU
        volumeMounts:                               # 容器内部映射路径
        - name: shared-memory-volume
          mountPath: /dev/shm
        - name: ascend-driver                       # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons                      # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: localtime
          mountPath: /etc/localtime
        - name: hccn                                # 驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: npu-smi                             # npu-smi
          mountPath: /usr/local/sbin/npu-smi
        - name: ascend-install
          mountPath: /etc/ascend_install.info
        - name: log
          mountPath: /var/log/npu/
        - name: sfs-volume
          mountPath: /mnt/sfs_turbo
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      volumes:                                      # 物理机外部路径
      - name: shared-memory-volume                  # 共享内存
        emptyDir:
          medium: Memory
          sizeLimit: "200Gi"
      - name: ascend-driver
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
```

```
      - name: ascend-install
        hostPath:
          path: /etc/ascend_install.info
      - name: log
        hostPath:
          path: /usr/slog
      - name: sfs-volume
        persistentVolumeClaim:
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
```

双个节点训练的config.yaml文件模板，用于实现双机分布式训练。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap1980-vcjob     # 前缀使用“configmap1980-”不变，后接vcjob的名字
  namespace: default                    # 命名空间自选，需要和下边的vcjob处在同一命名空间
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
data:                    #data内容保持不动，初始化完成，会被volcano插件自动修改
  jobstart_hccl.json: |
    {
      "status":"initializing"
    }
---
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: vcjob                          # job名字，需要和configmap中名字保持联系
  namespace: default                    # 和configmap保持一致
  labels:
    ring-controller.cce: ascend-1980   # 保持不动
    fault-scheduling: "force"
spec:
  minAvailable: 1
  schedulerName: volcano                # 保持不动
  policies:
    - event: PodEvicted
      action: RestartJob
  plugins:
    configmap1980:
    - --rank-table-version=v2  # 保持不动，生成v2版本ranktablefile
    env: []
    svc:
    - --publish-not-ready-addresses=true
  maxRetry: 5
  queue: default
  tasks:
  - name: main
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
```

```
      containers:
      - image: ${image_name}         # 镜像地址
        imagePullPolicy: IfNotPresent      # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
        name: ${container_name}
        securityContext:                        # 容器内 root 权限
          allowPrivilegeEscalation: false
          runAsUser: 0
        env:
        - name: name
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: ip
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
        - name: framework
          value: "PyTorch"
        command: ["/bin/sh", "-c"]
        args:
        - ${command}
        resources:
          requests:
            huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
            memory: ${requests_memory}                # 容器请求的最小内存
            cpu: ${requests_cpu}                      # 容器请求的最小 CPU
          limits:
            huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
            memory: ${limits_memory}                  # 容器可使用的最大内存
            cpu: ${limits_cpu}                        # 容器可使用的最大 CPU
        volumeMounts:                             # 容器内部映射路径
        - name: shared-memory-volume
          mountPath: /dev/shm
        - name: ascend-driver                     # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/driver
        - name: ascend-add-ons                    # 驱动挂载，保持不动
          mountPath: /usr/local/Ascend/add-ons
        - name: localtime
          mountPath: /etc/localtime
        - name: hccn                              # 驱动hccn配置，保持不动
          mountPath: /etc/hccn.conf
        - name: npu-smi                           # npu-smi
          mountPath: /usr/local/sbin/npu-smi
        - name: ascend-install
          mountPath: /etc/ascend_install.info
        - name: log
          mountPath: /var/log/npu/
        - name: sfs-volume
          mountPath: /mnt/sfs_turbo
      nodeSelector:
        accelerator/huawei-npu: ascend-1980
      volumes:                                    # 物理机外部路径
      - name: shared-memory-volume                        # 共享内存
        emptyDir:
          medium: Memory
          sizeLimit: "200Gi"
      - name: ascend-driver
        hostPath:
          path: /usr/local/Ascend/driver
      - name: ascend-add-ons
        hostPath:
          path: /usr/local/Ascend/add-ons
      - name: localtime
        hostPath:
          path: /etc/localtime
      - name: hccn
        hostPath:
          path: /etc/hccn.conf
```

```
      - name: npu-smi
        hostPath:
          path: /usr/local/sbin/npu-smi
      - name: ascend-install
        hostPath:
          path: /etc/ascend_install.info
      - name: log
        hostPath:
          path: /usr/slog
      - name: sfs-volume
        persistentVolumeClaim:
          claimName: ${pvc_name}    #已创建的PVC名称
      restartPolicy: OnFailure
  - name: work
    replicas: 1
    template:
      metadata:
        name: training
        labels:
          app: ascendspeed
          ring-controller.cce: ascend-1980  # 保持不动
      spec:
        affinity:
          podAntiAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchExpressions:
                    - key: volcano.sh/job-name
                      operator: In
                      values:
                        - vcjob
                topologyKey: kubernetes.io/hostname
        hostNetwork: true                    # 采用宿主机网络模式
        containers:
        - image: ${image_name}         # 镜像地址
          imagePullPolicy: IfNotPresent       # IfNotPresent：默认值，镜像在宿主机上不存在时才拉取；Always：每次创建Pod都会重新拉取一次镜像；Never：Pod永远不会主动拉取这个镜像
          name: ${container_name}
          securityContext:                    # 容器内 root 权限
            allowPrivilegeEscalation: false
            runAsUser: 0
          env:
          - name: name
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: ip
            valueFrom:
              fieldRef:
                fieldPath: status.hostIP
          - name: framework
            value: "PyTorch"
          command: ["/bin/sh", "-c"]
          args:
          - ${command}
          resources:
            requests:
              huawei.com/ascend-1980: "8"               # 需求卡数，key保持不变.
              memory: ${requests_memory}                # 容器请求的最小内存
              cpu: ${requests_cpu}                    # 容器请求的最小 CPU
            limits:
              huawei.com/ascend-1980: "8"               # 限制卡数，key保持不变。
              memory: ${limits_memory}                 # 容器可使用的最大内存
              cpu: ${limits_cpu}                     # 容器可使用的最大 CPU
          volumeMounts:                          # 容器内部映射路径
          - name: shared-memory-volume
            mountPath: /dev/shm
          - name: ascend-driver                   # 驱动挂载，保持不动
            mountPath: /usr/local/Ascend/driver
```

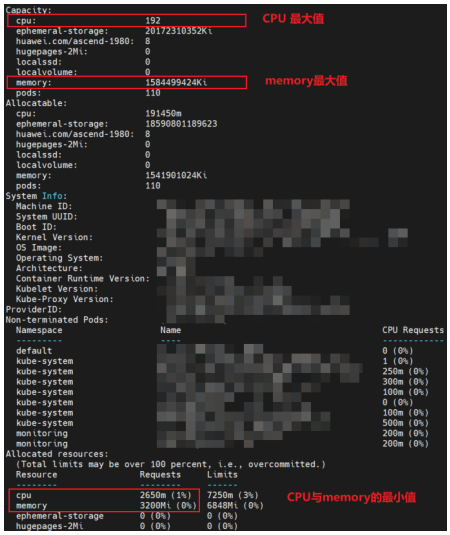
```
    - name: ascend-add-ons                  # 驱动挂载，保持不动
      mountPath: /usr/local/Ascend/add-ons
    - name: localtime
      mountPath: /etc/localtime
    - name: hccn                            # 驱动hccn配置，保持不动
      mountPath: /etc/hccn.conf
    - name: npu-smi                         # npu-smi
      mountPath: /usr/local/sbin/npu-smi
    - name: ascend-install
      mountPath: /etc/ascend_install.info
    - name: log
      mountPath: /var/log/npu/
    - name: sfs-volume
      mountPath: /mnt/sfs_turbo
  nodeSelector:
    accelerator/huawei-npu: ascend-1980
  volumes:                                  # 物理机外部路径
  - name: shared-memory-volume                      # 共享内存
    emptyDir:
      medium: Memory
      sizeLimit: "200Gi"
  - name: ascend-driver
    hostPath:
      path: /usr/local/Ascend/driver
  - name: ascend-add-ons
    hostPath:
      path: /usr/local/Ascend/add-ons
  - name: localtime
    hostPath:
      path: /etc/localtime
  - name: hccn
    hostPath:
      path: /etc/hccn.conf
  - name: npu-smi
    hostPath:
      path: /usr/local/sbin/npu-smi
  - name: ascend-install
    hostPath:
      path: /etc/ascend_install.info
  - name: log
    hostPath:
      path: /usr/slog
  - name: sfs-volume
    persistentVolumeClaim:
      claimName: ${pvc_name}    #已创建的PVC名称
  restartPolicy: OnFailure
```

参数说明：

- ${container_name} 容器名称，此处可以自己定义一个容器名称，例如ascendspeed。
- ${image_name} 为**步骤五 修改并上传镜像**中，上传至SWR上的镜像链接。
- ${command} 使用config.yaml文件创建pod后，在容器内自动运行的命令。在进行训练任务中会给出替换命令。
- /mnt/sfs_turbo 为宿主机中默认挂载SFS Turbo的工作目录，目录下存放着训练所需代码、数据等文件。
  - 同样，/mnt/sfs_turbo 也可以映射至容器中，作为容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。为方便访问两个地址可以相同。
- ${pvc_name} 为在**CCE集群关联SFS Turbo**步骤中创建的PVC名称。
- 在设置容器中需要的CPU与内存大小时，可通过运行以下命令查看申请的节点机器中具体的CPU与内存信息。

```
kubectl describe node
```

- ${requests_cpu} 指在容器中请求的最小CPU核心数量，可使用Requests中的值，例如2650m。
- ${requests_memory} 指在容器中请求的最小内存空间大小，可使用Requests中的值，例如3200Mi。
- ${limits_cpu} 指在容器中可使用的最大CPU核心数量，例如192。
- ${limits_memory} 指在容器中可使用的最大内存空间大小，例如换算成1500Gi。



4.31.3 预训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-288**所示。其他超参均有默认值，可以参考**表4-291**按照实际需求修改。

表 4-288 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 修改 config.yaml 中的${command}

请根据**步骤二 修改训练超参配置**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
  - name: work1
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
  - name: work2
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
```

```
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
- name: work3
  template:
    ...
    spec:
      ...
      containers:
      - image: ${image_name}
        command: ["/bin/sh", "-c"]
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_pretrain_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
...
spec:
  ...
  tasks:
  - name: main
    template:
      ...
      spec:
        ...
        containers:
        - image: ${image_name}
          command: ["/bin/sh", "-c"]
          args:
          - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
            sh scripts/llama2/0_pl_pretrain_13b.sh localhost 1 0; # 单机训练执行命令
```

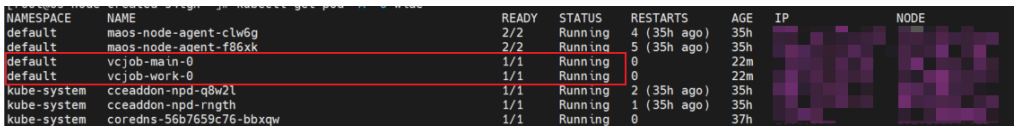
步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```



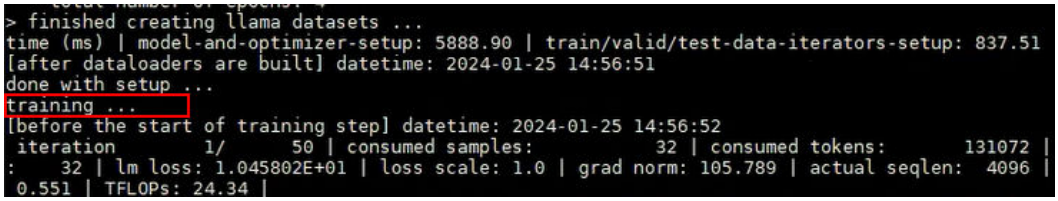
若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-558 等待模型载入



训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.31.4 SFT 全参微调训练任务

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-289**所示。其他超参均有默认值，可以参考**表4-291**按照实际需求修改。

表 4-289 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据表4-289修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
# 多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0; # 单机训练执行命令
```

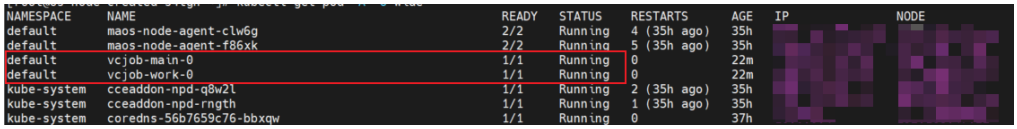
步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```



若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-559 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/       50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看sft微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.31.5 LoRA 微调训练

步骤一 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤二 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-290**所示。其他超参均有默认值，可以参考**表4-291**按照实际需求修改。

表 4-290 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。转换的Hugging Face格式权重会保存至OUTPUT_SAVE_DIR的目录中。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤三 启动训练脚本

请根据**表4-290**修改超参值后，修改config.yaml中的${command}，替换为容器中执行训练的命令。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，修改多机config.yaml模板中的${command}命令如下。多机启动需要在每个节点上执行。MASTER_ADDR为当前ssh远程主机的IP地址（**私网IP**）。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
# 仅需要修改预训练中的多机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 0; # 多机训练执行命令
 - name: work1
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 1; # 多机训练执行命令
 - name: work2
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 2; # 多机训练执行命令
 - name: work3
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_70b.sh xx.xx.xx.xx 4 3; # 多机训练执行命令
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，需要修改单机config.yaml模板中的${command}命令，可以选用单机启动，以 **Llama2-13B** 为例。

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
# 仅需要修改预训练中的单机训练执行命令即可
 - name: main
        args:
        - cd /mnt/sfs_turbo/llm_train/AscendSpeed;
          sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0; # 单机训练执行命令
```

步骤四 根据 config.yaml 启动作业

启动作业命令如下。首先会根据config.yaml创建pod，继而在pod容器内自动启动训练作业。

```
kubectl apply -f config.yaml
```

启动后，可通过以下命令获取所有已创建的pod信息。若pod已全部启动，则状态为：Running。

```
kubectl get pod -A -o wide
```

```
NAMESPACE     NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE
default       maos-node-agent-clw6g               2/2     Running   4 (35h ago)   35h
default       maos-node-agent-f86xk               2/2     Running   5 (35h ago)   35h
default       vcjob-main-0                        1/1     Running   0             22m
default       vcjob-work-0                        1/1     Running   0             22m
kube-system   cceaddon-npd-q8w2l                  1/1     Running   2 (35h ago)   35h
kube-system   cceaddon-npd-rngth                  1/1     Running   1 (35h ago)   35h
kube-system   coredns-56b7659c76-bbxqw            1/1     Running   0             37h
```

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为上述pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-560 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/mnt/sfs_turbo/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

最后，请参考**查看日志和性能**章节查看lora微调的日志和性能。

步骤五 删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.31.6 查看日志和性能

查看日志

若查看启动作业日志信息，可通过以下命令打印正在启动的日志信息。其中${pod_name}为pod信息中的NAME，例如vcjob-main-0。

```
kubectl logs -f ${pod_name}
```

训练过程中，训练日志会在最后的Rank节点打印。

图 4-561 打印训练日志

```
[before the start of training step] datetime: 2023-12-07 10:46:49
 iteration        1/      20 | consumed samples:           32 | consumed tokens:       131072 | elapsed time per iteration (ms): 97720.8 | learning rate: 4.687E-08 | global batch size:    32 | lm loss: 1.118024E+01 | loss scale: 1.0 | g
rad norm: 39.329 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 0.327 | TFLOPs: 7.66 |
[Rank 6] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0[Rank 4] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max alloca
ted: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0

[Rank 7] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
time (ms)
[Rank 0] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 5] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 1] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 3] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
[Rank 2] (after 1 iterations) memory (MB) | allocated: 12965.61865234375 | max allocated: 13261.03759765625 | reserved: 13712.0 | max reserved: 13712.0
 iteration        2/      20 | consumed samples:           64 | consumed tokens:       262144 | elapsed time per iteration (ms): 14402.9 | learning rate: 9.375E-08 | global batch size:    32 | lm loss: 1.118334E+01 | loss scale: 1.0 | g
rad norm: 39.675 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.222 | TFLOPs: 51.97 |
time (ms)
 iteration        3/      20 | consumed samples:           96 | consumed tokens:       393216 | elapsed time per iteration (ms): 14218.3 | learning rate: 1.406E-07 | global batch size:    32 | lm loss: 1.118030E+01 | loss scale: 1.0 | g
rad norm: 39.757 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.251 | TFLOPs: 52.65 |
time (ms)
 iteration        4/      20 | consumed samples:          128 | consumed tokens:       524288 | elapsed time per iteration (ms): 14315.5 | learning rate: 1.875E-07 | global batch size:    32 | lm loss: 1.117722E+01 | loss scale: 1.0 | g
rad norm: 38.376 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.29 |
time (ms)
 iteration        5/      20 | consumed samples:          160 | consumed tokens:       655360 | elapsed time per iteration (ms): 14324.0 | learning rate: 2.344E-07 | global batch size:    32 | lm loss: 1.116560E+01 | loss scale: 1.0 | g
rad norm: 39.495 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.234 | TFLOPs: 52.26 |
time (ms)
 iteration        6/      20 | consumed samples:          192 | consumed tokens:       786432 | elapsed time per iteration (ms): 14320.2 | learning rate: 2.813E-07 | global batch size:    32 | lm loss: 1.117150E+01 | loss scale: 1.0 | g
rad norm: 39.782 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.235 | TFLOPs: 52.27 |
time (ms)
 iteration        7/      20 | consumed samples:          224 | consumed tokens:       917504 | elapsed time per iteration (ms): 14233.5 | learning rate: 3.281E-07 | global batch size:    32 | lm loss: 1.114488E+01 | loss scale: 1.0 | g
rad norm: 39.099 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration        8/      20 | consumed samples:          256 | consumed tokens:      1048576 | elapsed time per iteration (ms): 14277.9 | learning rate: 3.750E-07 | global batch size:    32 | lm loss: 1.113013E+01 | loss scale: 1.0 | g
rad norm: 38.475 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.241 | TFLOPs: 52.43 |
time (ms)
 iteration        9/      20 | consumed samples:          288 | consumed tokens:      1179648 | elapsed time per iteration (ms): 14206.6 | learning rate: 4.219E-07 | global batch size:    32 | lm loss: 1.103702E+01 | loss scale: 1.0 | g
rad norm: 38.657 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.252 | TFLOPs: 52.69 |
time (ms)
 iteration       10/      20 | consumed samples:          320 | consumed tokens:      1310720 | elapsed time per iteration (ms): 14233.1 | learning rate: 4.687E-07 | global batch size:    32 | lm loss: 1.100142E+01 | loss scale: 1.0 | g
rad norm: 39.465 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.248 | TFLOPs: 52.59 |
time (ms)
 iteration       11/      20 | consumed samples:          352 | consumed tokens:      1441792 | elapsed time per iteration (ms): 14201.2 | learning rate: 5.156E-07 | global batch size:    32 | lm loss: 1.070105E+01 | loss scale: 1.0 | g
rad norm: 40.360 | actual seqlen:  4096 | number of skipped iterations:   0 | number of nan iterations:   0 | samples per second: 2.253 | TFLOPs: 52.71 |
```

训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

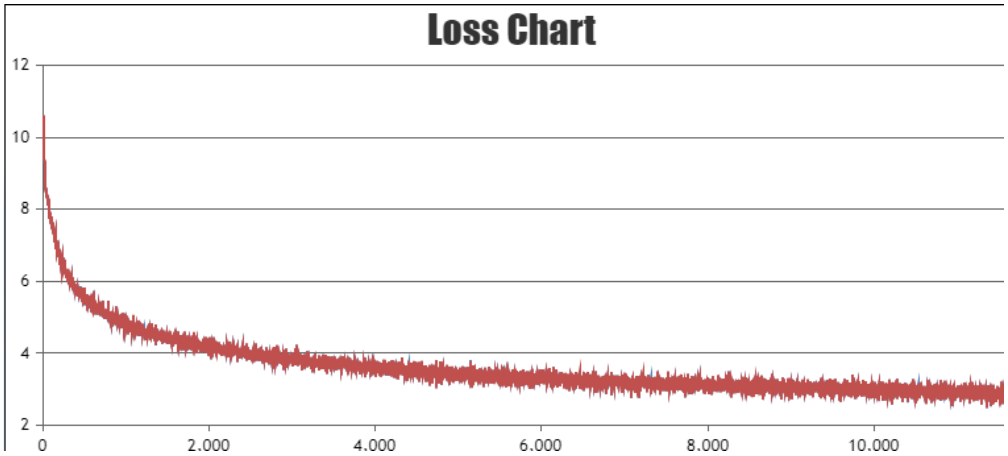
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-291**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-562**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 4-562 Loss 收敛情况（示意图）



4.31.7 训练脚本说明

4.31.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 4-291 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。若训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。

参数	示例值	参数说明
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-292所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-292 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
24	llama3.1	llama3.1-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
25		llama3.1-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

4.31.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。

- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

  ```
  def serialize_to_disk(self):
      """save idx and bin to disk"""
      startup_start = time.time()
      if not self.tokenized_dataset:
          self.tokenized_dataset = self.get_tokenized_data()
      output_bin_files = {}
      output_idx_files = {}
      builders = {}
      level = "document"
      if self.args.split_sentences:
         level = "sentence"
      logger.info("Vocab size: %s", self.tokenizer.vocab_size)
      logger.info("Output prefix: %s", self.args.output_prefix)
      for key in self.args.json_keys:
          ## 写入磁盘
  ```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

  ```
  def _filter(self, sample):
      sample = self._pre_process(sample)
      for key in self.args.json_keys:
          text = sample[key]
          doc_ids = []
          for sentence in self.splitter.tokenize(text):
              if len(sentence) > 0:
                  sentence_ids = self._tokenize(sentence)
                  doc_ids.append(sentence_ids)
          if len(doc_ids) > 0 and self.args.append_eod:
              doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
              doc_ids[-1]['attention_mask'].append(1)
              doc_ids[-1]['labels'].append(self.tokenizer.eod)
          sample[key] = doc_ids
          # for now, only input_ids are saved
          sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
      return sample
  ```

  - 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

    ```
    [
      {"text": "document"},
      {"other keys": "optional content"}
    ]
    ```

  - 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

    ```
    doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
    ```

  - 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
      "instruction": "指令（必填）",
      "input": "输入（选填）",
      "output": "模型回答（必填）",
   }
  ]
  ```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {instruction} + "\n" + {input}

  "### Response:"
  {output}
  ```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {instruction}

  "### Response:"
  ```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
```

```
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

   i. 对user和assistant的文本进行清洗

   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

   iii. input_ids是user_ids和assistant_ids的拼接

   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
```

```
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  {MOSS}
  ```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

  ```
  "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

  "### Instruction:"
  {Human}

  "### Response:"
  ```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/1_preprocess_data.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-293 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。

环境变量	示例	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.31.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋

值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-294 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径

参数	示例	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.31.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-563所示。

图 4-563 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-564所示。

图 4-564 修改 ChatGLMv3-6B tokenizer 文件

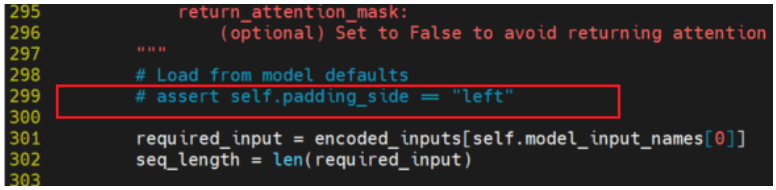
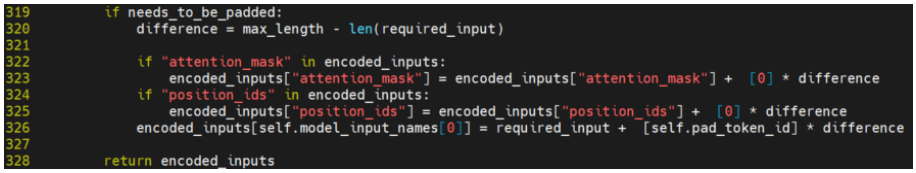


图 4-565 修改 ChatGLMv3-6B tokenizer 文件



GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下代码信息进行查找，修改后如图4-566所示。

图 4-566 修改 ChatGLMv4-9B tokenizer 文件

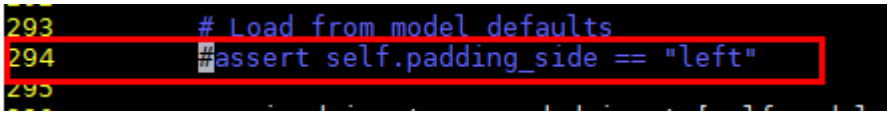
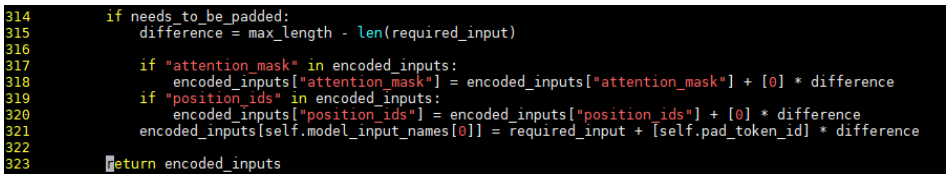


图 4-567 修改 ChatGLMv4-9B tokenizer 文件



Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下代码信息进行查找，修改后如图4-568所示。

图 4-568 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.31.8 常见错误原因和解决方法

4.31.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表4-292进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.31.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.31.8.3 工作负载 Pod 异常

Pod 状态为 Pending

当Pod状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。具体参考链接为**工作负载状态异常定位方法**。

```
NAMESPACE     NAME                                  READY   STATUS    RESTARTS      AGE   IP            NODE          NOMINATED NODE   READINESS GATES
default       maos-node-agent-clw6g                 2/2     Running   4 (35h ago)   35h   [redacted]                  <none>           <none>
default       maos-node-agent-f86xk                 2/2     Running   5 (35h ago)   35h   [redacted]                  <none>           <none>
default       vcjob-main-0                          0/1     Pending   0             6s    <none>        <none>        <none>           <none>
default       vcjob-work-0                          0/1     Pending   0             5s    <none>        <none>        <none>           <none>
kube-system   cceaddon-npd-q8w2l                    1/1     Running   2 (35h ago)   35h   [redacted]                  <none>           <none>
kube-system   cceaddon-npd-rngth                    1/1     Running   1 (35h ago)   35h   [redacted]                  <none>           <none>
```

通过以下命令打印Pod日志信息。

```
kubectl describe pod ${pod_name}
```

volcano 资源调度失败

当volcano的资源出现争抢时，会出现下图中的问题。

```
Events:
  Type     Reason            Age   From     Message
  ----     ------            ----  ----     -------
  Warning  FailedScheduling  26s   volcano  0/2 nodes are unavailable: 2 GPU topology evaluator provider davinci assignment gives no assignment.
```

解决方法：

1. 通过打印所有Pod的信息，并找到命名有scheduler字段的Pod。
   ```
   kubectl get pod -A -o wide
   ```
2. 重启该Pod，通过delete的方式删除，但随后会自动重新启动。
   ```
   kubectl delete pod -n kube-system ${pod_scheduler_name}
   ```
3. 若重启后，还是会Pending，建议多重复重启几次。

```
kube-system   volcano-admission-7c8f9fb8f5-6k8k4          1/1   Running   0             35h   [redacted]   <none>   <none>
kube-system   volcano-admission-7c8f9fb8f5-xvt8d          1/1   Running   3 (35h ago)   35h   [redacted]   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-rc8c4         1/1   Running   0             35h   [redacted]   <none>   <none>
kube-system   volcano-controller-84cbcbf9c8-vmz75         1/1   Running   1 (35h ago)   35h   [redacted]   <none>   <none>
kube-system   volcano-scheduler-c48c5c87-6kp48            1/1   Running   0             154m  [redacted]   <none>   <none>
kube-system   volcano-scheduler-c48c5c87-pqsh7            1/1   Running   0             175m  [redacted]   <none>   <none>
monitoring    kube-state-metrics-5b5498fccb-tqpzl         1/1   Running   3 (35h ago)   37h   [redacted]   <none>   <none>
monitoring    log-agent-fluent-bit-ls6fp                  2/2   Running   0             35h   [redacted]   <none>   <none>
monitoring    log-agent-fluent-bit-pb7tv                  2/2   Running   0             35h   [redacted]   <none>   <none>
```

其他实例调度失败问题

首先通过打印Pod日志信息。根据错误信息，可通过访问官网链接：**工作负载异常：实例调度失败**，进行查找。

如何删除 config.yaml 创建出的所有工作负载 Pod

若要删除config.yaml创建出的所有工作负载Pod，需要先找到config.yaml所在路径，并执行以下命令。

```
kubectl delete -f config.yaml
```

4.32 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.908）

4.32.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc3。
- Server驱动版本要求23.0.6。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-295 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	cann_8.0.rc3

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表4-296所示。

表 4-296 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如表4-297所示。

表 4-297 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
39	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	x	x	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	x	x	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
42	llama-3.1-405B	√	√	x	x	x	https://huggingface.co/hugging-quants/Meta-Llama-3.1-405B-Instruct-AWQ-INT4
43	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
44	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
45	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
46	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
47	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动，glm4-9b模型仅支持单卡启动。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.908中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
        ├── build.sh               # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant               # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
  ├──AutoAWQ             # W4A16量化工具
     ├──convert_awq_to_npu.py   # awq权重转换脚本
     ├──quantize.py   # 昇腾适配的量化转换脚本
     ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh            #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py             #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型基于Lite Server适配PyTorch NPU训练指导**。

4.32.2 部署推理服务

4.32.2.1 非分离部署推理服务

本章节介绍如何使用vLLM 0.5.0框架部署并启动推理服务。

什么是非分离部署

全量推理和增量推理在同一节点上进行。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。

- 安装过程需要连接互联网git clone，确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-295**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip到主机中，包获取路径请参见**表4-296**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-297**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：
```
df -h
```

Step4 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip。
```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```

修改build.sh，增加如下命令：

```
pip install outlines==0.0.46
```

图 4-569 修改 build.sh

```
#!/usr/bin/env bash

sed -i 's/\r//g' ./vllm_install.patch
git config --global http.sslVerify "false"
git clone -b v0.5.0 https://github.com/vllm-project/vllm.git vllm-gpu-0.5.0
mv vllm_install.patch vllm-gpu-0.5.0/
cd vllm-gpu-0.5.0
rm -rf pyproject.toml
git apply ./vllm_install.patch
pip install outlines==0.0.46
pip install -e .
cd ..
pip install ascend_vllm-*-py3-none-any.whl

```

执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

Step5 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

Step6 启动推理服务

1. 进入容器。
   ```
   docker exec -it -u ma-user ${container-name} /bin/bash
   ```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
   ```
   npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```
   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

   图 4-570 查询结果

   ```
   +------------------------------------------------------------------------------------------------+
   | npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
   +---------------------------+---------------+----------------------------------------------------+
   | NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
   | Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
   +===========================+===============+====================================================+
   | 4     910B2               | OK            | 91.4        50                0    / 0           |
   | 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
   +===========================+===============+====================================================+
   | 5     910B2               | OK            | 92.5        51                0    / 0           |
   | 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
   +===========================+===============+====================================================+
   +---------------------------+---------------+----------------------------------------------------+
   | NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
   +===========================+===============+====================================================+
   | 4       0                 | 10915         | python                   | 55400                   |
   +===========================+===============+====================================================+
   | 5       0                 | 21273         | python                   | 55388                   |
   +===========================+===============+====================================================+
   ```
4. 配置环境变量。
   ```
   export DEFER_DECODE=1
   # 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增
   ```

```
加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子（全量prefill阶段的flash-attention）是否使用高精度模式；默认值为1表示开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，会影响首token时延增加5%~10%。

export USE_IFA_HIGH_PRECISION_MODE=1
# IFA算子（增量decode阶段的flash-attention）是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B、Qwen2-57b、Qwen2-72B，在长序列下需要开启，否则会有概率性精度异常；其他模型不建议开启，会影响增量时延增加5%~10%。

export USE_PREFIX_HIGH_PRECISION_MODE=1
# 针对Qwen2-7B、Qwen2-72B模型，在开启prefix-caching时，需要同时使用带有prefix-caching的高精度attention算子避免精度异常。需要和prefix-caching特性一起使用，如果不使用prefix-caching特性则不配置该环境变量
```

5. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。
6. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**

     在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     （1）非多模态

     ```
     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

     （2）llava多模态

     ```
     export VLLM_IMAGE_FETCH_TIMEOUT=100
     export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
     export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False

     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --image-input-type pixel_values \
     --image-token-id 32000 \
     --image-input-shape 1,3,336,336 \
     ```

```
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:False；llava多卡启动时需要关闭虚拟内存扩展；开启时可能提升模型性能。允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。
- --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
- --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

  [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。
- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

  ```
  最小处理单元为14*14
  【llava1.5】
  336*336图像 ==(336/14=24)>> 24*24=576
  672*672图像 ==(672/14=48)>> 48*48=2304
  【llava1.6】
  336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336
  ==(336/14=24)>> 336/14+2*24*24=1176
  672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336
  ==(336/14=24)>> 672/14+5*24*24=2928
  ```

- --chat-template：llava对话构建模板。

- **方式二：通过vLLM服务API接口启动服务**

  在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

  ```
  python -m vllm.entrypoints.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --host=${docker_ip} \
  --port=8080 \
  --gpu-memory-utilization=0.9 \
  --trust-remote-code
  ```

- **方式三：多机部署vLLM服务API接口启动服务**（可选）

当单机显存无法放下模型权重时，可选用该种方式部署；该种部署方式，需要机器在同一个集群，NPU卡之间IP能够ping通方可，具体步骤如下：

i. 查看卡IP。
```
for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
```

ii. 检查卡之间的网络是否通。
```
# 在另一个节点上执行，29.81.3.172是上一步输出的ipaddr的值
hccn_tool -i 0 -ping -g address 29.81.3.172
```

iii. 启动Ray集群。
```
# 指定通信网卡，使用ifconfig查看，找到和主机IP一致的网卡名
export GLOO_SOCKET_IFNAME=enp67s0f5
export TP_SOCKET_IFNAME=enp67s0f5
# 指定可使用的卡
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
# 将其中一个节点设为头节点
ray start --head --num-gpus=8
# 在其他节点执行
ray start --address='10.170.22.18:6379' --num-gpus=8
```

   - --num-gpus：要跟ASCEND_RT_VISIBLE_DEVICES指定的可用卡数一致。
   - --address：头节点IP+端口号，头节点创建成功后，会有打印。

iv. 正常启服务即可。

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。

- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用prefix-caching特性，不添加表示不使用。开启该特性后，如果模型长度>8192，则需要在启动推理服务前添加如下环境变量降低显存占用；否则在长序列的推理中会触发Out of Memory，导致推理服务不可用。
  ```
  export USE_PREFIX_HIGH_PRECISION_MODE=1
  ```
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:
Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on
http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-298**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

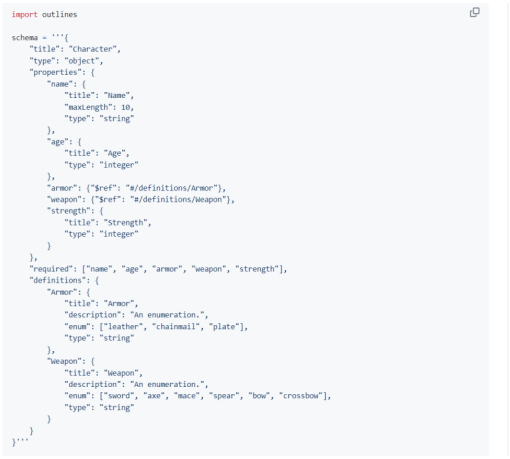
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-298 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。

参数	是否必选	默认值	参数类型	描述
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必须属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-571** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.32.2.2 分离部署推理服务

本章节介绍如何使用vLLM 0.5.0框架部署并启动推理服务。

什么是分离部署

大模型推理是自回归的过程，有以下两阶段：

- Prefill阶段（全量推理）

  将用户请求的prompt传入大模型，进行计算，中间结果写入KVCache并推出第1个token，属于计算密集型。

- Decode阶段（增量推理）

  将请求的前1个token传入大模型，从显存读取前文产生的KVCache再进行计算，属于访存密集型。

分离部署场景下，全量推理和增量推理在不同的容器上进行，用于提高资源利用效率。

分离部署的实例类型启动分为以下三个阶段：

1. **Step6 启动全量推理实例**：必须为NPU实例，用于启动全量推理服务，负责输入的全量推理。全量推理占用至少1个容器。
2. **Step7 启动增量推理实例**：必须为NPU实例，用于启动增量推理服务，负责输入的增量推理。增量推理占用至少1个容器。
3. **Step8 启动scheduler实例**：可为CPU实例，用于启动api-server服务，负责接收推理请求，向全量或增量推理实例分发请求，收集推理结果并向客户端返回推理结果。服务调度实例不占用显卡资源，建议增加1个容器，也可以在全量推理或增量推理的容器上启动。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂
```

```
载
npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
```

如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-295**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip到主机中，包获取路径请参见**表4-296**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-297**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。
3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：

```
df -h
```

Step4 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

Step5 生成 ranktable

介绍如何生成ranktable，以1p1d-tp2分离部署模式为例。当前1p1d分离部署模式，全量节点和增量节点分别占用2张卡，一共使用4张卡。

- 配置tools工具根目录环境变量

  使用AscendCloud-LLM发布版本进行推理，基于AscendCloud-LLM包的解压路径配置tool工具根目录环境变量：

  ```
  export LLM_TOOLS_PATH=${root_path_of_AscendCloud-LLM}/llm_tools
  ```

  其中，`${root_path_of_AscendCloud-LLM}`为AscendCloud-LLM包解压后的根路径。

  当使用昇腾云的官方指导文档制作推理镜像时，可直接基于该固定路径配置环境变量：

  ```
  export LLM_TOOLS_PATH=/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools
  ```

- 获取每台机器的rank_table

  在每个机器生成global rank_table信息与local rank_table信息。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode gen --prefill-server-list 4,5 --decode-server-list 6,7 --api-server --save-dir ./save_dir
  ```

  执行后，会生成一个global_ranktable.json文件和使用实例个数的local_ranktable.json文件；如果指定了`--api-server`，还会生成一个local_ranktable_host.json文件用于确定服务入口实例。

  ./save_dir 生成ranktable文件如下（假设本地主机ip为10.**.**.18）。

  ```
  global_ranktable_10.**.**.18.json       # global rank_table
  local_ranktable_10.**.**.18_45.json     # 全量节点local rank_table
  local_ranktable_10.**.**.18_67.json     # 增量节点local rank_table
  local_ranktable_10.**.**.18_host.json   # api-server
  ```

- 合并不同机器的global rank_table(可选)

  如果分离部署在多台机器，获取每台机器的rank_table后，合并各个机器的global rank_table得到完整的global rank_table。

  ```
  python ${LLM_TOOLS_PATH}/PD_separate/pd_ranktable_tools.py --mode merge --global-ranktable-list ./ranktable/global_ranktable_0.0,0,0.json ./ranktable/global_ranktable_1.1.1.1.json --save-dir ./save_dir
  ```

  pd_ranktable_tools.py的入参说明如下。

  - --mode：脚本的处理模式，可选值为`gen`或者`merge`。`gen`模式表示生成rank_table文件，`merge`模式表示合并global rank_table文件。
  - --save-dir：保存生成的rank_table文件的根目录，默认为当前目录。
  - --api-server：仅在`gen`模式有效，可选输入，当存在该输入时，表示分离部署的服务入口在该机器。注意，在多台机器启动分离部署时，只能有一台机器存在服务入口。当存在该输入时，会生成local_ranktable_xx_host.json文件，用于在启动推理服务时确定服务入口实例。
  - --prefill-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm全量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应全量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定全量实例。
  - --decode-server-list：仅在`gen`模式有效，可选输入，后续入参表示若干个vllm增量实例，使用空格隔开，每个vllm实例的数字表示使用的昇腾卡device_id，使用多个昇腾卡时，device_id之间使用`,`分隔开。当存在该输入时，会生成对应增量实例个数的local_ranktable_xx_yy.json文件，用于在启动推理服务时确定增量实例。
  - --global-ranktable-list：仅在`merge`模式有效，必选输入，后续入参表示需要合并的global rank_table，使用空格分隔开。

  执行后，会生成完成合并的global_ranktable_merge.json文件。

- global_rank_table.json格式说明

  server_group_list的长度必须为3，第一个元素(group_id="0")代表Scheduler实例的ip信息，只能有一个实例。

  第二个元素(group_id="1")代表全量实例信息，长度即为全量实例个数。其中需要配置每个全量实例的ip信息以及使用的device信息。rank_id为逻辑卡号，必然从0开始计算，device_id为物理卡号，device_ip则通过上面的hccn_tool获取。

  第三个元素(group_id="2")代表增量实例信息，长度即为增量实例个数。其余信息和全量类似。

  global_rank_table.json具体示例如下：

```
{
   "version": "1.0",
   "status": "completed",
   "server_group_list": [
      {
         "group_id": "0",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost"
            }
         ]
      },
      {
         "group_id": "1",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "4",
                     "device_ip": "10.**.**.22",
                     "rank_id": "0"
                  },
                  {
                     "device_id": "5",
                     "device_ip": "10.**.**.23",
                     "rank_id": "1"
                  }
               ]
            }
         ]
      },
      {
         "group_id": "2",
         "server_count": "1",
         "server_list": [
            {
               "server_id": "localhost",
               "server_ip": "localhost",
               "device": [
                  {
                     "device_id": "6",
                     "device_ip": "29.**.**.56",
                     "rank_id": "0"
                  },
                  {
                     "device_id": "7",
                     "device_ip": "29.**.**.72",
                     "rank_id": "1"
                  }
               ]
            }
```

```
        ]
      }
   ]
}
```

- local_rank_table.json格式说明

  每个全量/增量实例都需要local_rank_table.json。下面以某一个增量实例为例，需要和global_rank_table.json中的增量信息完全对应，group_id为0。

```
{
   "version": "1.0",
   "status": "completed",
   "group_id": "0",
   "server_count": "1",
   "server_list": [
      {
         "server_id": "localhost",
         "server_ip": "localhost",
         "device": [
            {
               "device_id": "6",
               "device_ip": "29.**.**.56",
               "rank_id": "0"
            },
            {
               "device_id": "7",
               "device_ip": "29.**.**.72",
               "rank_id": "1"
            }
         ]
      }
   ]
}
```

Step6 启动全量推理实例

以下介绍如何启动全量推理实例。

1. 启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci4、davinci5。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，$

{container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动全量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_45.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=8088 \
   --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量或增量推理实例启动的--port参数相关。--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(`--port`)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重

- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

参数定义和使用方式与vLLM0.5.0版本一致，此处介绍关键参数。详细参数解释请参见**https://github.com/vllm-project/vllm/blob/main/vllm/engine/arg_utils.py**。

Step7 启动增量推理实例

1. 启动增量推理容器

   启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了2张卡davinci6、davinci7。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

     说明

     - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
     - driver及npu-smi需同时挂载至容器。
     - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
   - --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动增量推理实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_67.json

export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
    --model=${model} \
    --tensor-parallel-size=2 \
    --max-model-len=4096 \
    --max-num-seqs=256 \
    --max-num-batched-tokens=4096 \
    --host=0.0.0.0 \
    --port=8089 \
    --served-model-name ${served-model-name}
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- RANK_TABLE_FILE_PATH：local rank_table的路径，必选。当实例类型为全量推理实例或者增量推理实例，local rank_table配置local_ranktable_xx_yy.json文件，其中xx表示当前实例的IP地址，yy表示当前实例使用的device_id信息；当实例类型为服务入口实例，local rank_table配置local_ranktable_xx_host.json文件，其中xx表示当前实例的IP地址。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用,（英文逗号）分隔开作为该环境变量的输入。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下：

- --host：服务部署的IP地址
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量`USE_OPENAI=1`时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

Step8 启动 scheduler 实例

建议在PD服务（即全量推理和增量推理服务）启动后，再启动scheduler服务。

1. 启动scheduler容器。启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

   参数说明：

   - --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了0张卡。
   - -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

     说明

     - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
     - driver及npu-smi需同时挂载至容器。
     - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
   - --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
   - {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

2. 进入容器。

```
docker exec -it -u ma-user ${container-name} /bin/bash
```

3. 启动scheduler实例，命令如下。

```
export GLOBAL_RANK_TABLE_FILE_PATH=global_ranktable_10.**.**.18.json
export RANK_TABLE_FILE_PATH=local_rank_table_10.**.**.18_host.json
export NODE_PORTS=8088,8089
export USE_OPENAI=1

sh AscendCloud-LLM/llm_tools/PD_separate/start_servers.sh \
   --model=${model} \
   --tensor-parallel-size=2 \
   --max-model-len=4096 \
   --max-num-seqs=256 \
   --max-num-batched-tokens=4096 \
   --host=0.0.0.0 \
   --port=9000 \
   --served-model-name ${served-model-name}

# 当前schduler端口port对外提供推理服务，故使用该端口进行性能验证和精度对齐
```

其中环境变量说明如下：

- GLOBAL_RANK_TABLE_FILE_PATH：global rank_table的路径，必选。不同实例类型的global rank_table均一致。
- NODE_PORTS：仅在服务入口实例生效，用于与全量推理实例、增量推理实例的信息交互。该参数入参为形如{port1},{port2},{portn}的字符串，与全量/增量推理实例启动的--port参数相关，--port表示服务部署的端口。每个全量/增量推理实例基于配置的端口号(--port)启动服务，并按照global rank_table中的全量实例、增量实例的顺序，对全量推理实例、增量推理实例启动的端口号进行排序，端口之间用`,`分隔开作为该环境变量的输入。当前端口9000是对外服务端口，而8088、8089则为scheduler调度推理服务端口。
- USE_OPENAI：仅在服务入口实例生效，用于配置api-server服务是否使用openai服务，默认为1。当配置为1时，启动服务为openai服务；当配置为0时，启动服务为vllm服务。

其中常见的参数如下，

- --host：服务部署的IP
- --port：服务部署的端口，注意如果不同实例部署在一台机器上，不同实例需要使用不同端口号
- --model：HuggingFace下载的官方权重
- --max-num-seqs：同时处理的最大句子数量
- --max-model-len：模型能处理的请求输入+输出的token长度
- --max-num-batched-tokens：最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192
- --tensor-parallel-size：模型并行数量
- --served-model-name：openai服务的model入参名称，仅在环境变量USE_OPENAI=1时候生效。
- --quantization：如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**、**使用SmoothQuant量化**或**使用GPTQ量化**章节对模型做量化处理。

说明

- 全量和增量节点的local rank table必须一一对应。
- 全量和增量节点不能使用同一个端口。
- scheduler实例中NODE_PORTS=8088,8089；端口设置顺序必须与global rank table文件中各全量和增量节点顺序一致，否则会报错。

Step9 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-299**。

通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
```

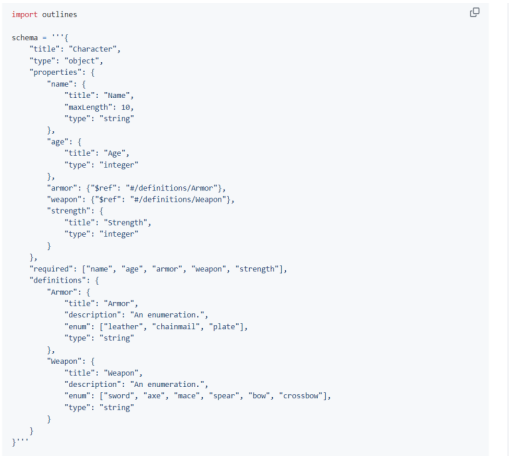
```
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-299 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必须属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-572** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.32.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step4 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```

   参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai、openai-chat等。本文档使用的推理接口是vllm，而llava多模态推理接口是openai-chat。
- --host ${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口8080。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-573 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

方法二：使用generate_dataset.py脚本生成数据集方法：

客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```

generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

```
cd benchmark_tools
conda activate python-3.9.10
```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_serving.py --backend vllm --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。

- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-574 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

投机推理 benchmark 验证

本章节介绍如何进行投机推理benchmark验证。

1. 已经上传投机推理benchmark验证脚本到推理容器中。如果在**Step4 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下,切换一个conda环境。
```
cd benchmark_tools
conda activate python-3.9.10
```
3. 运行验证脚本speculative_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
```
python speculative_benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --dataset human-eval-v2-20210705.jsonl \ --tokenizer /path/to/tokenizer  --num-prompts 80 \
--output_len 4096 --trust-remote-code
```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径，推荐使用human-eval-v2-20210705.jsonl数据集，数据集可从https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz下载压缩包解压获得。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --num-prompts：某个频率下请求数，默认80。
   - --output_len：输出长度,默认是1024。
   - --trust-remote-code：是否相信远程代码。

   a. 脚本运行完后，测试结果保存在终端输出。

单条请求性能测试

针对openai的/v1/completions以及/v1/chat/completions两个非流式接口，请求体中可以添加可选参数"return_latency"，默认为false，如果指定该参数为true，则会在相应请求的返回体中返回字段"latency"，返回内容如下：

1. prefill_latency（首token时延）：请求从到达服务开始到生成首token的耗时
2. model_prefill_latency（模型计算首token时延）：服务从开始计算首token到生成首token的耗时
3. avg_decode_latency（平均增量token时延）：服务计算增量token的平均耗时
4. time_in_queue（请求排队时间）：请求从到达服务开始到开始被调度的耗时
5. request_latency（请求总时延）：请求从到达服务开始到结束的耗时

以上指标单位均是ms，保留2位小数。

4.32.4 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
bash install.sh
```

3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下（可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤5进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- host：与起服务的host保持一致，比如起服务为0.0.0.0,host设置也为0.0.0.0。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen   model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

   ```
   cd opencompass
   python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
   ```

   output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。

   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 4-300 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远(小于10）认为分数有效。

4.32.5 推理模型量化

4.32.5.1 使用 AWQ 量化

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-297**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel, W8A16 per-channel

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

AutoAWQ量化工具的适配代码存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/AutoAWQ目录下。

1、在容器中使用ma-user用户运行以下命令下载并安装AutoAWQ源码。

```
cd llm_tools/AutoAWQ
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式离线转换（可选）

在GPU上AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.32.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-297**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数

- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step6 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   --dtype=float16
   ```

4.32.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-297**。

Step1 使用 tensorRT 量化工具进行模型量化，必须在 GPU 环境

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.32.5.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**表4-297**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
```
pip install auto-gptq
pip install --upgrade accelerate optimum transformers
```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
```
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
model_id = "meta-llama/CodeLlama-34b-hf"
tokenizer = AutoTokenizer.from_pretrained(model_id)
gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
```
dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1,
damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
```
quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
quantization_config=gptq_config)
```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
```
quantized_model.save_pretrained("CodeLlama-34b-hf")
tokenizer.save_pretrained("CodeLlama-34b-hf")

# if quantized with device_map set
quantized_model.to("cpu")
quantized_model.save_pretrained("CodeLlama-34b-hf")
```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step6 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
          max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.32.6 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-301 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	--		8	128
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
32	deepseek-coder-instruct-3 3b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x 7b	4	8	2	32
36	gemma-2 b	1	64	1	128
37	gemma-7 b	1	8	1	32
38	falcon-11 b	1	8	1	64

4.32.7 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`，

  解决方法：该问题通过将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()
- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.33 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.908）

4.33.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。
- 适配的CANN版本是cann_8.0.rc3。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-302**所示。

表 4-302 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
8	llama3-70b	√	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
16	qwen-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
24	qwen1.5-110b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	√	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	x	https://huggingface.co/google/gemma-2b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
32	gemma-7b	√	x	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	x	x	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持W8A16量化	是否支持kv-cache-int8量化	开源权重获取地址
41	llama3.1-70b	√	x	x	√	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
43	llava-1.5-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-7b-hf/tree/main
44	llava-1.5-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-1.5-13b-hf/tree/main
45	llava-v1.6-7b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-7b-hf/tree/main
46	llava-v1.6-13b	√	x	x	x	x	https://huggingface.co/llava-hf/llava-v1.6-vicuna-13b-hf/tree/main
47	llava-v1.6-34b	√	x	x	x	x	llava-hf/llava-v1.6-34b-hf at main (huggingface.co)

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动，glm4-9b模型仅支持单卡启动。

操作流程

图 4-575 操作流程图

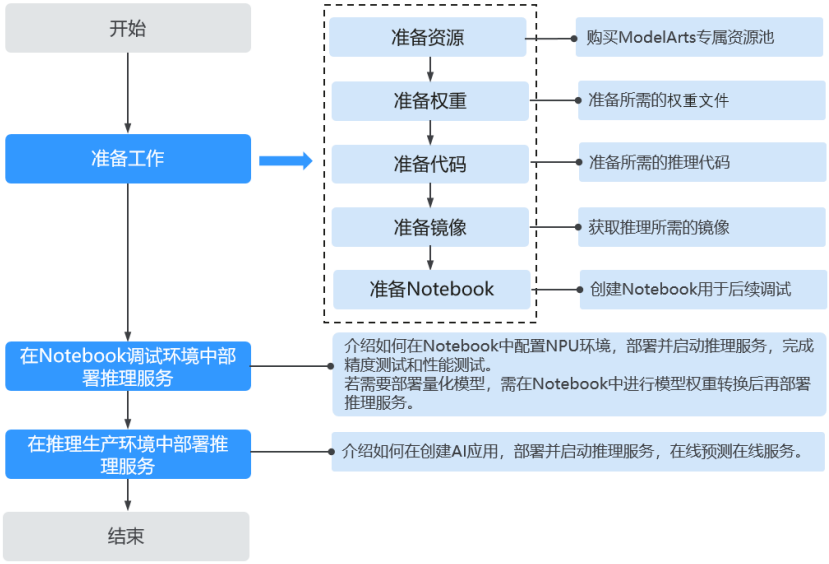


表 4-303 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.908-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 如果需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.33.2 准备工作

4.33.2.1 准备资源

创建专属资源池

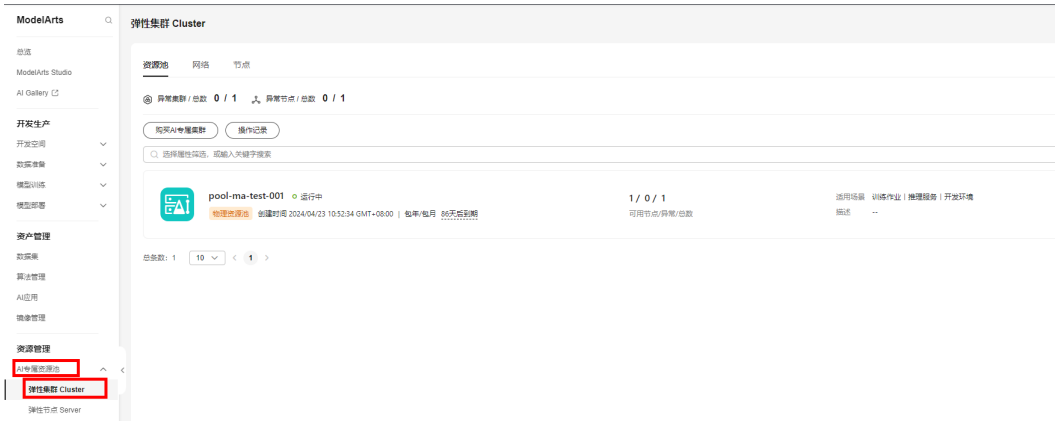
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 4-576 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 4-577 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.33.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.33.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-304**所示。

表 4-304 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.908中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
      ├── vllm_npu       # 推理源码
      ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
      ├── build.sh               # 推理构建脚本
      ├── vllm_install.patch       # 社区昇腾适配的补丁包
      ├── Dockerfile             # 推理构建镜像dockerfile
      ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
```

```
├──AutoSmoothQuant  # W8A8量化工具
    ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
    ├── autosmoothquant                 # 量化代码
    ├── build.sh                        # 安装量化模块的脚本
├──AutoAWQ              # W4A16量化工具
    ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──quantize.py   # 昇腾适配的量化转换脚本
    ├──build.sh      # 安装量化模块的脚本
├──llm_evaluation    # 推理评测代码包
  ├──benchmark_tools    #性能评测
    ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
    ├── benchmark_parallel.py   # 评测静态性能脚本
    ├── benchmark_serving.py    # 评测动态性能脚本
    ├── benchmark_utils.py      # 抽离的工具集
    ├── generate_datasets.py    # 生成自定义数据集的脚本
    ├── requirements.txt        # 第三方依赖
  ├──benchmark_eval #精度评测
    ├──opencompass.sh       #运行opencompass脚本
    ├──install.sh           #安装opencompass脚本
    ├──vllm_api.py          #启动vllm api服务器
    ├──vllm.py              #构造vllm评测配置脚本名字
```

4.33.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-305 基础容器镜像地址

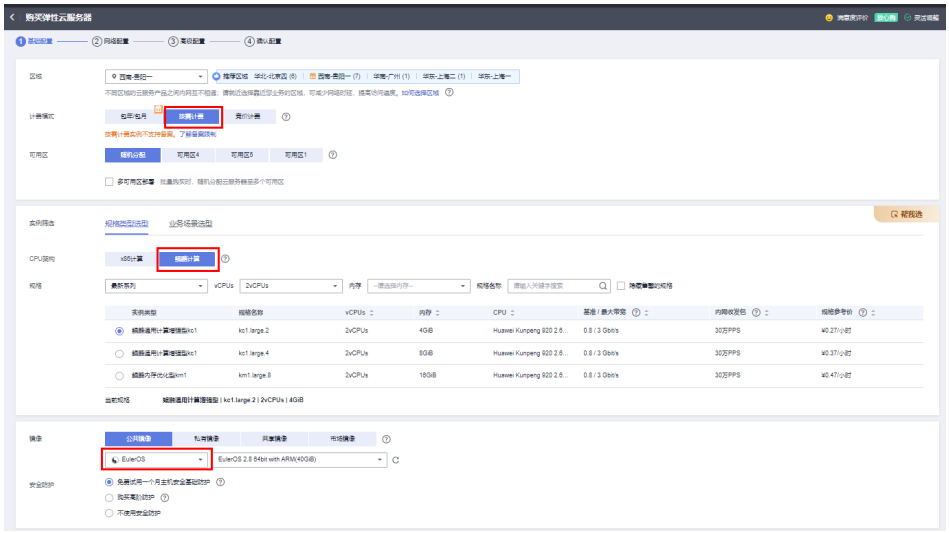
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	CANN：cann_8.0.rc3

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-578 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-579 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-304**。

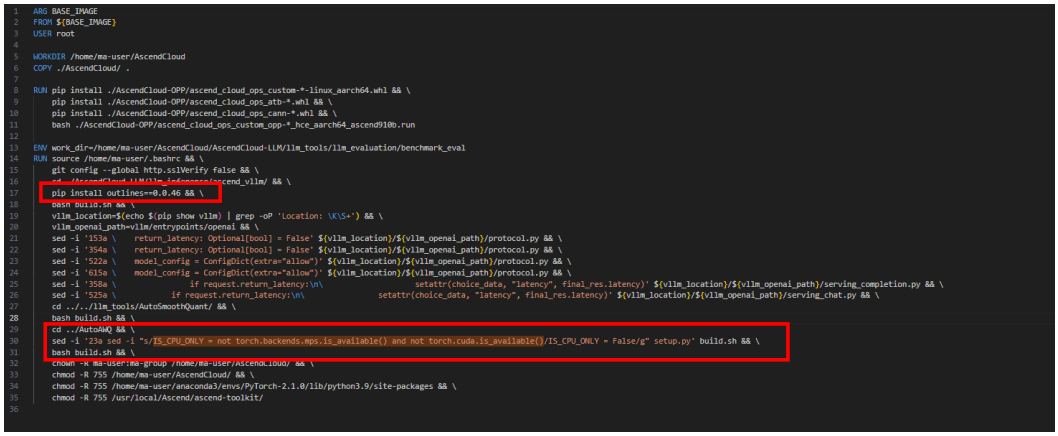
解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.908-xxx.zip和算子包AscendCloud-OPP-6.3.908-xxx.zip。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/
```

修改Dockerfile，增加如下命令：

```
RUN source /home/ma-user/.bashrc && \
    git config --global http.sslVerify false && \
    cd ./AscendCloud-LLM/llm_inference/ascend_vllm/ && \
    pip install outlines==0.0.46 && \
    bash build.sh && \
    ...
    cd ../../llm_tools/AutoSmoothQuant/ && \
    bash build.sh && \
    cd ../AutoAWQ && \
    sed -i '23a sed -i "s/IS_CPU_ONLY = not torch.backends.mps.is_available() and not torch.cuda.is_available()/IS_CPU_ONLY = False/g" setup.py' build.sh && \
    bash build.sh && \
    ...
```

图 4-580 修改 dockerfile



修改build_image.sh内容，将'ENTRYPOINT ["/home/mind/model/run_vllm.sh"]'修改为'ENTRYPOINT sh /home/mind/model/run_vllm.sh'。

图 4-581 修改 build_image.sh

```
#!/bin/bash

OPTIONS=$(getopt -n "$0" -o i:e:n --long base-image:,specify-enrtypoint:,image-name: -- "$@")

if [ $? != 0 ]; then
    echo "args error"
    exit 1
fi

eval set -- "$OPTIONS"

while true; do
    case "$1" in
        -i|--base-image)
            base_image="$2"
            shift 2
            ;;
        -e|--specify-enrtypoint)
            specify_enrtypoint="$2"
            shift 2
            ;;
        -n|--image-name)
            image_name="$2"
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "unknown options"
            exit 1
            ;;
    esac
done

if [ -z "$base_image" ]; then
    echo "--base-image not specified"
    exit 1
fi

if [ -z "$image_name" ]; then
    echo "--image-name not specified"
    exit 1
fi

if [ -n "$specify_enrtypoint" ]; then
    echo 'ENTRYPOINT sh /home/mind/model/run_vllm.sh' >> Dockerfile
fi

cd ../../../../
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile ./
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/.dockerignore ./

docker build -t $image_name --build-arg BASE_IMAGE=$base_image .

rm -f Dockerfile
rm -f .dockerignore

```

执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
sh build_image.sh --base-image=${base_image} --image-name=${image_name} --specify-enrtypoint=True
```

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

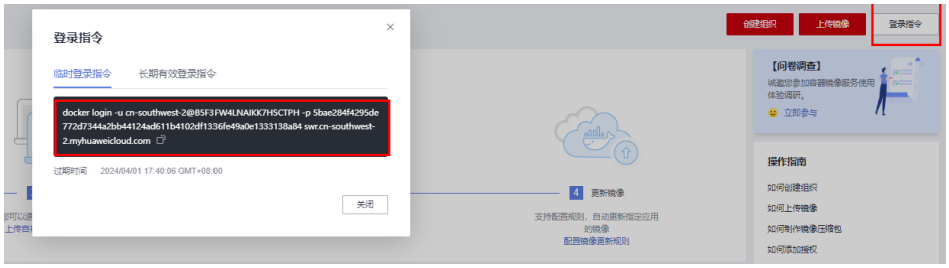
图 4-582 成功构建镜像

```
Step 12/12 : ENTRYPOINT ["/home/mind/model/run_vllm.sh"]
 ---> Running in 3183bafcdaaa
Removing intermediate container 3183bafcdaaa
 ---> 3f8a42ebda99
Successfully built 3f8a42ebda99
Successfully tagged swr.cn-nor                                                -standard
```

Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-583 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示上传镜像成功。

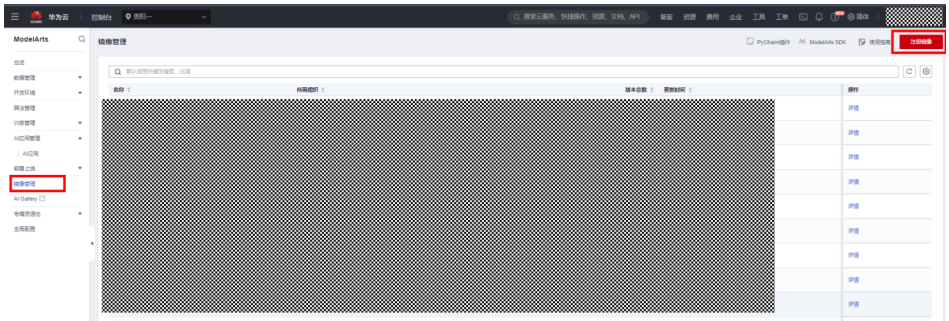
图 4-584 成功上传镜像

```
6.3.905-standard: digest: sha256:1f0b823e0fe6aa0967b                    7cf44e402d5b318b529f7 size: 12710
```

Step8 注册镜像

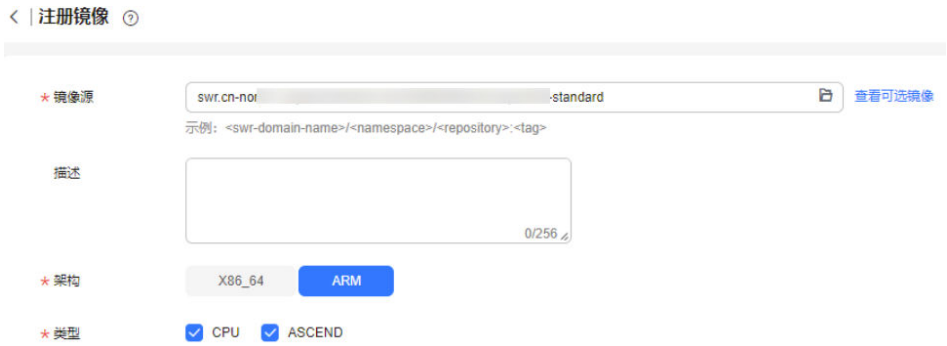
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中单击“注册镜像”。

图 4-585 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 4-586 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.33.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 4-587 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-588 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.33.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

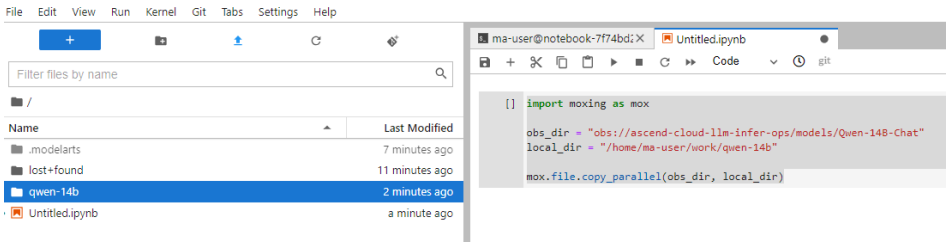
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-589 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-590 查询结果

```
+------------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                             |
+---------------------------+---------------+----------------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+====================================================+
| 4     910B2               | OK            | 91.4        50                0    / 0           |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+====================================================+
| 5     910B2               | OK            | 92.5        51                0    / 0           |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+====================================================+

+---------------------------+---------------+----------------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+====================================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+====================================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+====================================================+
```

2. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得本次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务**

  在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

  （1）非多模态

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
```

```
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

（2）llava多模态

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--image-input-type pixel_values \
--image-token-id 32000 \
--image-input-shape 1,3,336,336 \
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

多模态推理服务参数说明如下：

- VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
- VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
- PYTORCH_NPU_ALLOC_CONF=expandable_segments:False；llava多卡启动时需要关闭虚拟内存扩展；开启时可能提升模型性能。允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。
- --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
- --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

  [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。
- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

  ```
  最小处理单元为14*14
  【llava1.5】
  336*336图像 ==(336/14=24)>> 24*24=576
  672*672图像 ==(672/14=48)>> 48*48=2304
  【llava1.6】
  336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336
  ==(336/14=24)>> 336/14+2*24*24=1176
  672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336
  ==(336/14=24)>> 672/14+5*24*24=2928
  ```

- --chat-template：llava对话构建模板。

– **方式二：通过vLLM服务API接口启动服务**

在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- -model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

  如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。

- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-306**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
```

```
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。

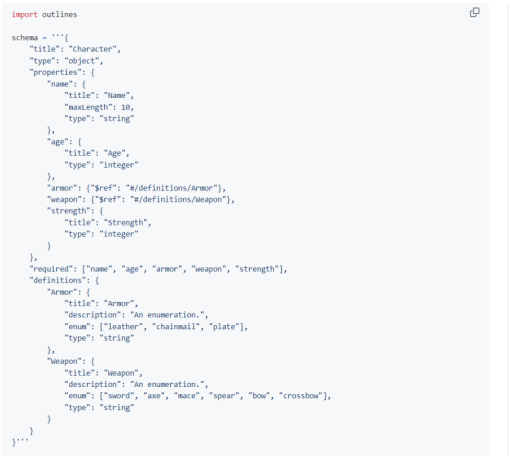
```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-306 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-591** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.33.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

  说明

  - 如果需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
  - 权重文件夹不要以"model"命名，如果以"model"命名会导致后续创建AI应用报错。
- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-592 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **方式一：通过OpenAI服务API接口启动服务**

  （1）非多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  （2）llava多模态

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--image-input-type pixel_values \
--image-token-id 32000 \
--image-input-shape 1,3,336,336 \
--image-feature-size 576 \
--chat-template examples/template_llava.jinja \
--dtype bfloat16 \
--served-model-name llava \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

  多模态推理服务参数说明如下：

  - VLLM_IMAGE_FETCH_TIMEOUT图片下载时间环境变量。
  - VLLM_ENGINE_ITERATION_TIMEOUT_S：服务间隔最大时长，超过会报timeout错误。
  - PYTORCH_NPU_ALLOC_CONF=expandable_segments:False；llava多卡启动时需要关闭虚拟内存扩展；开启时可能提升模型性能。允许分配器最初创建一个段，然后在以后需要更多内存时扩展它的大小。
  - --image-input-type：图像输入模式，pixel_values and image_features; 当前流程以pixel_values为例。具体使用方式见vllm官网。
  - --image-token-id：LLM模型图像输入占位input id，llava-1.5是32000，llava-v1.6是64000；格式如

    [1, 32000, ..., 32000, 29871, 13, 11889, 29901, 1724, 29915, 29879, 278, 2793, 310, 278, 1967, 29973, 13, 22933, 9047, 13566, 29901]，当前例子中一共576个32000，后面id则为prompt id。

- --image-input-shape：输入图片维度，当前不支持图片动态维度，如果图片不是（1，336，336）shape，将会被resize。
- --image-feature-size：图片输入解析维度大小；llava-v1.6图片输入维度与image-feature-size关系映射表见**git**；计算原理如下：

```
最小处理单元为14*14
【llava1.5】
336*336图像 ==(336/14=24)>> 24*24=576
672*672图像 ==(672/14=48)>> 48*48=2304
【llava1.6】
336*336图像 ==(1个patch+1个自身缩放+换行标记)>> 换行标记+2个336*336 ==(336/14=24)>> 336/14+2*24*24=1176
672*672图像 ==(4个patch+1个自身缩放+换行标记)>> 换行标记+5个336*336 ==(336/14=24)>> 672/14+5*24*24=2928
```

- --chat-template：llava对话构建模板。

- **方式二：通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}

python -m vllm.entrypoints.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

推理服务基础参数说明如下：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。

- --tensor-parallel-size：并行卡数。此处举例为1，表示使用单卡启动服务。
- --host：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- -max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。

- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**、**smoothquant**或者**GPTQ**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step1 准备模型文件和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。
- --served-model-name：vllm服务后台id。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得本次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 4-593 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-594 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

图 4-595 创建完成

说明

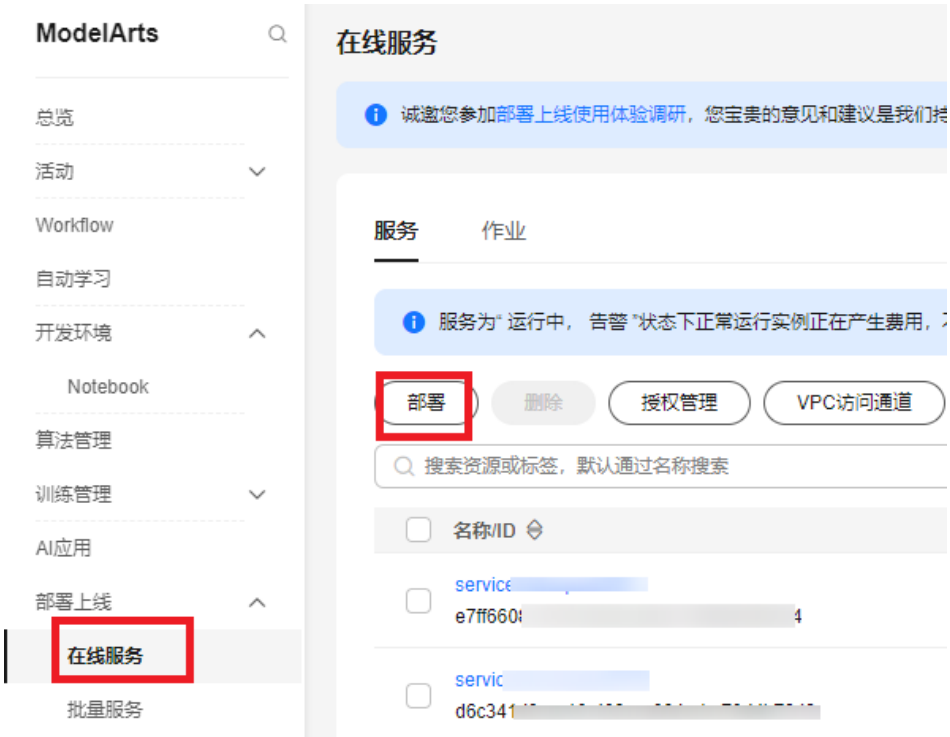
如果权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 4-596 部署在线服务

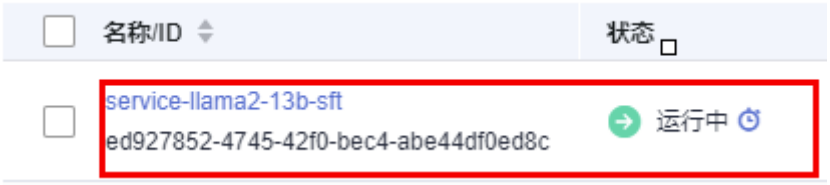


2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-597 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 4-598 服务部署完成

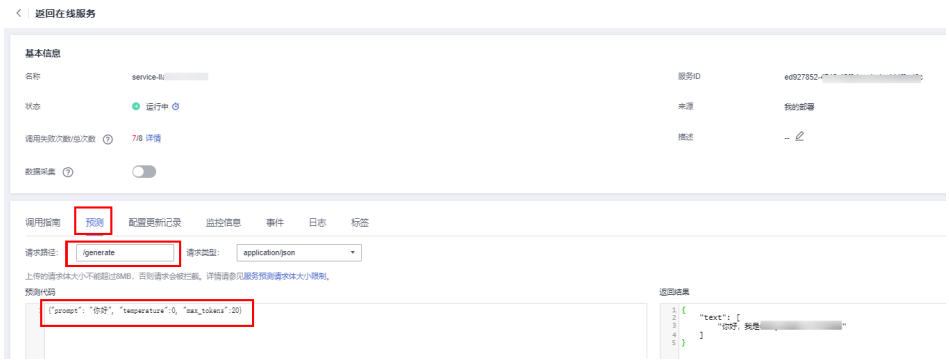


Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

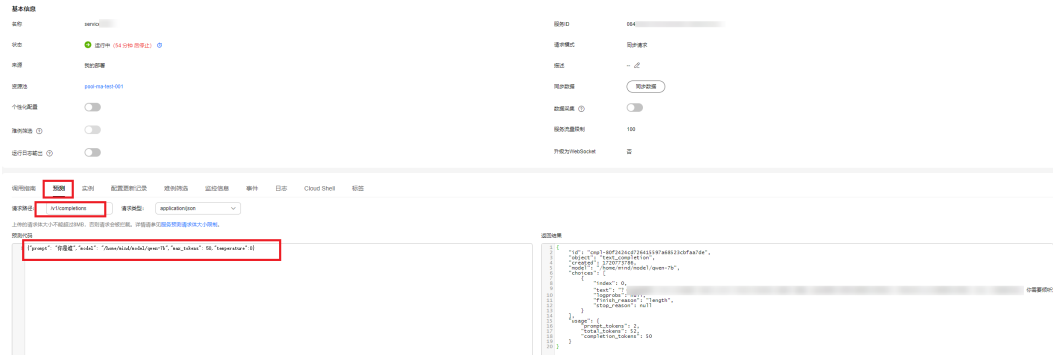
如果以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-599 预测-vllm



如果以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-600 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.33.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。目前使用的opencompass版本是0.2.6

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
bash install.sh
```

3. 在/home/ma-user/AscendCloud/AscendCloud-LLM/llm_tools/llm_evaluation/benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

4. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤5进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
```

```
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

5. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
host=$host \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- host：与起服务的host保持一致，比如起服务为0.0.0.0,host设置也为0.0.0.0。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm host=0.0.0.0 service_port=8080 max_out_len=16 batch_size=2
eval_datasets=mmlu_gen    model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

6. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

7. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

fix_id_list, 将最大值适当调低。

ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 4-307 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

8. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

```
npu：
mmlu：46.6
gpu：
mmlu：47
```

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

4.33.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。如果需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，如果静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试已经支持投机推理能力。

执行性能测试脚本前，需先安装相关依赖。

```
conda activate python-3.9.10
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

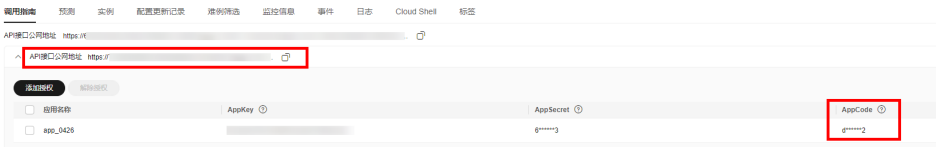
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-601 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-602 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57295766
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.72861049
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_dataset.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。

- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

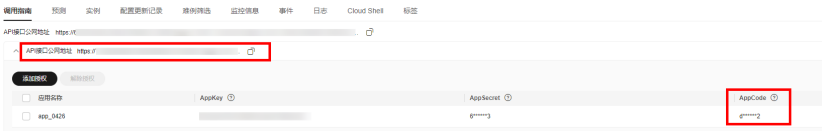
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：如果以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；如果以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-603 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。如果服务部署在Notebook中，该参数为Notebook中权重路径；如果服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
   - --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
   - --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
   - --benchmark-csv：结果保存路径，如benchmark_serving.csv。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-604 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每token平均时延（ms）	首token平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

投机推理 benchmark 验证

本章节介绍如何进行投机推理benchmark验证，当前投机推理benchmark仅支持在Notebook中进行测试。

1. 进入benchmark_tools目录下。
   ```
   cd benchmark_tools
   ```
2. 运行验证脚本speculative_benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python speculative_benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --dataset human-eval-v2-20210705.jsonl \ --tokenizer /path/to/tokenizer  --num-prompts 80 \
   --output_len 4096 --trust-remote-code
   ```
   - --backend：服务类型，如tgi，vllm，mindspore、openai。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口。
   - --dataset：数据集路径，推荐使用human-eval-v2-20210705.jsonl数据集，数据集可从https://github.com/openai/human-eval/blob/master/data/HumanEval.jsonl.gz下载压缩包解压获得。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
   - --num-prompts：某个频率下请求数，默认80。
   - --output_len：输出长度,默认是1024。
   - --trust-remote-code：是否相信远程代码。

   脚本运行完后，测试结果直接在终端输出。

4.33.7 推理模型量化

4.33.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16/W8A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-302**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化。

量化方法：W4A16 per-group/per-channel, W8A16 per-channel

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers==4.41.0 # AutoAWQ未适配transformers 4.42以上
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup --group-size 128 --w-bit 4
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --group-size：量化group size参数，指定-1时为per-channel权重量化，W4A16支持128和-1，W8A16支持-1。
- --w-bit：量化比特数，W4A16设置4，W8A16设置8。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式离线转换（可选）

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，在线转换会增加启动时间，可以提前对权重进行转换以减少启动时间，转换步骤如下：

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.33.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
   - --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
   - --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。
3. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。
   ```
   -q smoothquant 或者 --quantization smoothquant
   --dtype=float16
   ```

4.33.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

在GPU机器上使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-605 抽取 kv-cache 量化系数


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.33.7.4 使用 GPTQ 量化

当前版本使用GPTQ量化仅支持W8A16 perchannel的量化形式，使用W8A16的量化不仅可以保证精度在可接受的范围内，同时也有一定的性能收益。

GPTQ W8A16量化支持的模型请参见**表4-302**。

本章节介绍如何在GPU的机器上使用开源GPTQ量化工具**GPTQ (huggingface.co)**量化模型权重，然后在NPU的机器上实现推理量化。

具体操作如下：

1. 开始之前，请确保安装了以下库：
   ```
   pip install auto-gptq
   pip install --upgrade accelerate optimum transformers
   ```
2. 设置GPTQConfig的参数，并且创建一个数据集用于校准量化的权重，以及一个tokenizer用于准备数据集。
   ```
   from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig
   model_id = "meta-llama/CodeLlama-34b-hf"
   tokenizer = AutoTokenizer.from_pretrained(model_id)
   gptq_config = GPTQConfig(bits=8, dataset="c4", tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
3. 您也可以将自己的数据集作为字符串列表传递，但强烈建议使用GPTQ论文中的相同数据集。
   ```
   dataset = ["auto-gptq is an easy-to-use model quantization library with user-friendly apis, based on GPTQ algorithm."]
   gptq_config = GPTQConfig(bits=8, dataset=dataset, tokenizer=tokenizer, group_size=-1, damp_percent=0.01, desc_act=False, sym=True, use_exllama=False)
   ```
4. 加载要量化的模型，并将gptq_config传递给from_pretrained()方法。
   ```
   quantized_model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", quantization_config=gptq_config)
   ```
5. 您还可以使用save_pretrain()方法在本地保存您的量化模型。如果模型是用device_map参数量化的，请确保在保存之前将整个模型移动到GPU或CPU。例如，要将模型保存在CPU上。
   ```
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   tokenizer.save_pretrained("CodeLlama-34b-hf")

   # if quantized with device_map set
   quantized_model.to("cpu")
   quantized_model.save_pretrained("CodeLlama-34b-hf")
   ```

使用量化模型

使用量化模型需要在NPU的机器上运行。

1. 在模型的保存目录中创建quant_config.json文件，bits必须设置为8，指定量化为int8；group_size必须设置为-1，指定不使用pergroup；desc_act必须设置为false，内容如下：

```
{
    "bits": 8,
    "group_size": -1,
    "desc_act": false
}
```

2. 启动vLLM，如果是使用命令行的方式，指定--quantization "gptq"参数，其他参数请参考**Step3 启动推理服务**

```
python -m vllm.entrypoints.openai.api_server --model <your_model> --quantization "gptq"
```

3. 如果是sdk使用的方式，指定quantization="gptq"参数

```
llm = LLM(model="meta-llama/CodeLlama-34B-Instruct-bf16-GPTQ-w8a16", trust_remote_code=True,
          max_num_seqs=64, max_model_len=2048, block_size=128, quantization="gptq")
```

启动之后可以正常进行推理。

4.33.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-308 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	--		8	128
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32
38	falcon-11b	1	8	1	64

4.33.9 附录：Standard 大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。
- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：使用AWQ转换llama3.1系列模型权重出现报错ValueError: `rope_scaling` must be a dictionary with two fields, `type` and `factor`,

  解决方法：将transformers升级到4.44.0，修改对应transformers中的transformers/models/llama/modeling_llama.py，在class LlamaRotaryEmbedding中的forward函数中增加self.inv_freq = self.inv_freq.npu()

- 问题7：使用Qwen2-7B、Qwen2-72B模型有精度问题，重复输出感叹号

  检查【配置环境变量】章节中，高精度模式的环境变量是否开启

4.34 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.908）

4.34.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**表4-311**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。

- Server驱动版本要求23.0.6
- PyTorch版本：2.1.0
- 确保容器可以访问公网。

文档更新内容

6.3.908版本相对于6.3.907版本新增如下内容：

- 文档和代码中新增对mistral和mixtral模型的适配，并添加训练推荐配置。
- 文档中新增对Llama3支持长序列文本（ sequence_length > 32k ）训练内容，例如新增参数context-parallel-size。
- 文档中针对数据集预处理时，handler-name参数的说明，新增不同handler对训练数据的拼接和推理prompt的构造等说明。

训练支持的模型列表

本方案支持以下模型的训练，如表4-309所示。

表 4-309 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

操作流程

图 4-606 操作流程图

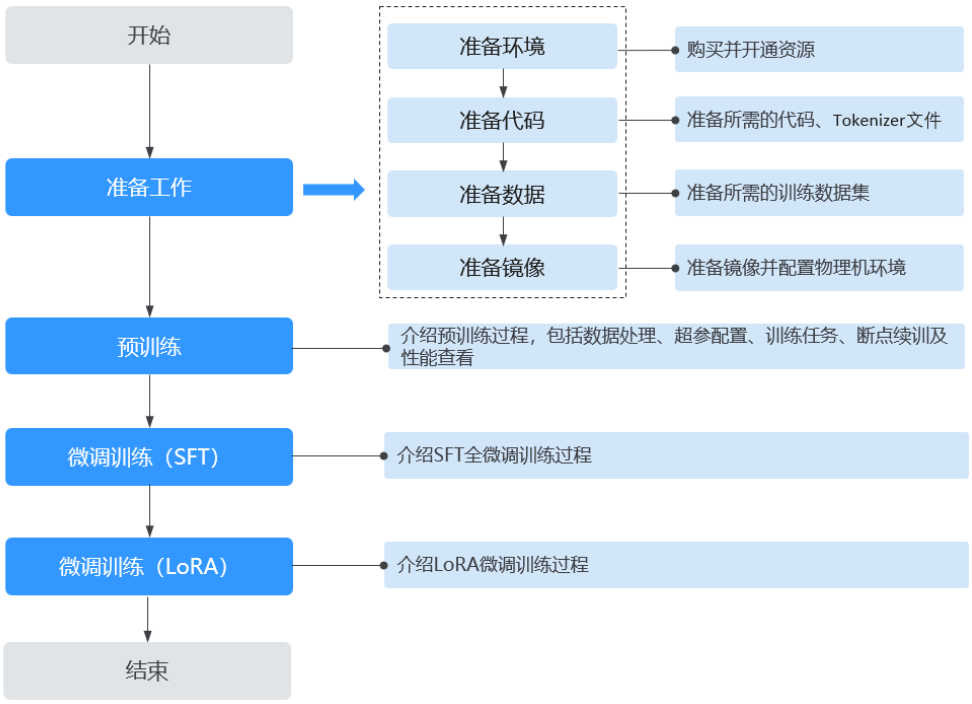


表 4-310 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.34.2 准备工作

4.34.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-319**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.34.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-311**所示，模型列表、对应的开源权重获取地址如**表4-312**所示。

表 4-311 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

获取模型权重文件

表 4-312 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2
23	mixtral	mixtral-8x7b	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
- **方法二：huggingface-cli**：huggingface-cli是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：HF-Mirror中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

  ```
  huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
  ```

  如果要下载指定版本的模型文件，则命令如下：

```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本> --local-dir <模型下载路径>
```

- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.908中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
  |——llm_train                # 模型训练代码包
     |——AscendSpeed          # 基于AscendSpeed的训练代码
        |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
        |——scripts/             # 训练需要的启动脚本
           |——llama2           # llama2系列模型执行脚本的文件夹
           |——llama3           # llama3系列模型执行脚本的文件夹
           |——qwen             # Qwen系列模型执行脚本的文件夹
           |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
           |—— ...
           |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
           |—— install.sh       # 环境部署脚本
        |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——llm_inference               # 推理代码包
  |——llm_tools                   # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
   |—— AscendSpeed             # 代码目录
      |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
      |——scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
   # 自动生成数据目录结构
   |—— processed_for_input          #目录结构会自动生成，无需用户创建
      |—— ${model_name}              # 模型名称
         |—— data              # 预处理后数据
         |—— pretrain    # 预训练加载的数据
         |—— finetune    # 微调加载的数据
      |——converted_weights  # HuggingFace格式转换megatron格式后权重文件
   |—— saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
      |—— ${model_name}              # 模型名称
         |—— logs              # 训练过程中日志（loss、吞吐性能）
            |—— saved_models
         |—— lora              # lora微调输出权重
         |—— sft               # 增量训练输出权重
         |—— pretrain          # 预训练输出权重
|—— tokenizers                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
   |—— Llama2-70B
|—— models                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
   |—— Llama2-70B
|—— training_data                #原始数据目录，需要用户手动创建，后续操作步骤中会提示
   |—— train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
   |—— alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。

2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。
```
unzip AscendCloud-*.zip
```
3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。
```
cd /home/ma-user/ws
mkdir -p tokenizers/Llama2-70B
```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.34.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。
```
{
   'id': '1',
   'url': 'https://simple.wikipedia.org/wiki/April',
   'title': 'April',
   'text': 'April is the fourth month...'
}
```
- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。

- output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

```
{
  "conversation_id": 1,
  "meta_instruction": "",
  "num_turns": 3,
  "chat": {
    "turn_1": {
      "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

  如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的 **scripts/tools/ExcelToJson.py** 工具，其转换的要求为：

  - 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
  - MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
    - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
    - Human: 数据集中每条数据的输入。
    - assistant: 数据集中每条数据的输出。
  - 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

    - user_id: 用户的唯一不重复的ID值，必选。
    - excel_addr: 待处理的excel文件的地址，必选。

- dataset_name: 处理后的数据集名称，必选。
- proportion: 测试集所占份数，范围[1,9]，可选。
- test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
- proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。
   ```
   mkdir training_data
   ```
   数据存放参考目录结构如下：
   ```
   ${workdir}（例如/home/ma-user/ws）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

说明

多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.34.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-313 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080

表 4-314 模型镜像版本

模型	版本
CANN	cann_8.0.rc3

模型	版本
驱动	23.0.6
PyTorch	2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --cpus 192 \
      --memory 1000g \
      --shm-size 200g \
      --net=host \
      -v ${work_dir}:${container_work_dir} \
   ```

```
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
```
docker exec -it ${container_name} bash
```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```
4. 使用ma-user用户安装依赖包。
```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```

注意

为了避免因使用不同版本的 transformers 库进行训练和推理而导致冲突的问题，建议用户分别为训练和推理过程创建独立的容器环境。

5. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：
```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/             # 训练需要的启动脚本
  |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/           # MindSpeed昇腾大模型加速库
  |——ModelLink/           # ModelLink端到端的大语言模型方案
      |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

如果git下载代码时报错，请参见**Git下载代码时报错**解决。

4.34.3 执行预训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-315**所示。其他超参均有默认值，可以参考**表4-318**按照实际需求修改。

表 4-315 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

请根据**Step2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx>
<NNODES=4> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
```

```
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
```

- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  ```
  示例：
  ```
  sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
  ```

  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_pretrain_7b.sh
  ```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-607 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

4.34.4 执行 SFT 全参微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-315**所示。其他超参均有默认值，可以参考**表4-318**按照实际需求修改。

表 4-316 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。

  多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:

```
# 第一台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
```

- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  ```
  示例：
  ```
  sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例：
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
  ```

  注意：如果单机运行需要指定使用NPU卡的数量，可提前定义变量 NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令：

  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_sft_7b.sh
  ```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.34.5 执行 LoRA 微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-315所示。其他超参均有默认值，可以参考表4-318按照实际需求修改。

表 4-317 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-319**所示。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
  示例:
  ```
  #第一台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
  # 第二台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
  # 第三台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
  # 第四台节点
  sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  示例:
  ```
  # 第一台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
  # 第二台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
  # 第三台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
  # 第四台节点
  MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
  ```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

启动训练脚本可使用以下两种启动命令，二选一即可，其中区别如下：

- **传递参数形式：将主节点IP地址、节点个数、节点RANK**的参数传递至运行的脚本中执行。
  ```
  # 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
  sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
  ```
- **定义变量形式**：提前定义**主节点IP地址、节点个数、节点RANK**的环境变量并赋值，再执行脚本。
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
  ```
  如果单机运行需要指定使用NPU卡的数量，可提前定义变量 NPUS_PER_NODE 。例如使用单机四卡训练Llama2-7B命令。
  ```
  MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 NPUS_PER_NODE=4 sh scripts/llama2/0_pl_lora_7b.sh
  ```

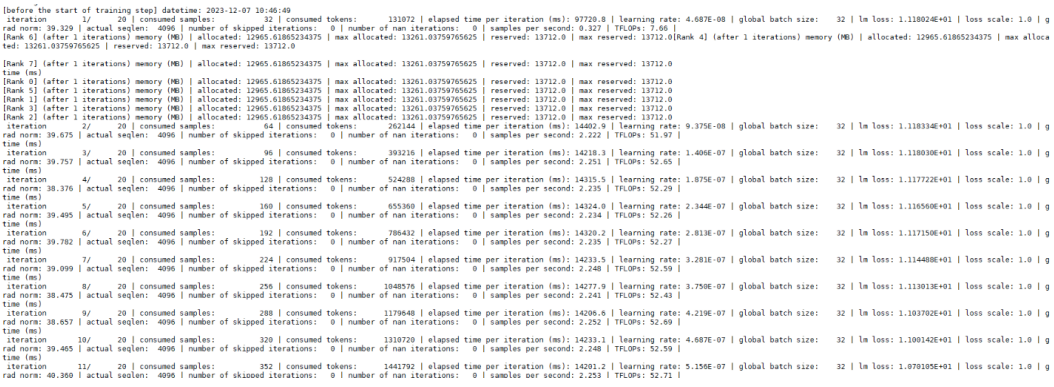
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

4.34.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-608 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

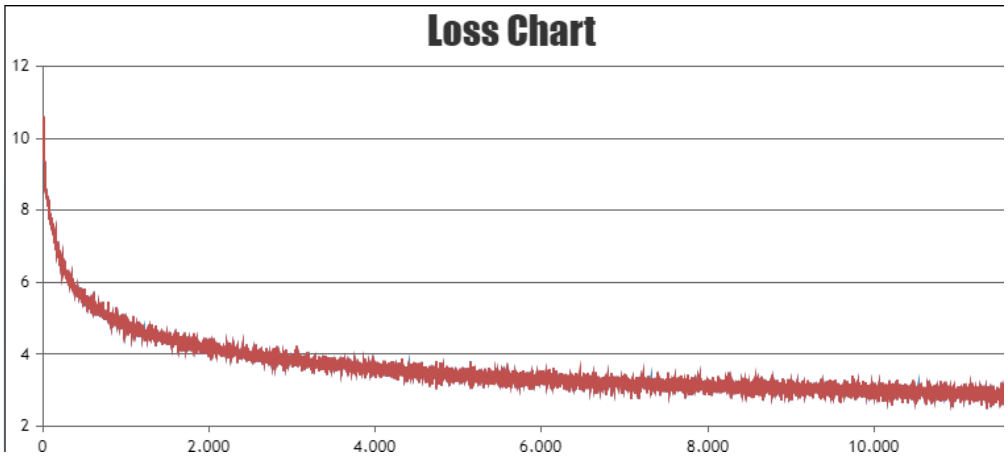
- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-318**。

- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具TrainingLogParser查看loss收敛情况，如图4-609所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 4-609 Loss 收敛情况（示意图）

4.34.7 训练脚本说明参考

4.34.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以下参数取值主要以**llama2-70b预训练**为例，请根据实际模型修改。

表 4-318 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。请根据实际修改。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	4	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。

参数	示例值	参数说明
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-319所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-319 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend

4.34.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：moss-003-sft-data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

  ```
  [
    {"text": "document"},
    {"other keys": "optional content"}
  ]
  ```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

  ```
  doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
  ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

  ```
  def _filter(self, sample):
      messages = self._format_msg(sample)
      full_prompt = self.prompter.generate_training_prompt(messages)
      tokenized_full_prompt = self._tokenize(full_prompt)
      if self.args.append_eod:
          tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
          tokenized_full_prompt["attention_mask"].append(1)
          tokenized_full_prompt["labels"].append(self.tokenizer.eod)
      if not self.train_on_inputs:
          user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
              self.prompter.template.assistant_token + "\n"
          tokenized_user_prompt = self._tokenize(user_prompt)
          user_prompt_len = len(tokenized_user_prompt["input_ids"])
          tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
      for key in self.args.json_keys:
          tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
      return tokenized_full_prompt
  ```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

    ```
    [
    {
        "instruction": "指令（必填）",
    ```

```
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) + self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

     iii. input_ids是user_ids和assistant_ids的拼接

iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
user_token = [195]
assistant_token = [196]
input_ids = user_token + {Human} + assistant_token + {MOSS}
```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context.
\n Write a response that appropriately completes the request. \n Please note that you need to
```



```
think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-320 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。

环境变量	示例	参数说明
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.34.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-321 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.34.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-610所示。

图 4-610 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-611所示。

图 4-611 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-612 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-613所示。

图 4-613 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-614 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-615所示。

图 4-615 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.34.8 常见错误原因和解决方法

4.34.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-319**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

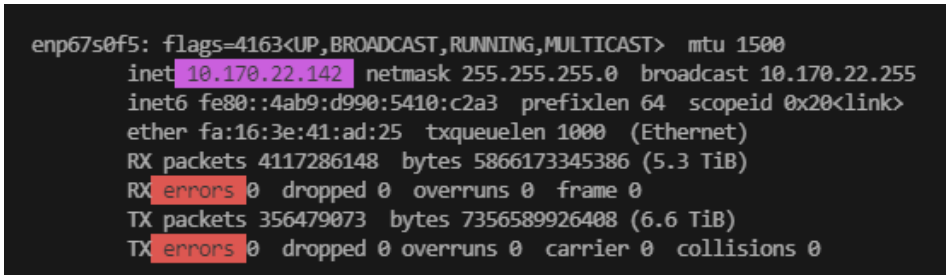
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.34.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-616 网卡名称错误



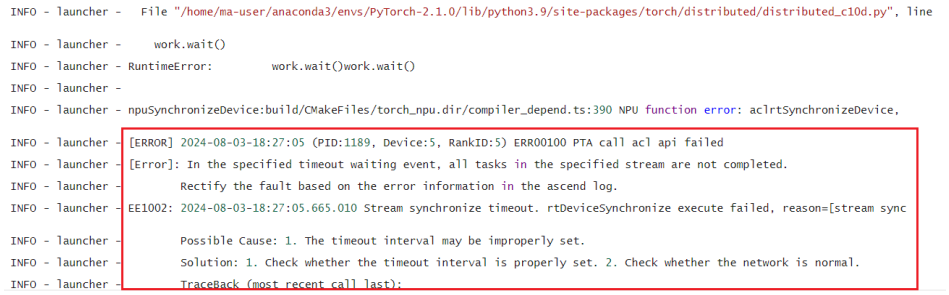

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.34.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-617 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.34.8.4 Git 下载代码时报错

在执行scripts/install.sh安装命令或使用Dockerfile构建镜像时，如遇到git下载代码出现以下类似的报错信息，关闭git验证即可。

报错信息：

```
fatal: unable to access 'https://gitee.com/ascend/ModelLink.git/': error setting certificate verify locations:
CAfile: /etc/pki/tls/certs/ca-bundle.crt CApath: none
```

关闭git验证命令如下：

```
git config --global http.sslverify false
```

4.35 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.908）

4.35.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的微调方案，包括SFT全参微调、LoRA微调、DPO训练方案。

- DPO(Direct Preference Optimization)：直接偏好优化方法，通过直接优化语言模型来实现对大模型输出的精确把控，不用进行强化学习，也可以准确判断和学习到使用者的偏好，最后，DPO算法还可以与其他优化算法相结合，进一步提高深度学习模型的性能。
- SFT监督式微调(Self-training Fine-tuning)：是一种利用有标签数据进行模型训练的方法。 它基于一个预先训练好的模型，通过调整模型的参数，使其能够更好地拟合特定任务的数据分布。 与从头开始训练模型相比，监督式微调能够充分利用预训练模型的知识和特征表示，从而加速训练过程并提高模型的性能。
- LoRA微调LoRA(Low-Rank Adaptation)：微调是一种用于调整大型预训练模型的高效微调技术。 这种方法主要针对如何在保持模型大部分参数固定的同时，通过引入少量可训练参数来调整模型以适应特定任务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**表4-324**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.RC3。
- Server驱动版本要求23.0.6
- PyTorch版本：2.2.0
- Python版本：3.10
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如**表4-322**所示。

表 4-322 支持的模型列表及权重文件地址

支持模型	Template	支持模型参数量	权重文件获取地址
Llama2	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-chat-hf**
Llama3	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
Llama3.1	llama3	llama3.1-8b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct/tree/main**
		llama3.1-70b	**https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct/tree/main**
Qwen1.5	qwen	qwen1.5-0.5b	**https://huggingface.co/Qwen/Qwen1.5-0.5B**
		qwen1.5-1.8b	**https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat**
		qwen1.5-4b	**https://huggingface.co/Qwen/Qwen1.5-4B**
		qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
Yi	yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
Qwen2	qwen	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**

支持模型	Template	支持模型参数量	权重文件获取地址
		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
Falcon2	falcon	falcon-11B	https://huggingface.co/tiiuae/falcon-11B
GLM-4	glm4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce

表 4-323 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
微调训练	指令监督微调训练	介绍如何进行SFT全参微调/lora微调、训练任务、性能查看。

4.35.2 准备工作

4.35.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见表4-329。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。

- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.35.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-324**所示，模型列表、对应的开源权重获取地址如**表4-322**所示。

表 4-324 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.908中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                    # 模型训练代码包
      |——LLaMAFactory                 # 基于LLaMAFactory的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——demo.yaml            # 样例yaml配置文件
         |——demo.sh              # 指令微调启动shell脚本
         |——intall.sh            # 需要的依赖包
         |——LLaMA-Factory        # LLaMAFactory的代码目录
      |——AscendSpeed               # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── LLaMAFactory               # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──demo.sh              # 指令微调启动shell脚本
        |──demo.yaml            # 样例yaml配置文件
        |──intall.sh            # 需要的依赖包
        |──LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考Step3 启动容器镜像
            |──data             # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|── tokenizers                   #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
    |── Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表4-327表格中output_dir参数值
|── saved_dir_for_output_lf          # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |── ${model_name}               # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen2-72B
   ```

4.35.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-325 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_2_ascend:pytorch_2.2.0-cann_8.0.rc3-py_3.10-hce_2.0.2312-aarch64-snt9b-20240829092203-4ccf328

表 4-326 模型镜像版本

模型	版本
CANN	cann_8.0.RC3
驱动	23.0.6
PyTorch	2.2.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装NPU设备和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --cpus 192 \
   ```

```
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如： chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

4.35.2.4 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前支持alpaca格式和sharegpt格式的微调数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加数据集描述；具体示例如下。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。

```
cd  /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
```

2. 将自定义原始数据（样例数据集：alpaca_gpt4_data.json.json）按照下面的数据存放目录要求放置。

   说明

   样例数据集alpaca_gpt4_data.json.json的下载链接：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**

   数据存放参考目录结构如下：

```
${workdir}（例如/home/ma-user/ws/llm_train ）
 |── LLaMAFactory/data
     |── alpaca_en_demo.json                # 代码原有数据集
     |── identity.json                    # 代码原有数据集
     ...
     |── alpaca_gpt4_data.json              # 自定义数据集
```

3. 更新代码目录下data/dataset_info.json文件。如使用以下示例数据集则命令如下。关于数据集文件格式及配置，更多信息请参考**data/README_zh.md** 的内容。

```
vim dataset_info.json
```

   新加配置参数如下：

```
"alpaca_gpt4_data": {
   "file_name": "alpaca_gpt4_data.json"
 },
```

   样例截图：

```
{
  "identity": {
    "file_name": "identity.json"
  },
  "alpaca_en_demo": {
    "file_name": "alpaca_en_demo.json"
  },
  "alpaca_zh_demo": {
    "file_name": "alpaca_zh_demo.json"
  },
  "alpaca_gpt4_data": {
    "file_name": "alpaca_gpt4_data.json"
  },
```

4.35.3 执行微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

Step2 修改训练 yaml 文件配置

LlamaFactroy配置文件为Yaml文件，启动训练前需修改Yaml配置文件，Yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示。

1. 选择训练策略类型。

- sft，复制**sft_yaml样例模板**内容覆盖demo.yaml文件内容。
- lora，复制**lora_yaml样例模板**内容覆盖demo.yaml文件内容。
- dpo，复制**dpo_yaml样例模板**内容覆盖demo.yaml文件内容。

2. 修改**yaml文件**(demo.yaml)的参数如**表4-327**所示。

表 4-327 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-322**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	**可修改**。指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配。取值可参考**表4-329**中**梯度累积值**列。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配
dataset	identity,alpaca_en_demo	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

3. 是否选择加速深度学习训练框架Deepspeed，可参考**表4-329**选择不同的框架。
   - 是，选用ZeRO (Zero Redundancy Optimizer)优化器。
     - ZeRO-0，配置以下参数
       ```
       deepspeed: examples/deepspeed/ds_z0_config.json
       ```

- ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
  ```
  deepspeed: examples/deepspeed/ds_z1_config.json
  ```
- ZeRO-2，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z2_config.json
  ```
- ZeRO-3，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z3_config.json
  ```
- ZeRO-3-Offload，配置以下参数
  ```
  deepspeed: examples/deepspeed/ds_z3_offload_config.json
  ```

– 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。

4. 是否开启NPU FlashAttention融合算子，具体约束详见**NPU_Flash_Attn融合算子约束**
   – 是，配置以下参数。
     ```
     flash_attn: sdpa
     ```
   – 否，注释掉flash_attn: sdpa参数
5. 是否使用固定句长。
   – 是，配置以下参数
     ```
     packing: true
     ```
   – 否，默认使用动态句长，**注释**掉packing参数。
6. 选用数据精度格式，以下参数二选一。
   – bf16，配置以下参数
     ```
     bf16: true
     ```
   – fp16，配置以下参数
     ```
     fp16: true
     ```
7. 是否使用自定义数据集。
   – 是，参考**准备数据（可选）**后，配置以下参数：参考**表4-327**dataset_dir和dataset参数说明；如alpaca_gpt4_data.json数据集前缀则为alpaca_gpt4_data。
     ```
     dataset: alpaca_gpt4_data
     dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
     ```
   – 否，使用代码包自带数据集，注释掉dataset_dir参数，配置参数如下：
     - sft或lora
       ```
       dataset: identity,alpaca_en_demo
       ```
     - dpo
       ```
       dataset: dpo_en_demo
       ```
8. 是否使用chatglm4-9b、falcon-11b模型。
   – 是，更新配置或命令。
     - chatglm4-9b，更新transformers为4.41.2版本。
       ```
       pip install transformers==4.41.2
       ```
     - falcon-11b，参考**falcon-11B模型**替换文件。
   – 否，忽略此步骤，执行下一步。
9. 如需其他配置参数，可参考**表4-328**按照实际需求修改。

Step3 启动训练脚本

修改完yaml配置文件后，启动训练脚本。模型不同最少NPU卡数不同，NPU卡数建议值可参考**表4-329**。

1. **修改启动脚本demo.sh**

   进入代码目录{work_dir}/llm_train/LLaMAFactory下**修改启动脚本**，其中{work_dir}为容器挂载路径；修改demo.sh最后一行代码：

   将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，例如将以下命令：

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
   ```

   修改为：

   ```
   FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
   ```

2. **执行多机启动命令**（可选）

   多台机器执行训练启动命令如下。

   ```
   多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
   示例:
   #第一台节点
   sh demo.sh  xx.xx.xx.xx  4  0
   # 第二台节点
   sh demo.sh  xx.xx.xx.xx  4  1
   # 第三台节点
   sh demo.sh  xx.xx.xx.xx  4  2
   # 第四台节点
   sh demo.sh  xx.xx.xx.xx  4  3
   ```

   以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

3. **执行单机启动命令**（可选）

   一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

   进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

   ```
   # 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
   sh demo.sh localhost 1 0
   ```

   单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
   ```

   其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务。

训练成功标志

“******* train metrics *******”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：微调训练常见问题**解决。

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：微调训练常见问题**问题3小节。

3、大模型参数如（qwen2-72B、llama2-70B）等sft训练完成后多线程退出时报“torch.distributed.DistStoreError: Socket Timeout”时请参考**问题4：Error waiting on exit barrier错误**

4、需要开启profiling功能进行性能数据采集和解析请参考**录制Profiling**

5、训练过程中报"ModuleNotFoundError: No module named 'multipart'"关键字异常，可更新python-multipart为0.0.12版本，具体请参考**问题5：No module named 'multipart'"报错:**

4.35.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-618 打印训练日志

```
  0%|              | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|▏             | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▎             | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍             | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌             | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋             | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊             | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉             | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█             | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏            | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▍            | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▌            | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▋            | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|█▊            | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表4-327**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

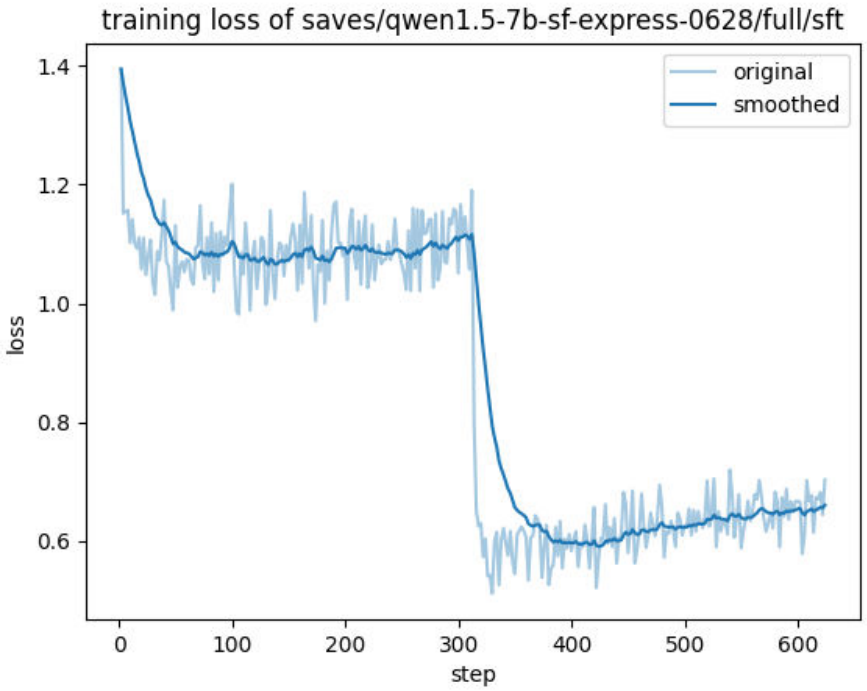
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：可通过**表4-327**表格中**output_dir**参数值路径下的**train_results.json**查看性能。吞吐计算公式为"num_input_tokens_seen / train_runtime / 训练卡数"。相关参数可查看**表4-327**。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-327**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-619**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

图 4-619 Loss 收敛情况（示意图）



4.35.5 训练脚本说明

4.35.5.1 Yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-328 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。

参数	示例值	参数说明
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示当前的训练阶段。可选择值：[pt、sft、rm、ppo、dpo]，pt代表预训练，sft代表指令监督微调，rm代表奖励模型训练，ppo代表PPO训练，dpo代表DPO训练。
finetuning_type	full	用于指定微调策略类型，可选择值【full、lora】如果设置为"full"，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
dataset	identity,alpaca_en_demo	【**可选**】注册在dataset_info.json文件数据集名称。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件，并将数据集存放于dataset_info.json同目录下。
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】dataset_info.json配置文件所属的**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表4-322**中的**template列**
max_samples	50000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。
per_device_train_batch_size	1	指定每个设备的训练批次大小。

参数	示例值	参数说明
gradient_accumulation_steps	8	**必须修改**，指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可参考**表4-329**
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
save_steps	5000	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	2.0e-5	指定学习率
disable_gradient_checkpointing	true	关闭重计算，用于禁用梯度检查点，**默认开启**梯度检查点;在深度学习模型训练中用于保存模型的状态，以便在需要时恢复。这种技术可以帮助减少内存使用，特别是在训练大型模型时，但同时影响性能。True表示关闭重计算功能。
include_tokens_per_second include_num_input_tokens_seen	true	用于在训练过程中包含每秒处理的tokens和已经看到的输入tokens，方便计算性能。

sft_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: full
```

```
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 100000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

lora_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: lora
lora_target: all
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 2.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

dpo_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: dpo
do_train: true
finetuning_type: lora
lora_target: all
pref_beta: 0.1
pref_loss: sigmoid
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: dpo_en_demo
dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
template: qwen
cutoff_len: 4096
packing: true
max_samples: 50000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/dpo
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 5.0e-6
num_train_epochs: 3.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
bf16: true
flash_attn: sdpa
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
```

```
  }
}
```

4.35.5.2 模型 NPU 卡数、梯度累积值取值表

不同模型推荐的训练参数和计算规格要求如表4-329所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推。

表 4-329 NPU 卡数、加速框架、梯度配置取值表

模型	模型参数量	训练类型	序列长度cutoff_len	梯度累积值	优化工具(Deepspeed)	规格与节点数
llama2	7B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		sft		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
	13B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
		sft		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
	70B	lora/dpo	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			8192	gradient_accumulation_steps: 8	ZeRO-3 - Offload	2*节点 & 8*Ascend
		sft	4096/8192	gradient_accumulation_steps: 4	ZeRO-3 - Offload	4*节点 & 8*Ascend
llama3	70B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		sft		gradient_accumulation_steps: 4	ZeRO-3 - Offload	4*节点 & 8*Ascend
	8B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
		sft		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
llama3.1	8B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

模型	模型参数量	训练类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
	70B	lora/dpo	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
		sft	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
Qwen2	72B	lora/dpo	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
			8192	gradient_accumulation_steps: 8	ZeRO-3-Offload	2*节点 & 8*Ascend
		sft	4096/8192	gradient_accumulation_steps: 4	ZeRO-3-Offload	4*节点 & 8*Ascend
	7B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
	0.5/1.5B	lora/sft/dpo	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
Qwen1.5	0.5/1.8B	lora/sft/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-0	1*节点 & 1*Ascend
	4B	lora/dpo	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		sft	*4096/8192*	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
	7B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
	14B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend

模型	模型参数量	训练类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
	32B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend
		sft	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		sft	8192	gradient_accumulation_steps: 4	ZeRO-3 - Offload	2*节点 & 8*Ascend
	72B	lora/dpo	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		lora	8192	gradient_accumulation_steps: 8	ZeRO-3 - Offload	2*节点 & 8*Ascend
		sft	4096/8192	gradient_accumulation_steps: 4	ZeRO-3 - Offload	4*节点 & 8*Ascend
falcon2	11B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend
GLM4	9B	lora	4096/8192	gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 8*Ascend
Yi	6B	lora/dpo	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 1*Ascend
		sft	4096/8192	gradient_accumulation_steps: 8	ZeRO-1	1*节点 & 4*Ascend
	34B	sft	4096	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		lora/dpo		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 2*Ascend
		sft	8192	gradient_accumulation_steps: 8	ZeRO-3	2*节点 & 8*Ascend
		lora/dpo		gradient_accumulation_steps: 8	ZeRO-3	1*节点 & 4*Ascend

说明

以上参数为未开启NPU FlashAttention融合算子，上述参数值仅供参考，请根据自己实际要求合理配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器、NPU节点数及其他配置。

具体优化工具使用说明可参考**如何选择最佳性能的zero-stage和-offloads**。

4.35.5.3 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

4.35.5.4 NPU_Flash_Attn 融合算子约束

1. query、key、value都需要梯度。默认开启重计算，则前向时qkv没有梯度，如果需要关闭重计算，可以在yaml配置 `disable_gradient_checkpointing: true` 关闭，但显存占用会直线上升。
2. attn_mask 只支持布尔（bool）数据类型，或者为None。
3. query的shape仅支持 [B, N1, S1, D]，其中N1≤ 2048，D≤ 512并且dim== 4。
4. 对于GQA，key的shape是 [B, N2, S2, D]，其中 N2 ≤ 2048，并且N1是N2的正整数倍。

不满足以上场景，则不能实现NPU_Flash_Attn功能。

4.35.5.5 录制 Profiling

Ascend PyTorch Profiler是针对PyTorch框架开发的性能数据采集和解析工具，通过在PyTorch训练脚本中插入Ascend PyTorch Profiler接口，执行训练的同时采集性能数据，完成训练后直接输出可视化的性能数据文件，提升了性能分析效率。

Ascend PyTorch Profiler接口可全面采集PyTorch训练场景下的性能数据，主要包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等，可以全方位分析PyTorch训练时的性能状态。

录制命令如下：

在启动训练脚本基础上**Step3 启动训练脚本** 新加DO_PROFILER=1和PROF_SAVE_PATH=/save_path参数，单机启动举例说明：

```
DO_PROFILER=1 PROF_SAVE_PATH=/save_path sh demo.sh localhost 1 0
```

- PROF_SAVE_PATH：Profiling录制结果存放路径

- DO_PROFILER：是否开启Profiling录制功能

4.35.6 附录：微调训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

1. 容器内执行以下命令，指定NPU内存分配策略的环境变量，开启动态内存分配，即在需要时动态分配内存，可以提高内存利用率，减少OOM错误的发生。
   ```
   export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True
   ```
2. 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。
3. 替换深度学习训练加速的工具或增加zero等级，可参考**模型NPU卡数、梯度累积值取值表**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。
   a. - ZeRO-0 数据分布到不同的NPU
   b. - ZeRO-1 Optimizer States分布到不同的NPU
   c. - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
   d. - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU
4. 增加卡数重新训练，未解决找相关人员定位。

问题 2：访问容器目录时提示 Permission denied

解决方法：

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: XXX not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/dynamic_module_utils.py
```

问题 4：Error waiting on exit barrier 错误

错误截图：

```
[ERROR] Error waiting on exit barrier. Elapsed: 300.1057639122009 seconds
[ERROR] Traceback (most recent call last):
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py

[ERROR]     store_util.barrier(
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     synchronize(store, data, rank, world_size, key_prefix, barrier_timeout)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li

[ERROR]     agent_data = get_all(store, rank, key_prefix, world_size)
[ERROR]   File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/utils/store.py", li
[ERROR]     store_get(f"{prefix}{node_rank}_FIN")
[ERROR] torch.distributed.DistStoreError: Socket Timeout
```

报错原因：多线程退出各个节点间超时时间默认为300s，时间设置过短。

解决措施：

修改容器内torch/distributed/elastic/agent/server/api.py文件参数：

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/torch/distributed/elastic/agent/server/api.py
```

修改def _exit_barrier(self)方法中的barrier_timeout参数，修改后如图4-620所示。

```
#修改前
barrier_timeout=self._exit_barrier_timeout
#修改后
barrier_timeout=3000
```

图 4-620 修改后的 barrier_timeout 参数

```
913     def _exit_barrier(self):
914         """
915         Define a barrier that keeps the agent process alive until all workers finish.
916
917         Wait for ``exit_barrier_timeout`` seconds for all agents to finish
918         executing their local workers (either successfully or not). This
919         acts as a safety guard against user scripts that terminate at different
920         times.
921         """
922         log.info(
923             "Local worker group finished (%s). "
924             "Waiting %s seconds for other agents to finish",
925             self._worker_group.state, self._exit_barrier_timeout
926         )
927         start = time.time()
928         try:
929             store_util.barrier(
930                 self._store,
931                 self._worker_group.group_rank,
932                 self._worker_group.group_world_size,
933                 key_prefix= TERMINAL_STATE_SYNC_ID,
934                 barrier_timeout=3000,
935             )
936             log.info(
937                 "Done waiting for other agents. Elapsed: %s seconds", time.time() - start
938             )
939         except SignalException as e:
940             log.warning("Got termination signal: %s", e.sigval)
941             raise
942         except Exception:
943             log.exception(
944                 "Error waiting on exit barrier. Elapsed: %s seconds",
945                 time.time() - start
946             )
```

问题 5：训练过程中报"ModuleNotFoundError: No module named 'multipart'" 报错：

截图如下：

```
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio.components.base import FormComponent
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio.components.annotated_image import Annota
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio.components.base import Component
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio.blocks import Block, BlockContext
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio import (
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from gradio.routes import App  # HACK: to avoid circ
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/p
    from multipart.multipart import parse_options_header
ModuleNotFoundError: No module named 'multipart'
```

解决措施：可更新python-multipart为0.0.12版本，具体步骤如下：

- 启动训练任务前更新python-multipart版本：

```
pip install python-multipart==0.0.12
```

4.36 主流开源大模型基于 Standard+OBS 适配 ModelLink PyTorch NPU 训练指导（6.3.908）

4.36.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表4-330所示。

表 4-330 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)

序号	支持模型	支持模型参数量	权重文件获取地址
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**

操作流程

图 4-621 操作流程图

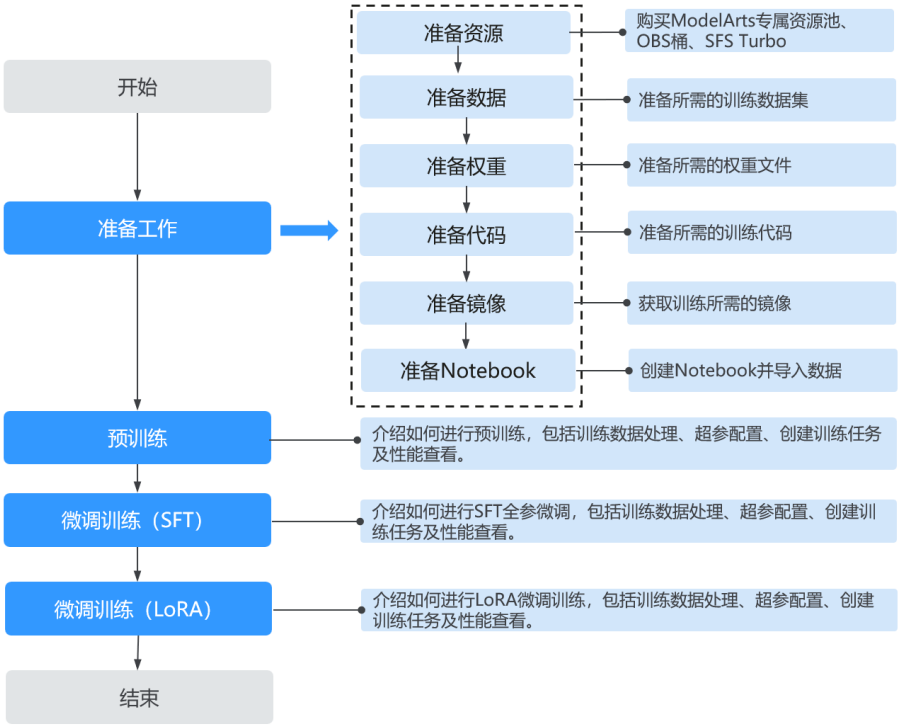


表 4-331 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。

阶段	任务	说明
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.36.2 准备工作

4.36.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-337**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.36.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
   "conversation_id": 1,
   "meta_instruction": "",
   "num_turns": 3,
   "chat": {
  ```

```
  "turn_1": {
    "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？ <eoh>\n",
    "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
    "Commands": "<|Commands|>: None<eoc>\n",
    "Tool Responses": "<|Results|>: None<eor>\n",
    "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
  },
  "turn_2": { ... },
  "turn_3": { ... },
 "category": "Brainstorming"
}
```


如果用户希望将MOSS数据集的Excel格式转换为.json 格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

  ```
  1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
  2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
  (随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
  3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
  (随机选择 3个数据作为测试集)
  ```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.36.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-330**。

   权重文件下载有如下几种方式，但不仅限于以下方式：

   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：

     ```
     huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
     ```

     如果要下载指定版本的模型文件，则命令如下：

     ```
     huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
     ```

   - **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
   - **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.36.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-332**所示。

表 4-332 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.908代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
                |——llama2           # llama2系列模型执行脚本的文件夹
                |——llama3           # llama3系列模型执行脚本的文件夹
                |——qwen             # Qwen系列模型执行脚本的文件夹
                |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |—— ...
                |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |—— install.sh       # 环境部署脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

修改代码

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后。在上传代码前，需要对解压后的训练脚本代码进行修改。具体文件为：llm_train/AscendSpeed/scripts/obs_pipeline.sh，具体修改代码内容以及位置，如下所示。

1. 训练作业中存在2个代码目录，一个是从OBS上传到ModelArts Standard训练容器中的代码目录**OBS_CODE_DIR**，一个是后续构建新镜像步骤**ECS中构建新镜像**中镜像的代码目录**CODE_DIR**。修改代码如**图4-622**。

   图 4-622 修改区分训练作业中 2 个代码目录

   ```
   OBS_CODE_DIR=`pwd`
   CODE_DIR=/home/ma-user/AscendSpeed

   ModelLink_PATH=$CODE_DIR/ModelLink
   MindSpeed_PATH=$CODE_DIR/MindSpeed
   Scripts_PATH=$OBS_CODE_DIR/scripts
   ASCENDCLOUD_PATCH_PATH=$CODE_DIR/ascendcloud_patch
   INPUT_PROCESSED_DIR=${INPUT_PROCESSED_DIR:=$ROOT_DIR/processed_for_input/$MODEL_NAME}
   OUTPUT_SAVE_DIR=${OUTPUT_SAVE_DIR:=$ROOT_DIR/saved_dir_for_output/${MODEL_NAME}_${RUN_TYPE}_${SEQ_LEN}}
   ```

2. 使用环境变量**SAVE_PATH**重新覆盖权重文件保存路径，作为最终的权重保存路径。修改代码如**图4-623**。

图 4-623 修改权重保存路径

```
# PTD args
cd $ASCENDCLOUD_PATCH_PATH
MODEL_TYPE=$MODEL_NAME
SAVE_PATH=$OUTPUT_SAVE_DIR
MODEL_PATH=${MODEL_PATH:-$CONVERT_MODEL_PATH}
CKPT_SAVE_PATH=${SAVE_PATH}/saved_models/${RUN_TYPE}
mkdir -p ${SAVE_PATH}/logs
mkdir -p ${SAVE_PATH}/saved_models/${RUN_TYPE}

SAVE_PATH=${SAVE_PATH}/saved_models/${RUN_TYPE}

if [[ $Instruction_Data =~ ${DATA_TYPE} ]]; then
    run_type_args="
        --is-instruction-dataset \
    "
fi
```

3. 多机训练场景下，需要将**CODE_DIR**修改为**OBS_CODE_DIR**目录，则可以使用scripts/tools/sync_with_obs.py工具将其它节点的权重文件同步上传到主节点。修改代码如**图4-624**。

图 4-624 多机同步权重文件

```
if [[ $MOUNT == "OBS" ]]; then
  shard=$((TP * PP))
  if [ "$shard" -gt 8 ]; then
    cd $OBS_CODE_DIR
    python scripts/tools/sync_with_obs.py --local_dir=$CKPT_SAVE_PATH
    CKPT_SAVE_PATH=${MA_JOB_DIR}/mg_weights
  else
    echo "skip sync with obs"
  fi
fi
sleep 5s
```

代码上传至 OBS

将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed             # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── models                                         #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
|── tokenizers                                     #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-13b-hf
```

4.36.2.5 准备镜像

4.36.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-333 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行install.sh文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

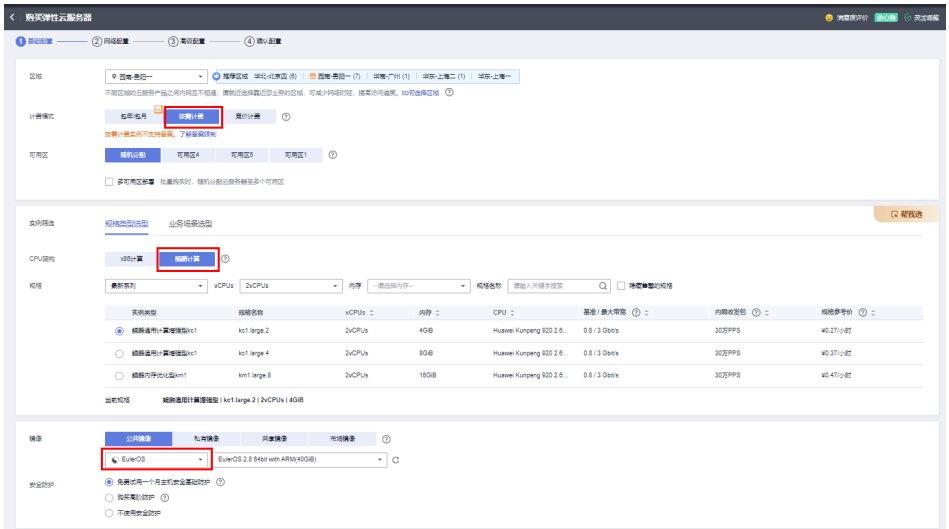
4.36.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

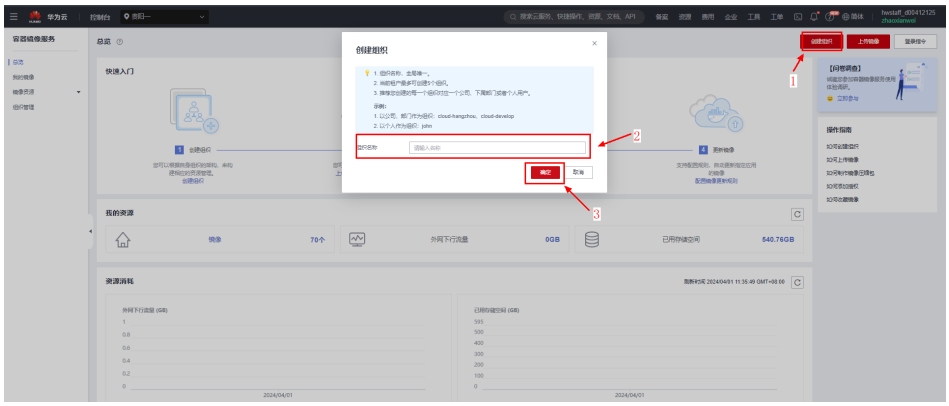
图 4-625 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-626 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

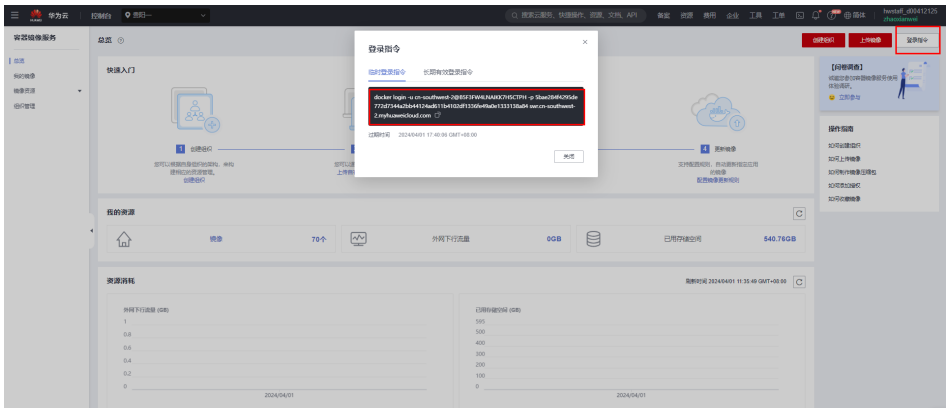
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-333**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-627 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.36.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-628**中都需要执行install.sh文件，来安装依赖以及下载完整代码。命令如下：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-628 训练作业启动命令

4.36.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

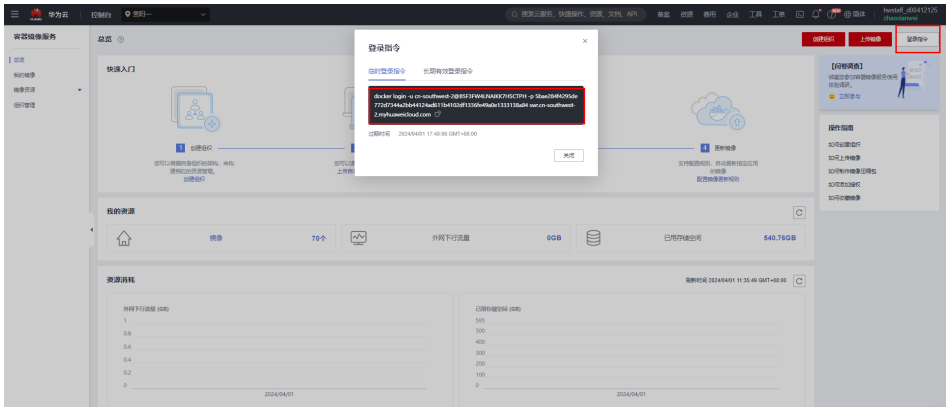
获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-332**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.908-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-629 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.36.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中，如果用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-630 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，如果需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-631 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

4.36.3 预训练

前提条件

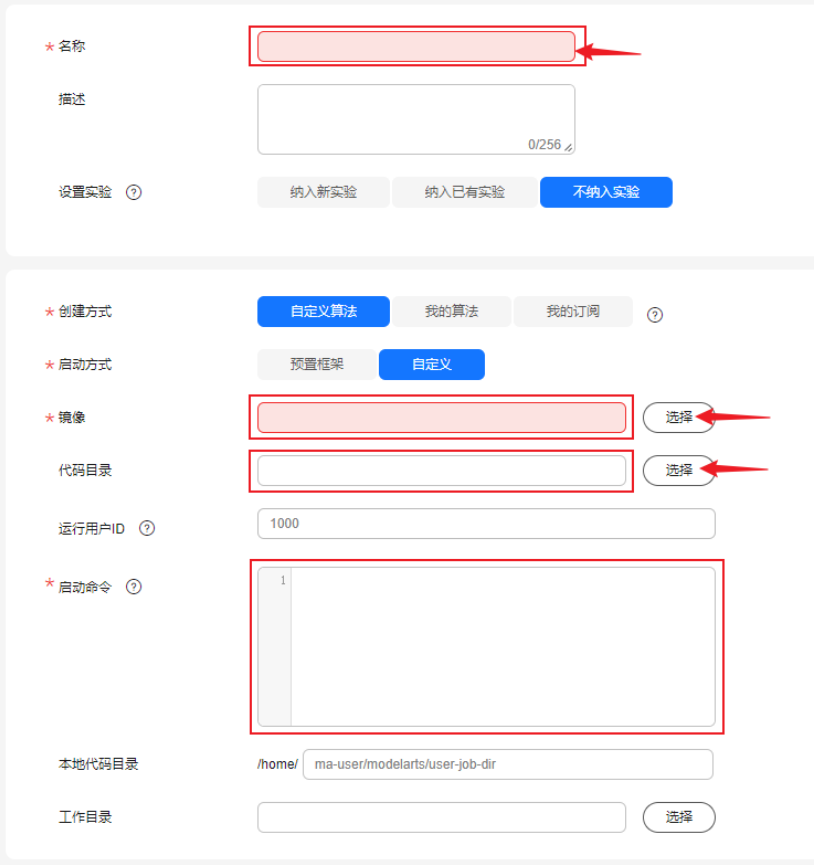
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-632 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-334**表格中的配置进行填写。

表 4-334 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。

- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-337**进行配置。

图 4-633 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.36.4 SFT 全参微调训练

前提条件

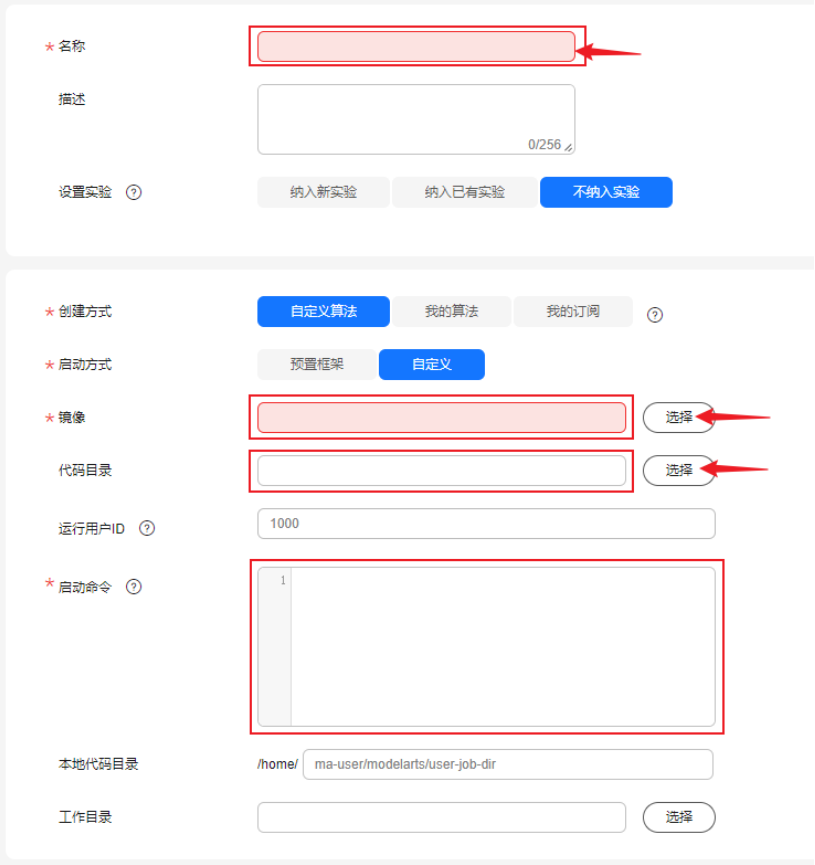
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-634 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表4-334表格中的配置进行填写。

图 4-635 环境变量

表 4-335 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-337**进行配置。

图 4-636 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.36.5 LoRA 微调训练

前提条件

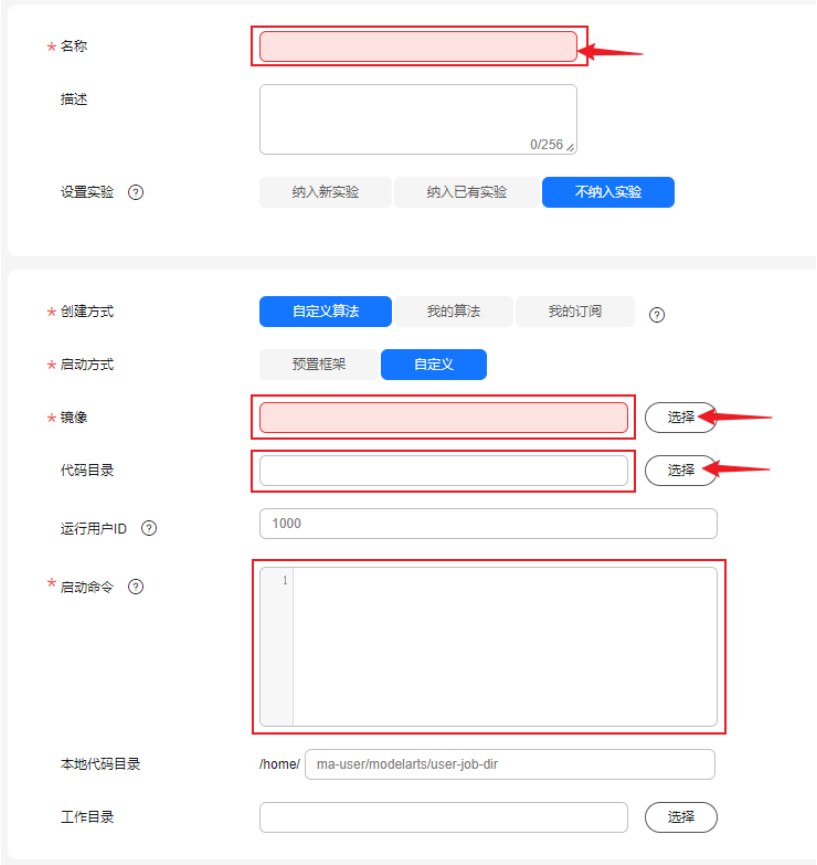
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的llm_train/AscendSpeed代码目录。

图 4-637 创建训练作业



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-334**表格中的配置进行填写。

表 4-336 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数**context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。如果不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

模型参数设置规定：

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。

- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-337**进行配置。

图 4-638 选择资源池规格

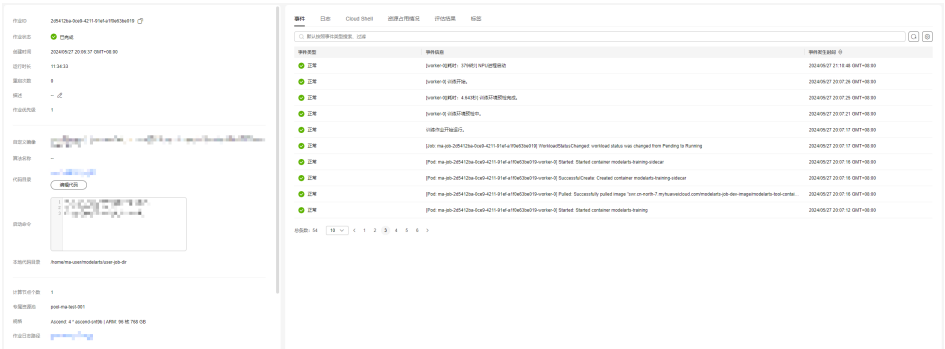
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.36.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

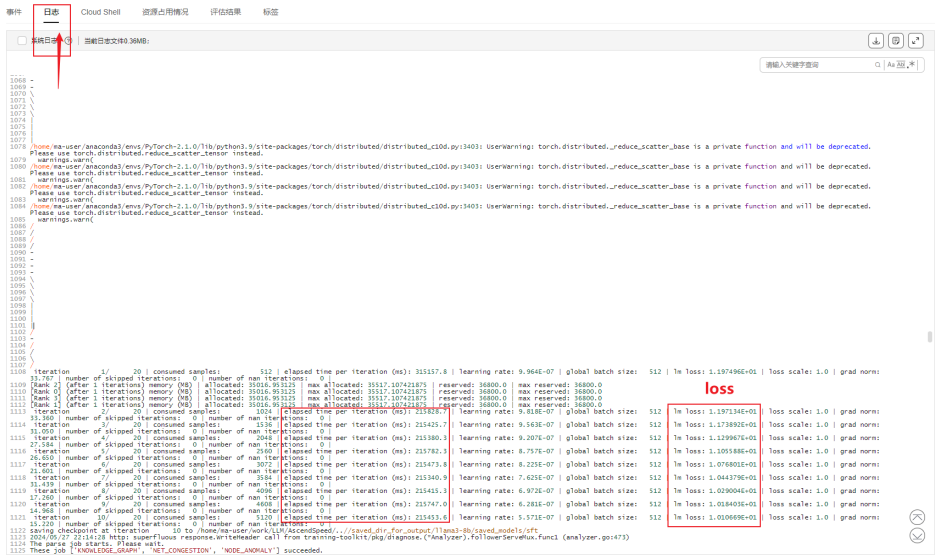
图 4-639 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-640 查看日志和性能



4.36.7 训练脚本说明

4.36.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如**表4-337**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-337 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend

4.36.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。如果未进行数据集预处理，则会自动执行**scripts/llama2/1_preprocess_data.sh**。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。

- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
```

```
    for key in self.args.json_keys:
        ## 写入磁盘
```

- 先调用self.get_tokenized_data()对数据集进行encode
- self.get_tokenized_data()中调用self._filter方法处理每一个sample
- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

  - 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

    ```
    [
      {"text": "document"},
      {"other keys": "optional content"}
    ]
    ```

  - 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

    ```
    doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
    ```

  - 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
```

```
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

```
[
 {
    "instruction": "指令（必填）",
    "input": "输入（选填）",
    "output": "模型回答（必填）",
 }
]
```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
```

```
    "labels": [labels]
}
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

   i. 对user和assistant的文本进行清洗

   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

   iii. input_ids是user_ids和assistant_ids的拼接

   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler中未定义prompt格式用于训练，因此在推理时可直接自定义prompt内容用于推理。

- **MOSSInstructionHandler解析**

```
def _filter(self, sample):
    messages = []
    tokenized_chats = []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        messages.append(dict(role=self.prompter.user_role, content=user))
        messages.append(dict(role=self.prompter.assistant_role, content=assistant))
        full_prompt = self.prompter.generate_training_prompt(messages)
        tokenized_full_prompt = self._tokenize(full_prompt)
        if not self.train_on_inputs:
            user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                          self.prompter.template.assistant_token + "\n"
            tokenized_user_prompt = self._tokenize(user_prompt)
            user_prompt_len = len(tokenized_user_prompt["input_ids"])
            tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
[user_prompt_len:]
        tokenized_chats.append(tokenized_full_prompt)
    for key in self.args.json_keys:
        sample[key] = [chat[key] for chat in tokenized_chats]
    return sample
```

- 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
{MOSS}
```

  - 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{Human}

"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-338 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.36.7.3 训练的权重转换说明

以**llama2-13b**举例，使用训练作业运行**obs_pipeline.sh**脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

如果用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。如果用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。

- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

注意：权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-339 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径

参数	示例	参数说明
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.36.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-641所示。

图 4-641 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-642所示。

图 4-642 修改 ChatGLMv3-6B tokenizer 文件

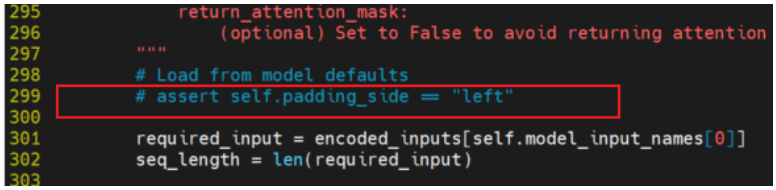
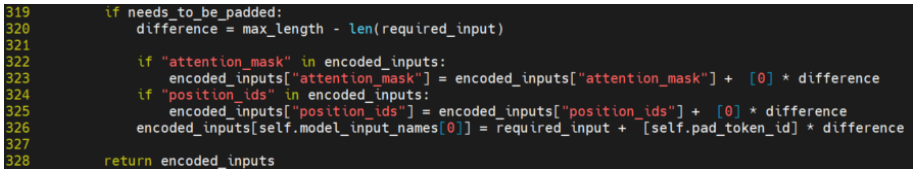


图 4-643 修改 ChatGLMv3-6B tokenizer 文件



GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图4-644所示。

图 4-644 修改 ChatGLMv4-9B tokenizer 文件

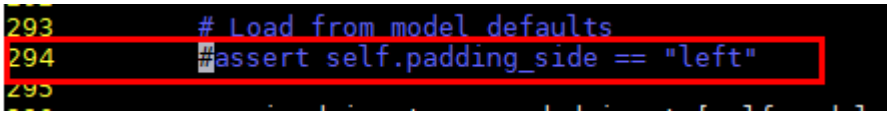
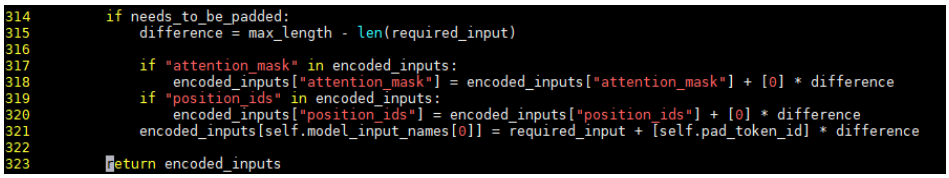


图 4-645 修改 ChatGLMv4-9B tokenizer 文件



Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图4-646所示。

图 4-646 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.36.8 常见错误原因和解决方法

4.36.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-337**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.36.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.37 主流开源大模型基于 Standard+OBS+SFS 适配 ModelLink PyTorch NPU 训练指导（6.3.908）

4.37.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅使用OBS的存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现数据灵活管理、高性能读取数据等。通过OBS上传训练所需的模型文件、训练数据等，再将OBS中的数据文件导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行。

文档更新内容

6.3.908版本相对于6.3.907版本新增如下内容：

- 文档和代码中新增对mistral和mixtral模型的适配，并添加训练推荐配置。
- 文档准备镜像步骤中，仅提供：直接使用基础镜像方案、ECS中构建新镜像方案，删除使用Notebook创建镜像方案。

- 文档中新增对 llama3 支持长序列文本（sequence_length > 32k）训练内容，例如新增参数context-parallel-size。
- 文档中针对数据集预处理时，handler-name参数的说明，新增不同 handler 对训练数据的拼接和推理prompt的构造等说明。

支持的模型列表

本方案支持以下模型的训练，如表4-340所示。

表 4-340 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
19		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**
20		qwen2-72b	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
21	GLMv4	glm4-9b	**https://huggingface.co/THUDM/glm-4-9b-chat** **说明** glm4-9b模型必须使用版本4b556ad4d70c38924cb8c120adbf21a0012de6ce
22	mistral	mistral-7b	**https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2**
23	mixtral	mixtral-8x7b	**https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1**

操作流程

图 4-647 操作流程图

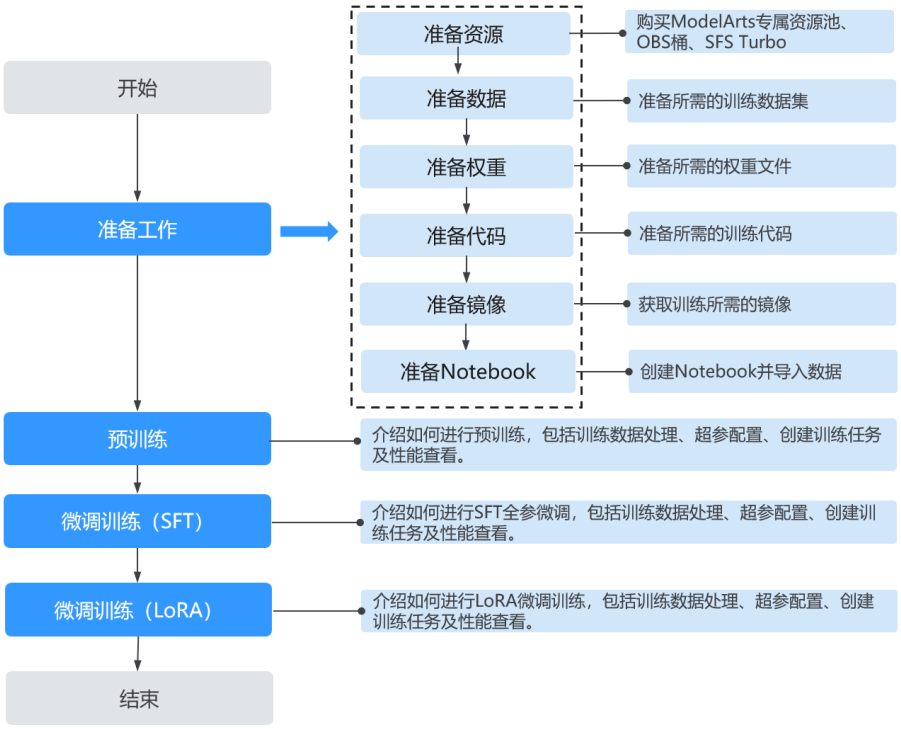


表 4-341 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.37.2 准备工作

4.37.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-348**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。具体过程请参考**创建OBS桶**，例如桶名：standard-llama2-13b。

由于ModelArts创建训练作业时，需要将作业日志输出至OBS桶中，因此创建OBS桶为必选项。用户可通过**OBS Browser+**、**obsutil**等工具访问和管理OBS桶，将代码、模型文件、数据集等数据上传或下载进行备份。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-648 创建 SFS Turbo

其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

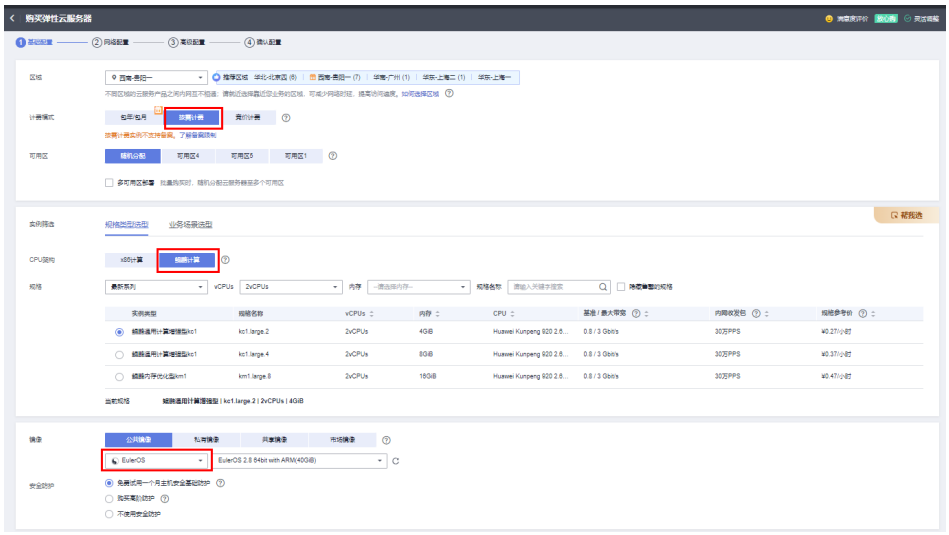
图 4-649 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 4-650 购买 ECS

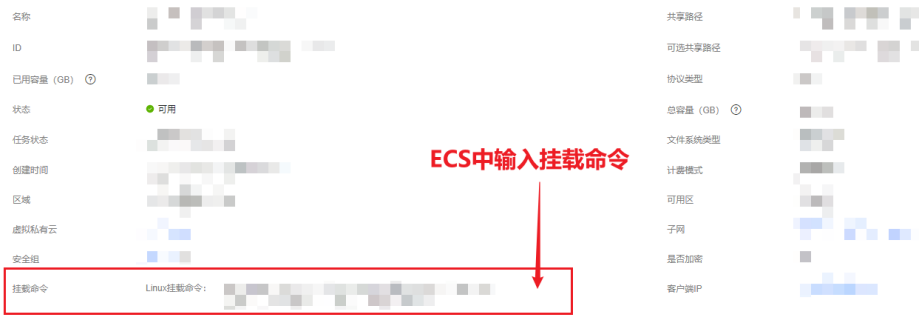


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看基本信息**图4-651**，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 4-651 SFS Turbo 基本信息



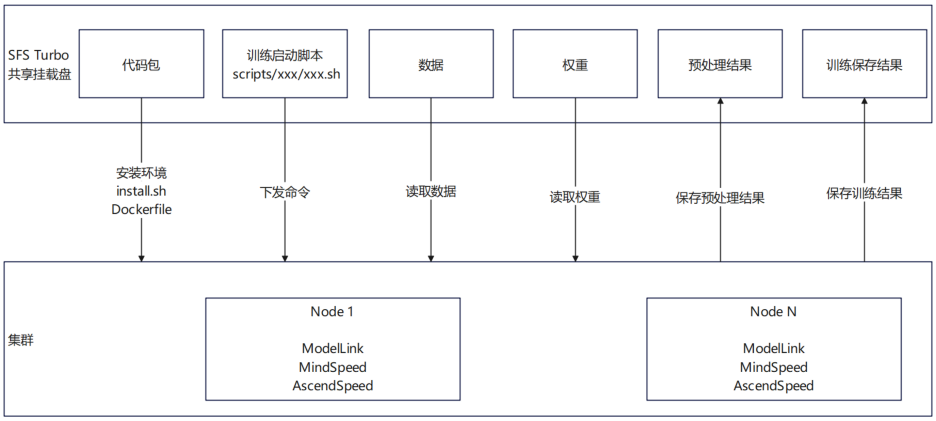
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-652 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表4-343**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.37.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

- **预训练数据**：用户也可以自行准备预训练数据。数据要求如下：

  使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

  ```
  {
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
  }
  ```

- **Alpaca 指令微调数据**：如上述提供的 alpaca_gpt4_data.json 数据集，数据集包含有以下字段：
  - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
  - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
  - output：生成的指令的答案。

  ```
  [
   {
     "instruction": "指令（必填）",
     "input": "输入（选填）",
     "output": "模型回答（必填）",
   }
  ]
  ```

- **MOSS 指令微调数据**：本案例中还支持 MOSS 格式数据，标准的.json格式的数据，内容包括可以多轮对话、指令问答。例如以下样例：

  ```
  {
    "conversation_id": 1,
    "meta_instruction": "",
    "num_turns": 3,
    "chat": {
      "turn_1": {
        "Human": "<|Human|>: 如何保障工作中遵循正确的安全准则？<eoh>\n",
        "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
        "Commands": "<|Commands|>: None<eoc>\n",
        "Tool Responses": "<|Results|>: None<eor>\n",
        "MOSS": "<|MOSS|>: 为了保障工作中遵循正确的安全准则，以下是一些建议：\n\n1.了解相关安全规定和标准：了解相关的安全规定和标准，并遵守它们。这可以包括公司和政府的安全标准，以及行业标准和最佳实践。\n\n2.培训和教育：确保您和您的同事接受了必要的培训和教育，以了解正确的安全准则和行为。\n\n3.使用正确的工具和设备：确保您使用正确的工具和设备，并且它们得到了正确的维护和保养。\n\n4.个人防护装备：确保您和您的同事穿戴正确的个人防护装备，如安全鞋、透明眼镜或面罩、手套等。\n\n5.创建一个安全文化：鼓励个人对安全的责任感和意识，并创建一个安全文化，使人们始终关注自己和他人的安全。\n\n6.持续监测和改进：持续监测和改进安全准则和程序，以确保它们保持最新，并适应变化的情况。\n\n这些措施可以帮助您和您的同事在工作中保障正确的安全准则，确保您的工作场所是一个安全的环境。<eom>\n"
      },
      "turn_2": { ... },
      "turn_3": { ... },
    "category": "Brainstorming"
  }
  ```

如果用户希望将MOSS数据集的Excel格式转换为.json格式。可使用代码中提供的**scripts/tools/ExcelToJson.py**工具，其转换的要求为：

- 本脚本可以处理的格式有：.xls .xlsx .csv .xlsb .xlsm .xlst
- MOSS 数据集的 Excel 中需要有三个列名称：conversation_id, Human, assistant
  - conversation_id: 指定的对话id，如果相同，转换后就放在同一conversation_id的不同turn_X下。如果为空，则放在新的conversation_id下。
  - Human: 数据集中每条数据的输入。
  - assistant: 数据集中每条数据的输出。
- 运行命令示例：

```
1. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --test_count=3
2. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2
(随机选择十分之二的总数据量作为测试集，小数时则四舍五入)
3. python ExcelToJson.py --user_id=001 --excel_addr=xxx.xlsx(.csv) --dataset_name=example --proportion=2 --test_count=3
(随机选择 3个数据作为测试集)
```

  - user_id: 用户的唯一不重复的ID值，必选。
  - excel_addr: 待处理的excel文件的地址，必选。
  - dataset_name: 处理后的数据集名称，必选。
  - proportion: 测试集所占份数，范围[1,9]，可选。
  - test_count: 测试集的个数，范围[1,处理后数据集总长度 - 1]，可选。(用户在输入test_count时，要小于 Excel文件中指定的不同conversation_id的个数 + conversation_id为空的个数)
  - proportion 和 test_count 二选一即可，如果同时输入，则优先使用test_count，如果都未输入，则返回处理失败 False。

上传数据集至 SFS Turbo

准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。可通过两种方式，将数据集上传至SFS Turbo中。

方式一：将下载的原始数据通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/training_data目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具。数据存放参考目录：

```
/mnt/sfs_turbo/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
2. 利用**OBS Browser+工具**将下载的数据集上传至创建的文件夹目录下。得到OBS下数据集结构：
```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```
3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.37.2.3 准备权重

获取对应模型的权重文件，获取链接参考**表4-340**。权重文件下载有如下几种方式，但不仅限于以下方式：

- **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。文件会直接下载用户本地，需要再上传至SFS Turbo中。
- **方法二：huggingface-cli**：**huggingface-cli**是Hugging Face官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Llama2-70B为例：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --local-dir <模型下载路径>
```
  如果要下载指定版本的模型文件，则命令如下：
```
huggingface-cli download --resume-download meta-llama/Llama-2-70b-chat-hf --revision <模型版本>  --local-dir <模型下载路径>
```
- **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
- **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

随后可通过以下两种方式，将下载到本地的模型文件上传至SFS Turbo中。

本地上传权重文件至 SFS Turbo

通过以下两种方式将下载到本地的模型文件上传至SFS Turbo中。方式一操作简单，但是数据传输速度比较慢，费时间。方式二操作相对方式一复杂一些，但是数据传输速度较快。

方式一：将已下载的模型文件通过SSH直接上传至SFS Turbo中。具体步骤如下：

1. 进入到/mnt/sfs_turbo/目录下。创建目录“training_data”，将原始数据存放在/mnt/sfs_turbo/model目录下。
2. 通过拖拽文件的方式，上传文件。使用CloudShell或者其它SSH远程工具

方式二：通过OBS Browser+将数据上传至OBS，最后在ECS中使用obsutil工具将OBS数据下载至SFS Turbo中。具体步骤如下：

1. 在**创建OBS桶**创建的桶下创建文件夹用以存放模型，例如在桶standard-llama2-13b中创建文件夹model/llama-2-13b-hf。
2. 利用**OBS Browser+工具**将下载的模型文件上传至创建的文件夹目录下。

3. 在ECS服务器中安装**obsutil工具**，具体命令可参考**obsutil工具快速使用**，将OBS桶中的数据下载至SFS Turbo中。注意：需要使用用户账号中的AK和SK进行签名验证，确保通过授权的账号才能访问指定的OBS资源。

4.37.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-342**所示。

表 4-342 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.908-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.908代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
                |——llama2           # llama2系列模型执行脚本的文件夹
                |——llama3           # llama3系列模型执行脚本的文件夹
                |——qwen             # Qwen系列模型执行脚本的文件夹
                |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |—— ...
                |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |—— install.sh       # 环境部署脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

代码上传至 SFS Turbo

将AscendSpeed代码包AscendCloud-LLM-xxx.zip直接上传至ECS服务器中的SFS Turbo中，例如存放在/mnt/sfs_turbo/AscendCloud-LLM-xxx.zip目录下并解压缩。

```
unzip AscendCloud-*.zip
```

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至SFS Turbo后，目录结构如下。

```
/mnt/sfs_turbo/
|——llm_train                  # 解压代码包后自动生成的代码目录，无需用户创建
```

```
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          # 目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain    # 预训练加载的数据
          |── finetune    # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |─── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                          #微调数据文件
|── models                                          #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
|── tokenizers                                      #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-13b-hf
```

4.37.2.5 准备镜像

4.37.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-343 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**的方式（**可二选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。

- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  如果用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

使用以上方案时，都会下载Megatron-LM、MindSpeed、ModelLink源码至AscendSpeed文件夹中。下载后的源码文件结构如下：

```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/             # 训练需要的启动脚本
  |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/           # MindSpeed昇腾大模型加速库
  |——ModelLink/           # ModelLink端到端的大语言模型方案
      |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

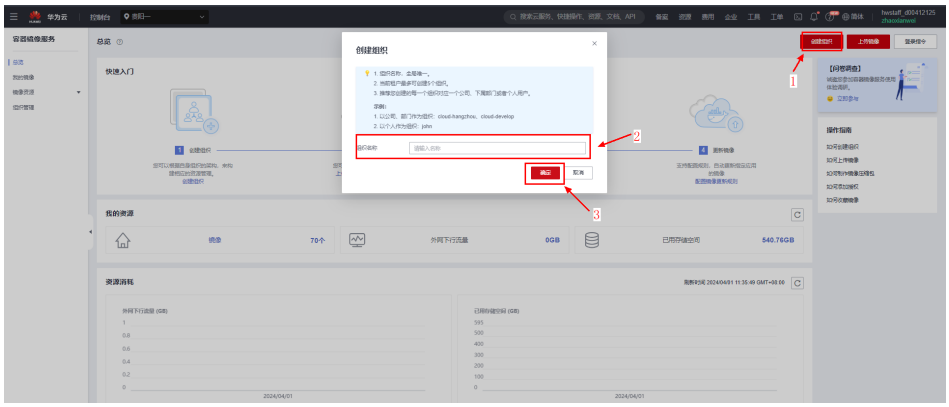
训练作业的资源池以及ECS都需要连通公网，否则会安装和下载失败。资源池打通公网配置请参见**配置Standard专属资源池访问公网**，ECS打通公网配置请参见**ECS绑定弹性公网IP**。

4.37.2.5.2 ECS 获取和上传基础镜像

Step1 创建镜像组织

在SWR服务页面创建镜像组织。

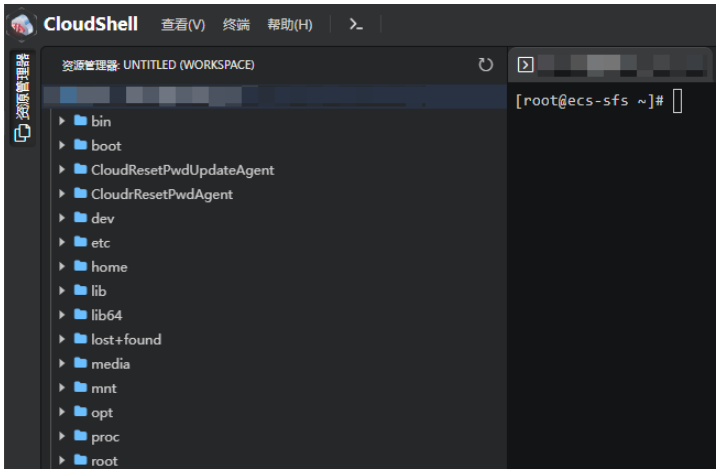
图 4-653 创建镜像组织



Step2 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录如图所示。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

图 4-654 CloudShell 远程登录界面



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

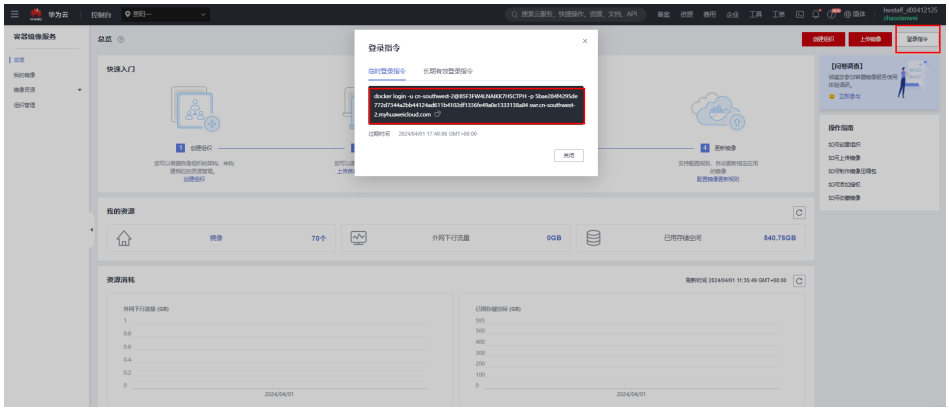
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-343**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-655 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.37.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

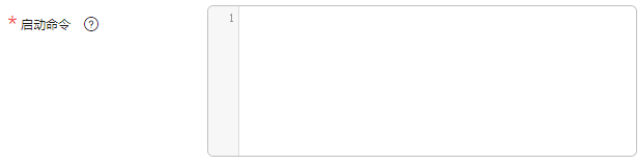
由于基础镜像内需要安装固定版本依赖包，如果直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-656**中都需要执行 install.sh文件，来安装依赖以及下载完整代码。

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而install.sh则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-656 训练作业启动命令



4.37.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

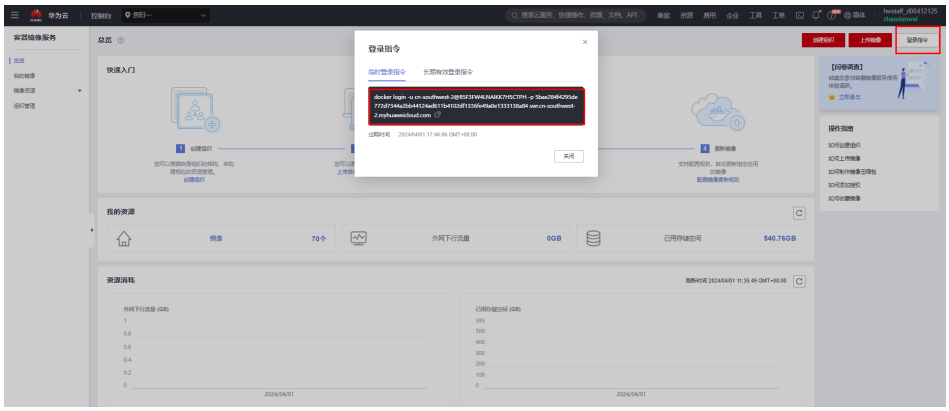
获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-342**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   cd ./llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" && \
   git config --global user.name "Your Name" && \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称> .
   ```
   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称> .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-657 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.37.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-344**所示。其他超参均有默认值，可以参考**表4-347**按照实际需求修改。

表 4-344 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	**可添加**。该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。

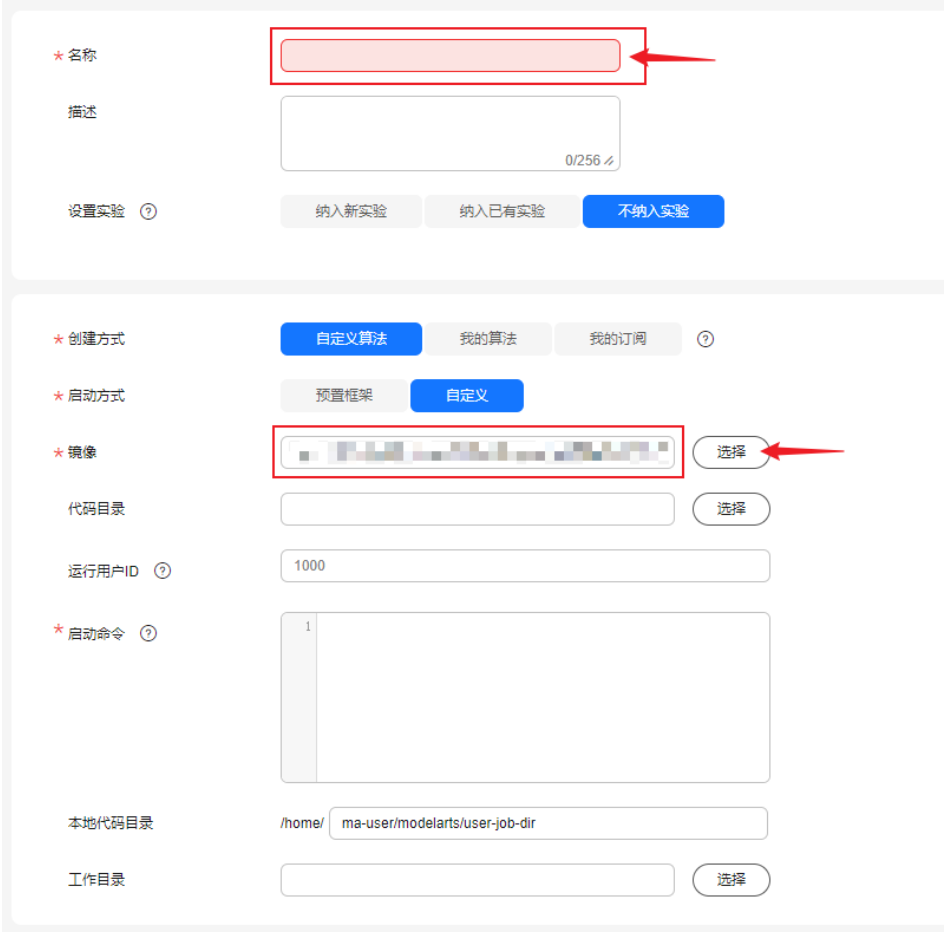
参数	示例值	参数说明
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-658 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

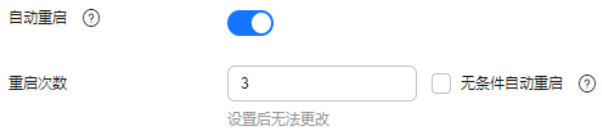
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-659 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-348**进行配置。

图 4-660 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-661 选择 SFS Turbo

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.37.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-345**所示。其他超参均有默认值，可以参考**表4-347**按照实际需求修改。

表 4-345 训练超参配置说明

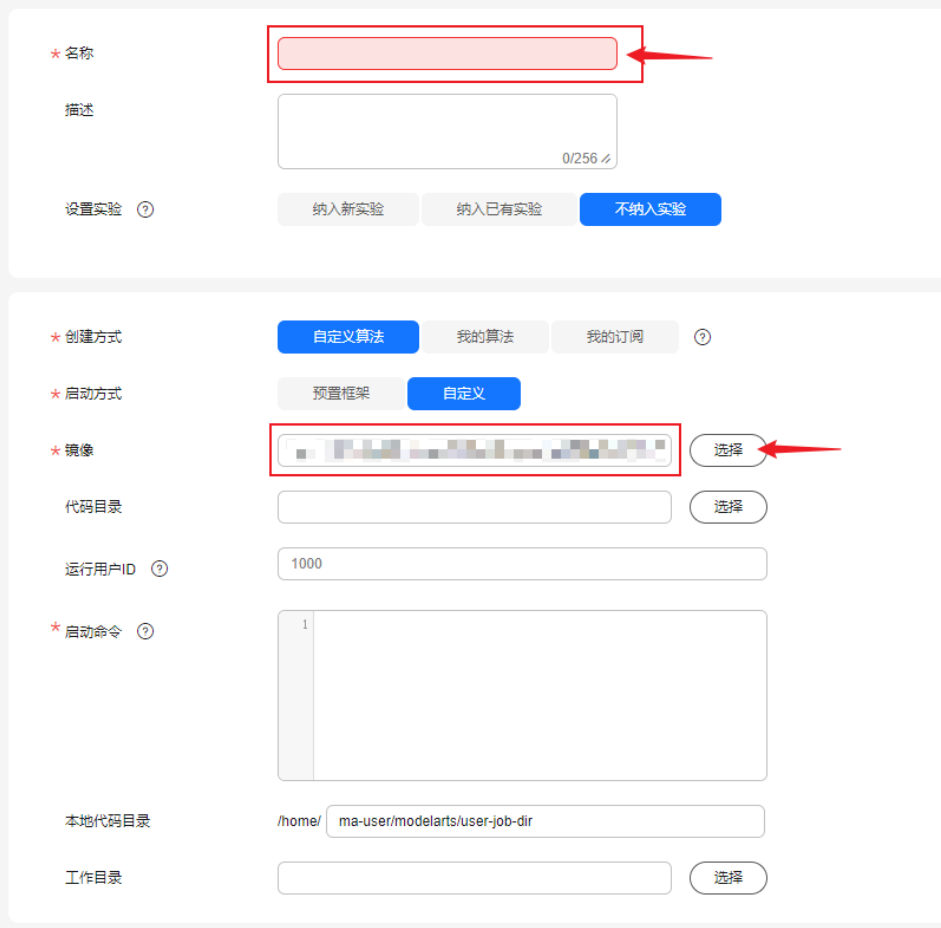
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-662 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

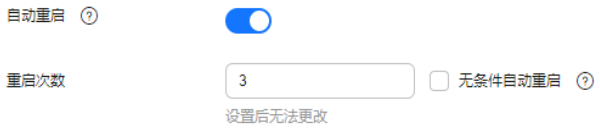
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-663 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-348**进行配置。

图 4-664 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-665 选择 SFS Turbo

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.37.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-346**所示。其他超参均有默认值，可以参考**表4-347**按照实际需求修改。

表 4-346 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。

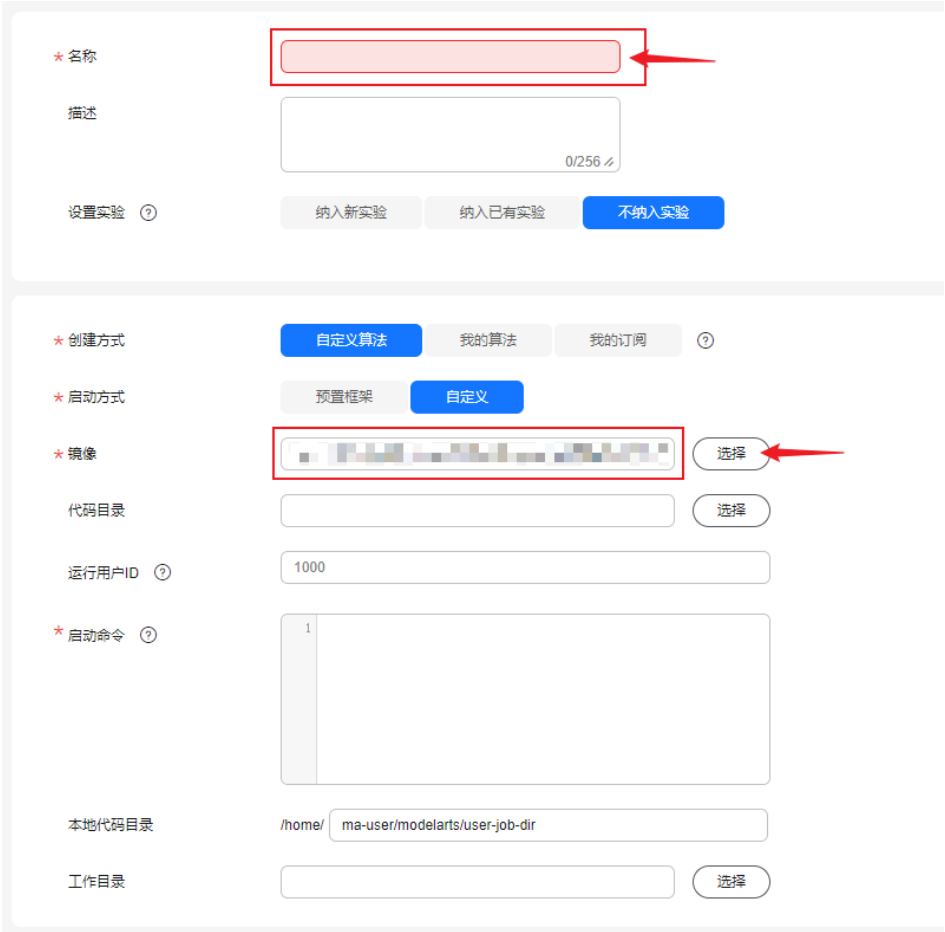
参数	示例值	参数说明
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-666 选择镜像



如果镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

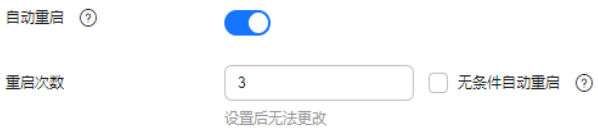
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

如果镜像使用**ECS中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-667 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-348**进行配置。

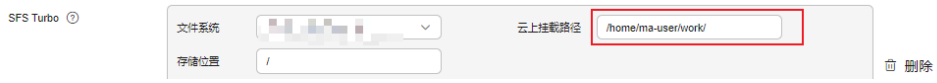
图 4-668 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/

- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 4-669 选择 SFS Turbo



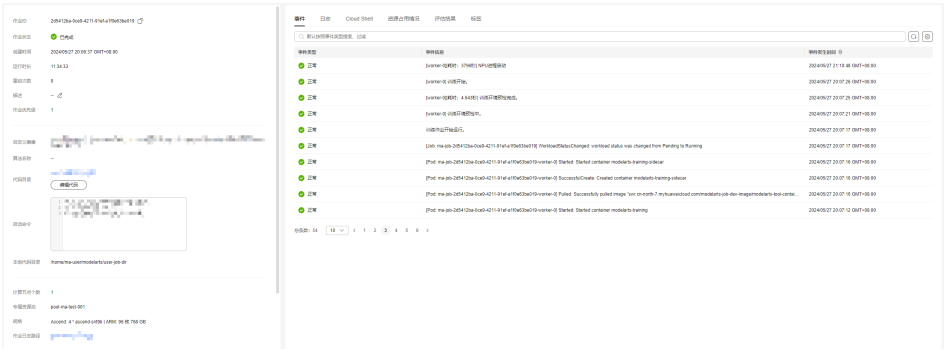
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.37.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

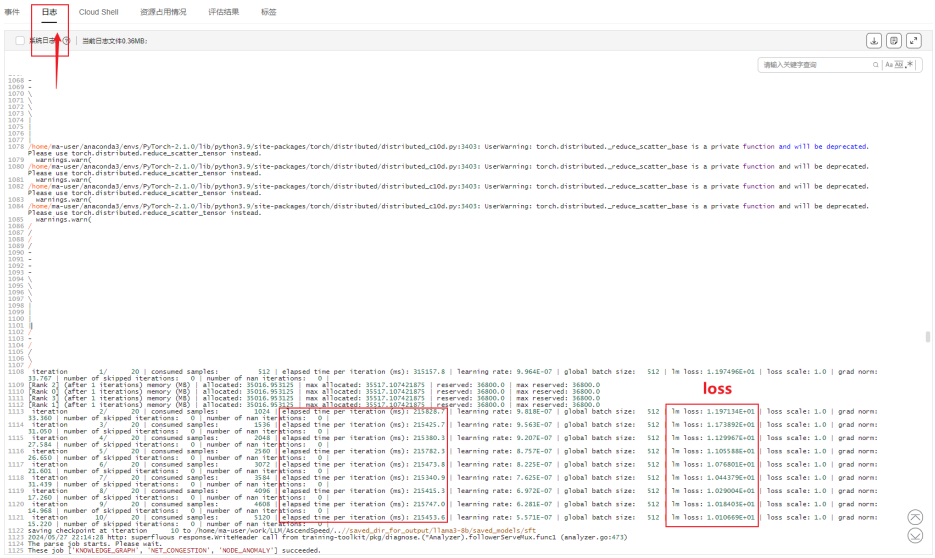
图 4-670 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-671 查看日志和性能



4.37.7 训练脚本说明

4.37.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-347 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。

参数	示例值	参数说明
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。对应训练参数 **tensor-model-parallel-size** 。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。对应训练参数 **pipeline-model-parallel-size** 。
CP	1	表示context并行，默认为1。应用于训练长序列文本的模型。如果训练时SEQ_LEN超过32768长度，则推荐增加CP值（CP ≥ 2）。对应训练参数 **context-parallel-size** 。 （此参数目前**仅适用于**Llama3系列模型长序列训练）
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

模型参数设置规定

- TP张量并行 、PP流水线并行、CP context并行的参数设置：TP×PP×CP的值要被NPU数量（word_size）整除。
- TP×CP的值要被模型参数中 num_attention_heads 整除。
- MBS（micro-batch-size）、GBS（global-batch-size）的设置：需要遵循GBS/MBS的值能够被NPU/(TP×PP×CP)的值进行整除。

模型推荐的参数与 NPU 卡数设置

不同模型推荐的训练参数和计算规格要求如表4-348所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-348 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
22	mistral	mistral-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
23	mixtral	mixtral-8x7b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=8	2*节点 & 8*Ascend

4.37.7.2 训练的数据集预处理说明

以**llama2-13b**举例，使用训练作业运行：**0_pl_pretrain_13b.sh**训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。支持 .parquet \ .csv \ .json \ .jsonl \ .txt \ .arrow 格式。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）

- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
       level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
```

```
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- 支持的是预训练数据风格，会根据参数args.json_keys的设置，从数据集中找到对应关键字的文本内容。例如本案例中提供的 train-00000-of-00001-a09b74b3ef9c3b56.parquet 预训练数据集的关键字为“text”，格式如下：

  ```
  [
    {"text": "document"},
    {"other keys": "optional content"}
  ]
  ```

- 训练数据构造：在 _filter 函数中会根据关键字将内容提取后，直接使用模型特定的tokenizer分词器，将对应关键字中的内容进行预处理。最后实际用于训练的内容如下：

  ```
  doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
  ```

- 推理prompt构造：由于 GeneralPretrainHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

  ```
  def _filter(self, sample):
      messages = self._format_msg(sample)
      full_prompt = self.prompter.generate_training_prompt(messages)
      tokenized_full_prompt = self._tokenize(full_prompt)
      if self.args.append_eod:
          tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
          tokenized_full_prompt["attention_mask"].append(1)
          tokenized_full_prompt["labels"].append(self.tokenizer.eod)
      if not self.train_on_inputs:
          user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
              self.prompter.template.assistant_token + "\n"
          tokenized_user_prompt = self._tokenize(user_prompt)
          user_prompt_len = len(tokenized_user_prompt["input_ids"])
          tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
      for key in self.args.json_keys:
          tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
      return tokenized_full_prompt
  ```

  - 本案例中 alpaca_gpt4_data.json 数据集包含有以下字段：

    - instruction：描述模型应执行的任务。指令中的每一条都是唯一的。
    - input：任务的可选上下文或输入。instruction 对应的内容会与 input 对应的内容拼接后作为指令，即指令为 instruction\ninput。
    - output：生成的指令的答案。

    ```
    [
     {
        "instruction": "指令（必填）",
        "input": "输入（选填）",
        "output": "模型回答（必填）",
     }
    ]
    ```

- 训练数据构造：在 _filter 函数中会使用 Alpaca 微调指令的模板 self.prompter 将数据集中 instruction、input、output 关键字的内容进行拼接，并用于训练。拼接方式如下，其中 {instruction}、{input}、{output} 分别对应数据集中 instruction、input、output 关键字的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction} + "\n" + {input}

"### Response:"
{output}
```

- 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {instruction} 为用户推理测试时输入的内容。

```
"Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

"### Instruction:"
{instruction}

"### Response:"
```

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

  b. 循环处理其中的单轮对话

  c. 在单轮对话中

     i. 对user和assistant的文本进行清洗

     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

     iii. input_ids是user_ids和assistant_ids的拼接

     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

  d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- 训练数据构造：在 _filter 函数中会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

  ```
  user_token = [195]
  assistant_token = [196]
  input_ids = user_token + {Human} + assistant_token + {MOSS}
  ```

- 推理prompt构造：MOSSMultiTurnHandler 中未定义 prompt 格式用于训练，因此在推理时可直接自定义 prompt 内容用于推理。

- **MOSSInstructionHandler解析**

  ```
  def _filter(self, sample):
      messages = []
      tokenized_chats = []
      for turn in sample["chat"].values():
          if not turn:
              continue
          user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
          assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
          messages.append(dict(role=self.prompter.user_role, content=user))
          messages.append(dict(role=self.prompter.assistant_role, content=assistant))
          full_prompt = self.prompter.generate_training_prompt(messages)
          tokenized_full_prompt = self._tokenize(full_prompt)
          if not self.train_on_inputs:
              user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
                            self.prompter.template.assistant_token + "\n"
              tokenized_user_prompt = self._tokenize(user_prompt)
              user_prompt_len = len(tokenized_user_prompt["input_ids"])
              tokenized_full_prompt["labels"] = [-100] * user_prompt_len + tokenized_full_prompt["labels"]
  [user_prompt_len:]
          tokenized_chats.append(tokenized_full_prompt)
      for key in self.args.json_keys:
          sample[key] = [chat[key] for chat in tokenized_chats]
      return sample
  ```

  - 训练数据构造：在 _filter 函数中同样会读取 MOSS 数据集的“Human”和“MOSS”字段的文本内容，并将内容中"<|Human|>: "、"<|MOSS|>:"、"<eom>"字符串去除。随后将“Human”和“MOSS”的文本内容进行拼接，拼接方式如下，其中 {Human}、{MOSS}分别对应Human、MOSS关键字的内容。

    ```
    "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

    "### Instruction:"
    {Human}

    "### Response:"
    {MOSS}
    ```

  - 推理prompt构造：通过微调训练后进行推理时，同样需要根据训练时的prompt模板来构造prompt内容。prompt拼接格式如下，其中 {Human} 为用户推理测试时输入的内容。

    ```
    "Below is an instruction that describes a task, paired with an input that provides further context. \n Write a response that appropriately completes the request. \n Please note that you need to think through your response logically and step by step."

    "### Instruction:"
    {Human}
    ```

```
"### Response:"
```

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以llama2为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-349 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.37.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、

tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-350 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.37.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-672所示。

图 4-672 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-673 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297             """
298             # Load from model defaults
299             # assert self.padding_side == "left"
300
301             required_input = encoded_inputs[self.model_input_names[0]]
302             seq_length = len(required_input)
303
```

图 4-674 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-675 修改 ChatGLMv4-9B tokenizer 文件

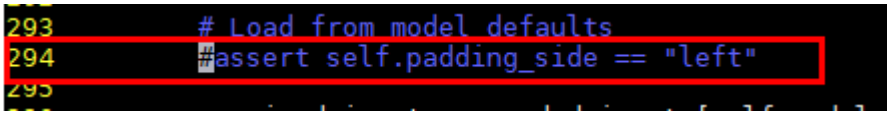
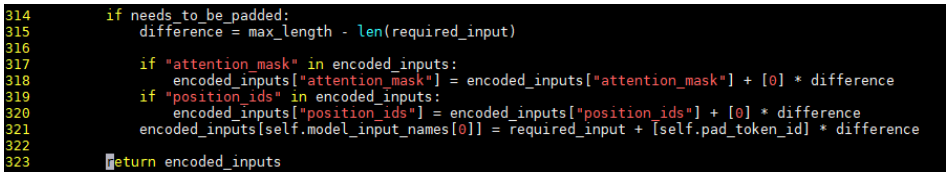


图 4-676 修改 ChatGLMv4-9B tokenizer 文件

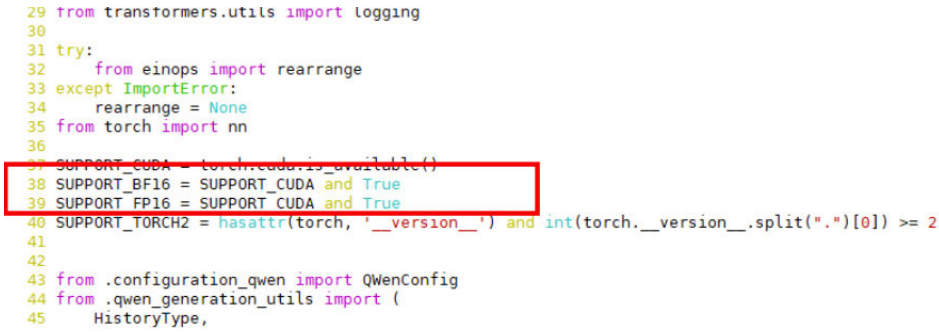


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-677 修改 Qwen tokenizer 文件



4.37.8 常见错误原因和解决方法

4.37.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法：

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-348**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.37.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.38 主流开源大模型基于 Lite Server 适配 ModelLink PyTorch NPU 训练指导（6.3.907）

4.38.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。训练框架使用的是ModelLink。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考表4-353获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如表4-351所示。

表 4-351 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 4-678 操作流程图

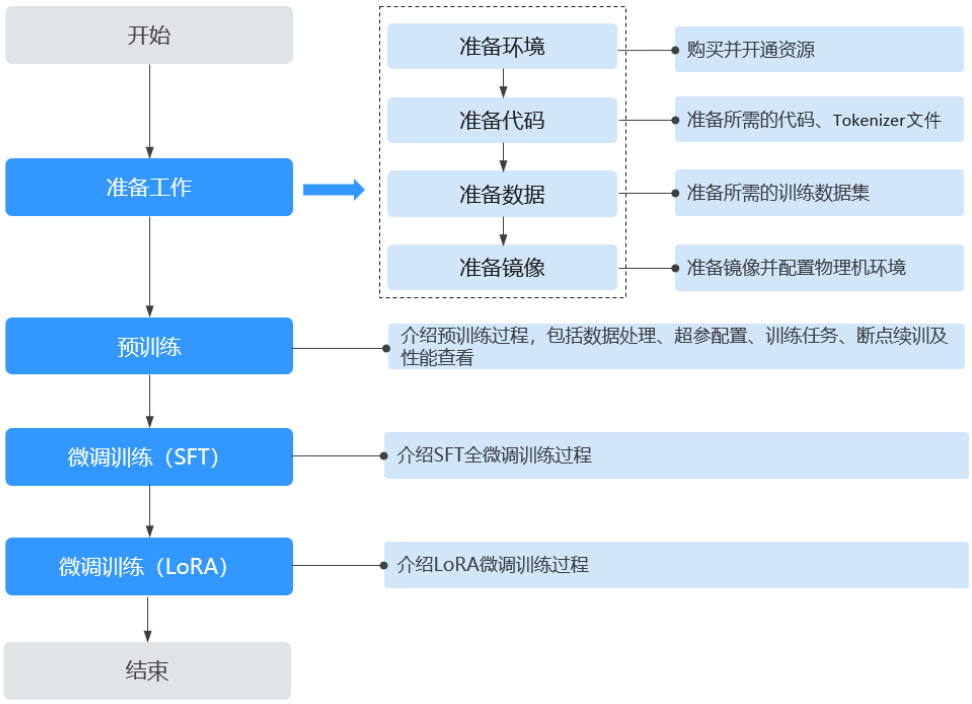


表 4-352 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.38.2 准备工作

4.38.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-361**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.38.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-353**所示，模型列表、对应的开源权重获取地址如**表4-354**所示。

表 4-353 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

表 4-354 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
           |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
           |——scripts/             # 训练需要的启动脚本
               |——llama2           # llama2系列模型执行脚本的文件夹
               |——llama3           # llama3系列模型执行脚本的文件夹
               |——qwen             # Qwen系列模型执行脚本的文件夹
               |——qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
               |—— ...
               |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
               |—— install.sh       # 环境部署脚本
           |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
   |——llm_inference                  # 推理代码包
   |——llm_tools                      # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在scripts文件夹中。

```
${workdir}（例如/home/ma-user/ws）
|——llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
```

```
    |── AscendSpeed              # 代码目录
          |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── data              # 预处理后数据
             |── pretrain    # 预训练加载的数据
             |── finetune    # 微调加载的数据
       |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── logs              # 训练过程中日志（loss、吞吐性能）
                   |─── saved_models
                |── lora              # lora微调输出权重
                |── sft               # 增量训练输出权重
                |── pretrain          # 预训练输出权重
|── models                          #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── tokenizers                      #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                  #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

注意：多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.38.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

```
{
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
}
```

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

注意：多机情况下，只有在rank_0节点进行数据预处理，转换权重等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下。

4.38.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-355 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a

表 4-356 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
驱动	23.0.5
PyTorch	2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --cpus 192 \
   ```

```
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
```
docker exec -it ${container_name} bash
```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```
4. 使用ma-user用户安装依赖包。
```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```
5. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，如果手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：
```
AscendSpeed/
  |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
  |——scripts/             # 训练需要的启动脚本
  |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
  |——MindSpeed/           # MindSpeed昇腾大模型加速库
  |——ModelLink/           # ModelLink端到端的大语言模型方案
      |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
      |——...
```

如果git下载代码时报错，请参见**Git下载代码时报错**解决。

4.38.3 预训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-357**所示。其他超参均有默认值，可以参考**表4-360**按照实际需求修改。

表 4-357 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

请根据**Step2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

示例:

```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第三台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第四台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

示例:
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
或者:
sh scripts/llama2/0_pl_pretrain_13b.sh

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-679 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

4.38.4 SFT 全参微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-357**所示。其他超参均有默认值，可以参考**表4-360**按照实际需求修改。

表 4-358 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

示例:
```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
# 第三台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
# 第四台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

示例:
```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
```
或者:
```
sh scripts/llama2/0_pl_sft_13b.sh
```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.38.5 LoRA 微调训练

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-357**所示。其他超参均有默认值，可以参考**表4-360**按照实际需求修改。

表 4-359 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。如果用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。如果用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。如果用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。如果用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。如果用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。如果用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-361**所示。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

示例:

```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
# 第三台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
# 第四台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

示例:

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
```

或者:

```
sh scripts/llama2/0_pl_lora_13b.sh
```

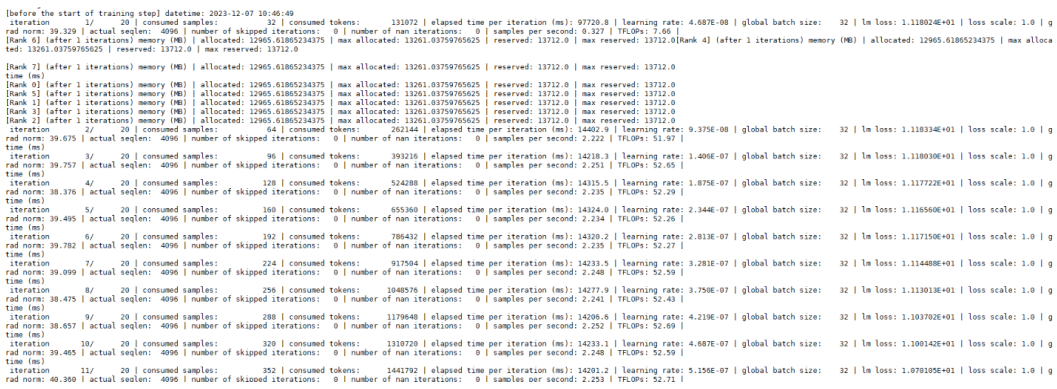
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

4.38.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-680 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

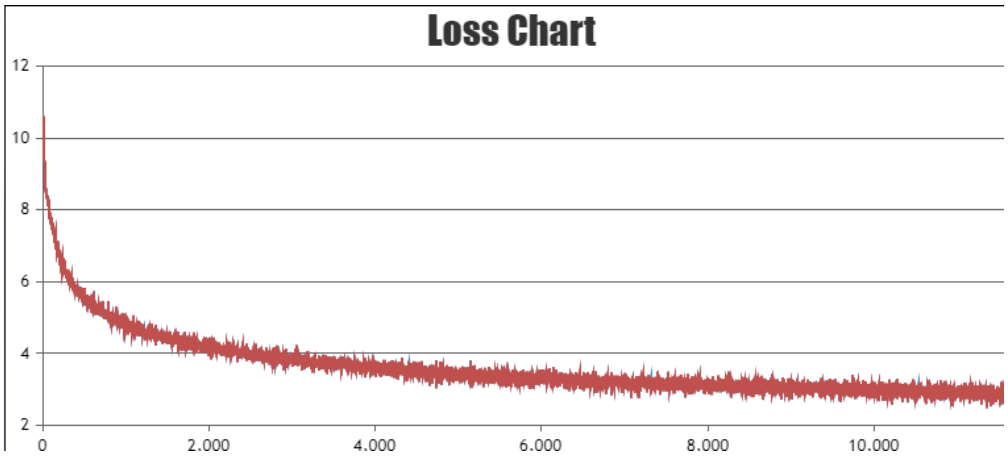
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-360**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-681**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 4-681 Loss 收敛情况（示意图）



4.38.7 训练脚本说明

4.38.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

如果用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

如果用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 4-360 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。

参数	示例值	参数说明
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-361所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-361 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

4.38.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

如果已完成数据集预处理，则直接执行预训练任务。如果未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。

- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

  所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
```

```
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 对数据集 full_prompt 中的 user_prompt 进行 mask 操作。

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

  a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行
  b. 循环处理其中的单轮对话
  c. 在单轮对话中
     i. 对user和assistant的文本进行清洗
     ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids
     iii. input_ids是user_ids和assistant_ids的拼接
     iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids
  d. attention_mask是和input_ids等长的全1序列
  e. 返回input_ids\attention_mask\labels的字典
  f. 处理完单一sample

  注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-362 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.38.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

如果已完成权重转换，则直接执行预训练任务。如果未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。

- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**如果用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。如果缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

如果用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。

- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-363 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.38.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-682所示。

图 4-682 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-683 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-684 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-685 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-686 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-687 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.38.8 常见错误原因和解决方法

4.38.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-361**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

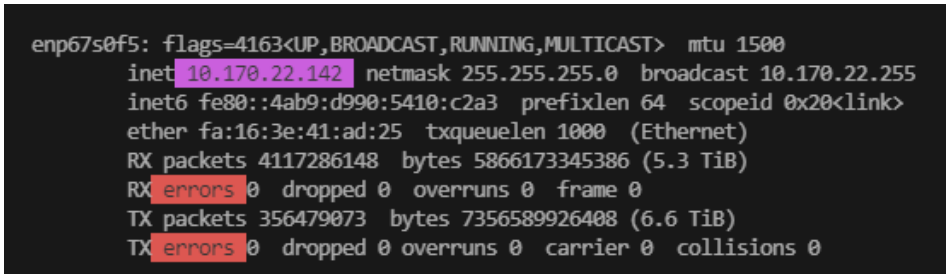
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.38.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-688 网卡名称错误



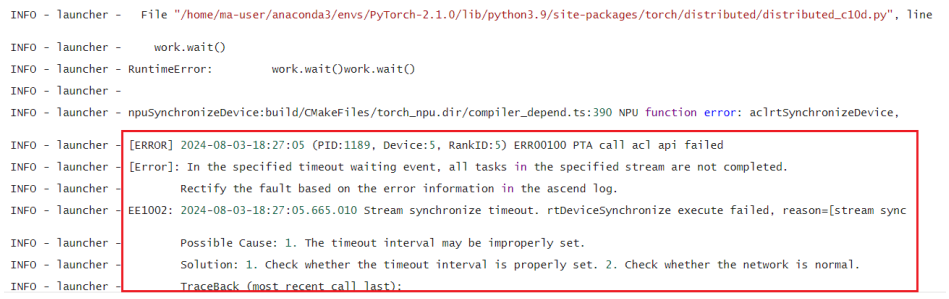

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.38.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-689 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.38.8.4 Git 下载代码时报错

在执行scripts/install.sh安装命令或使用Dockerfile构建镜像时，如遇到git下载代码出现以下类似的报错信息，关闭git验证即可。

报错信息：

```
fatal: unable to access 'https://gitee.com/ascend/ModelLink.git/': error setting certificate verify locations:
CAfile: /etc/pki/tls/certs/ca-bundle.crt CApath: none
```

关闭git验证命令如下：

```
git config --global http.sslverify false
```

4.39 主流开源大模型基于 Lite Server 适配 LlamaFactory PyTorch NPU 训练指导（6.3.907）

4.39.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的微调方案，包括sft全参和lora 微调。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表4-366**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。
- Server驱动版本要求23.0.5

训练支持的模型列表

本方案支持以下模型的训练，如**表4-364**所示。

表 4-364 支持的模型列表及权重文件地址

支持模型	Template	支持模型参数量	权重文件获取地址
Llama3	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**

支持模型	Template	支持模型参数量	权重文件获取地址
Qwen1.5	qwen	qwen1.5-0.5b	https://huggingface.co/Qwen/Qwen1.5-0.5B
		qwen1.5-1.8b	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
		qwen1.5-4b	https://huggingface.co/Qwen/Qwen1.5-4B
		qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
Yi	yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
Qwen2	qwen	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
Falcon2	falcon	falcon-11B	https://huggingface.co/tiiuae/falcon-11B

表 4-365 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。

阶段	任务	说明
微调训练	指令监督微调训练	介绍如何进行SFT全参微调/lora微调、训练任务、性能查看。

4.39.2 准备工作

4.39.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-372**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

购买共享存储硬盘资源（多机训练场景）

用户若购买开通多个节点机器资源，并使用多机进行分布式训练时，则需要用户购买可挂载的存储硬盘资源，以实现多机共同访问同一存储硬盘资源。ModelArts Lite Server 支持配置的存储方案请参考**配置Lite Server存储**。其中访问方式中，可支持在裸金属服务器中挂载的有弹性文件服务SFS和云硬盘EVS。

4.39.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-366**所示，模型列表、对应的开源权重获取地址如**表4-364**所示。

表 4-366 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                      # 模型训练代码包
       |——LLaMAFactory                 # 基于LLaMAFactory的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——demo.yaml            # 样例yaml配置文件
            |——demo.sh              # 指令微调启动shell脚本
            |——intall.sh            # 需要的依赖包
            |——LLaMA-Factory        # LLaMAFactory的代码目录
       |——AscendSpeed                # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
    |—— LLaMAFactory              # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——demo.sh              # 指令微调启动shell脚本
          |——demo.yaml            # 样例yaml配置文件
          |——intall.sh            # 需要的依赖包
          |——LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考Step3 启动容器镜像
              |——data             # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|—— tokenizers                    #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
    |—— Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表4-369表格中output_dir参数值
|—— saved_dir_for_output_lf       # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |—— ${model_name}             # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   unzip AscendCloud-LLM-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

具体步骤如下：

进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

```
cd /home/ma-user/ws
mkdir -p tokenizers/Qwen2-72B
```

4.39.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-367 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a

表 4-368 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
驱动	23.0.5
PyTorch	2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如：  chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
```
docker exec -it ${container_name} bash
```
4. 使用ma-user用户安装依赖包。
```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

4.39.2.4 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前指令微调数据集支持alpaca格式和sharegpt格式的数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加*数据集描述*。

关于数据集文件的格式及配置，请参考**data/README_zh.md**的内容。可以使用HuggingFace/ModelScope上的数据集或加载本地数据集。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。
```
cd  /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
```
2. 将自定义原始数据如demo.json放置在此处。

   数据存放参考目录结构如下：
```
${workdir}（例如/home/ma-user/ws/llm_train ）
 |── LLaMAFactory/data
     |── alpaca_en_demo.json                # 代码原有数据集
     |── identity.json                      # 代码原有数据集
     ...
     |── demo.json                          # 自定义数据集
```
3. 更新代码目录下 data/dataset_info.json 文件。关于数据集文件的格式及配置，请参考 **data/README_zh.md** 的内容。
```
vim dataset_info.json
```

4.39.3 指令监督微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

Step2 修改训练 yaml 文件配置

LlamaFactroy配置文件为yaml文件，启动训练前需修改yaml配置文件，yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示：

步骤1 选择指令微调类型

- sft，复制**sft_yaml样例模板**内容覆盖demo.yaml文件内容。
- lora，复制**lora_yaml样例模板**内容覆盖demo.yaml文件内容。

步骤2 修改**yaml文件**(demo.yaml)的参数如**表4-369**所示

表 4-369 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表4-364**中的**template列**
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
dataset	identity,alpaca_en_demo	【**可选**】指定用于训练的数据集，数据集都放置在此处为identity，alpaca_en_demo表示使用了两个数据集，一个是 identity，一个是alpaca_en_demo。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件

参数	示例值	参数说明
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】自定义数据集dataset_info.json配置文件**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

步骤3 是否选择加速深度学习训练框架Deepspeed，可参考**表4-371**选择不同的框架

- 是，选用ZeRO (Zero Redundancy Optimizer)优化器
  - ZeRO-0，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z0_config.json
    ```
  - ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
    ```
    deepspeed: examples/deepspeed/ds_z1_config.json
    ```
  - ZeRO-2，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z2_config.json
    ```
  - ZeRO-3，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z3_config.json
    ```
- 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。

步骤4 是否使用固定句长

- 是，配置以下参数
  ```
  packing: true
  ```
- 否，默认使用动态句长，**注释**掉packing参数。

步骤5 选用数据精度格式，以下参数二选一。

- bf16，配置以下参数
  ```
  bf16: true
  ```
- fp16，配置以下参数
  ```
  fp16: true
  ```

步骤6 是否使用自定义数据集

- 是，参考**准备数据（可选）**后，填写自定义注册后数据集前缀名称及数据集**绝对路径**，参考**表4-369**dataset_dir行，如demo.json数据集前缀则为demo
  ```
  dataset: demo
  dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
  ```
- 否，使用代码包自带数据集，配置参数如
  ```
  dataset: identity,alpaca_en_demo
  ```

步骤7 如需其他配置参数，可参考**表4-370**按照实际需求修改

----结束

Step3 启动训练脚本

说明

- **启动训练前**需**修改**启动训练脚本**demo.sh 内容**。具体请参考**•修改启动脚本**。
- 对于**falcon-11B**训练任务开始前，需**手动替换tokenizer**中的config.json，具体请参见**falcon-11B模型**。

修改完yaml配置文件后，启动训练脚本；模型不同最少npu卡数不同，npu卡数建议值可参考**模型NPU卡数取值表**。

- **修改启动脚本**

进入代码目录{work_dir}/llm_train/LLaMAFactory 下**修改启动脚本**，其中{work_dir}为容器挂载路径；修改demo.sh 最后一行代码：

将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，例如将以下命令：

```
FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
```

修改为：

```
FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
```

- **多机启动**

多台机器执行训练启动命令如下。

```
多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
#第一台节点
sh demo.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh demo.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh demo.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh demo.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

- **单机启动**

一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh demo.sh localhost 1 0
```

单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定挂载到容器里面的卡的索引，使用执行命令如下：

```
ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
```

其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务

- **训练成功标志**

“******* train metrics *******”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：指令微调训练常见问题**解决

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：指令微调训练常见问题**问题3小节。

3、训练过程中报"ModuleNotFoundError: No module named 'multipart'"关键字异常，可更新python-multipart为0.0.12版本，具体请参考**问题4："No module named 'multipart'"报错：**

4.39.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 4-690 打印训练日志

```
  0%|          | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|▏         | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▎         | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍         | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌         | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋         | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊         | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉         | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█         | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏        | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▍        | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▍        | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▌        | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▋        | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▊        | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|██        | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表4-369**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

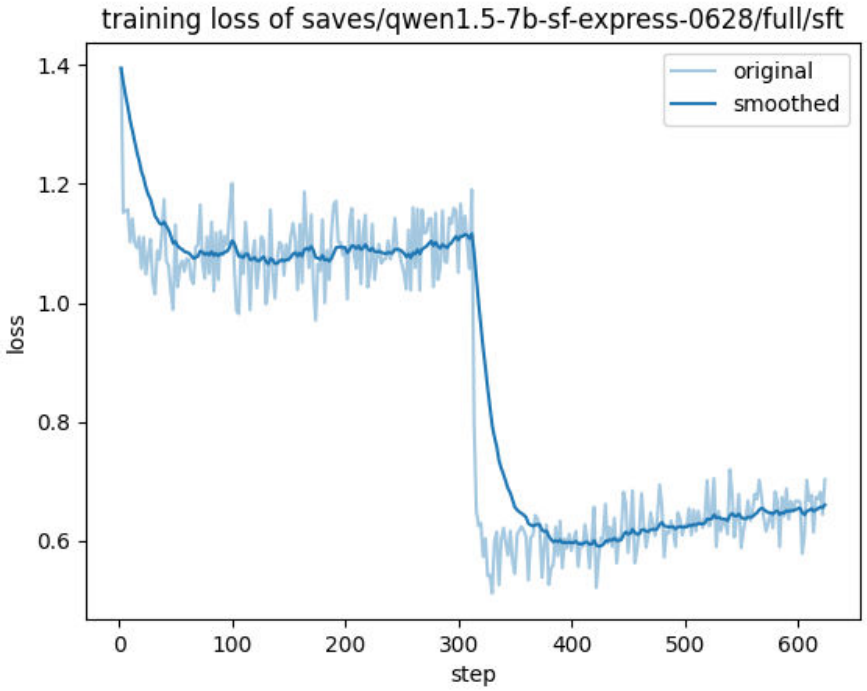
- 吞吐量（tokens/s/p）：可通过**表4-369**表格中**output_dir**参数值路径下的**train_results.json**查看性能。吞吐计算公式为"num_input_tokens_seen / train_runtime / 训练卡数"。相关参数可查看**表4-369**。

- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表4-369**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-691**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

  图 4-691 Loss 收敛情况（示意图）



4.39.5 训练脚本说明

4.39.5.1 yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml样例配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 4-370 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。

参数	示例值	参数说明
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
packing	true	**可选项**。当选用**静态数据长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数据长度**则**去掉**此参数。
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示训练类型。可选择值：[pt、sf、rm、ppo]，pt代表预训练，sft代表指令监督微调，rm代表奖励模型训练，ppo代表PPO训练。
finetuning_type	full	用于指定微调的类型，可选择值【full、lora】如果设置为"full"，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
dataset	identity,alpaca_en_demo	指定用于训练的数据集，数据集都放置在此处为identity，alpaca_en_demo表示使用了两个数据集，一个是identity，一个是alpaca_en_demo。如选用定义数据请参考**准备数据（可选）**
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练,模板选择可参照**表4-364**中的**template列**
max_samples	1000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。

参数	示例值	参数说明
per_device_train_batch_size	1	**必须修改**，指定每个设备的训练批次大小。
gradient_accumulation_steps	8	指定梯度累积的步数,这可以增加批次大小而不增加内存消耗。
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
save_steps	500	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	1.0e-5	指定学习率

sft_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: full
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
template: qwen
cutoff_len: 4096
packing: true
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
```

```
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
fp16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

lora_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: lora
lora_target: all
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
template: qwen
cutoff_len: 4096
packing: true
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
fp16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
```

```
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
    "round_robin_gradients": true
  }
}
```

4.39.5.2 各个模型深度学习训练加速框架的选择

1. LlamaFactory框架使用两种训练框架：
   - DeepSpeed和Accelerate都是针对深度学习训练加速的工具，但是它们的实现方式和应用场景有所不同。
     - DeepSpeed是一种深度学习加速框架，主要针对大规模模型和大规模数据集的训练。DeepSpeed的核心思想是在单个GPU上实现大规模模型并行训练，从而提高训练速度。DeepSpeed提供了一系列的优化技术，如ZeRO内存优化、分布式训练等，可以帮助用户更好地利用多个GPU进行训练
     - Accelerate是一种深度学习加速框架，主要针对分布式训练场景。Accelerate的核心思想是通过模型并行和数据并行来实现分布式训练，从而提高训练速度。Accelerate提供了一系列的优化技术，如模型切分、梯度累积等，可以帮助用户更好地利用多个节点进行训练。
2. 各个模型选用加速框架

表 4-371 模型加速框架建议表

序号	模型参数量	文本序列长度	优化工具(Deepspeed&Accelerator)
0	小于4B	cutoff_len=4096	Deepspeed-ZeRO-0
		cutoff_len=8192	Deepspeed-ZeRO-0
1	小于7B	cutoff_len=4096	Deepspeed-ZeRO-1
		cutoff_len=8192	Deepspeed-ZeRO-1
2	7B至13B	cutoff_len=4096	Deepspeed-ZeRO-2
		cutoff_len=8192	Deepspeed-ZeRO-2
3	14B-72B	cutoff_len=4096	Deepspeed-ZeRO-3
		cutoff_len=8192	Deepspeed-ZeRO-3

说明

以上为建议值，上述参数值仅供参考，如需配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器用户可自行选用配置。

4.39.5.3 模型 NPU 卡数取值表

不同模型推荐的训练参数和计算规格要求如表4-372所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推

表 4-372 模型 NPU 卡数取值表

支持模型	支持模型参数量	文本序列长度	训练类型	Zero并行	规格与节点数
llama 3	70B	cutoff_len=4096	lora	per_device_train_batch_size=1	2*节点 & 8*Ascend
			sft	per_device_train_batch_size=1	8*节点 & 8*Ascend
		cutoff_len=8192	lora	per_device_train_batch_size=1	2*节点 & 8*Ascend
			sft	per_device_train_batch_size=1	8*节点 & 8*Ascend
	8B	cutoff_len=4096/8192	lora sft	per_device_train_batch_size=1	1*节点 & 1*Ascend 1*节点 & 4*Ascend
Qwen 2	72B	cutoff_len=4096	lora sft	per_device_train_batch_size=1	2*节点 & 8*Ascend 4*节点 & 8*Ascend
		cutoff_len=8192	lora sft	per_device_train_batch_size=1	2*节点 & 8*Ascend 8*节点 & 8*Ascend
	7B	cutoff_len=4096	lora/sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=8192	lora/sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
	0.5/1.5B	*cutoff_len=4096/8192*	lora/sft	per_device_train_batch_size=1	1*节点 & 1*Ascend
Qwen 1.5	0.5B/1.8B	cutoff_len=4096/8192	lora/sft	per_device_train_batch_size=1	1*节点 & 1*Ascend
	4B	*cutoff_len=4096/8192*	sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend

支持模型	支持模型参数量	文本序列长度	训练类型	Zero并行	规格与节点数
	7B	cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
		cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
	14B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
falcon 2	11B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
Yi	6B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
	34B	cutoff_len=4096	sft lora	per_device_train_batch_size=1	2*节点 & 8*Ascend 1*节点 & 2*Ascend
		cutoff_len=8192	sft lora	per_device_train_batch_size=1	2*节点 & 8*Ascend 1*节点 & 4*Ascend

4.39.5.4 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

4.39.6 附录：指令微调训练常见问题

问题 1：在训练过程中遇到 NPU out of memory

解决方法：

步骤1 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。

步骤2 替换深度学习训练加速的工具或增加zero等级，可参考**各个模型深度学习训练加速框架的选择**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。

- - ZeRO-0 数据分布到不同的NPU
- - ZeRO-1 Optimizer States分布到不同的NPU
- - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
- - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU

步骤3 增加卡数重新训练，未解决找相关人员定位。

----结束

问题 2：访问容器目录时提示 Permission denied

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题 3：训练过程报错：ImportError: This modeling file requires the following packages that were not found in your environment: flash_attn

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

```
177         except ImportError:
178             missing_packages.append(imp)
179
180 #    if len(missing_packages) > 0:
181 #        raise ImportError(
182 #            "This modeling file requires the following packages that were not found in your environment: "
183 #            f"{', '.join(missing_packages)}. Run `pip install {' '.join(missing_packages)}`"
184 #        )
185
186     return get_relative_imports(filename)
187
188
189 def get_class_in_module(class_name: str, module_path: Union[str, os.PathLike]) -> typing.Type:
```

问题 4：训练过程中报"ModuleNotFoundError: No module named 'multipart'"报错：

截图如下：


```
    from .webui.interface import run_web_demo, run_web_ui
  File "/home/ma-user/LLaMAFactory/LLaMA-Factory/src/llamafactory/webui/interface.py"
    from .common import save_config
  File "/home/ma-user/LLaMAFactory/LLaMA-Factory/src/llamafactory/webui/common.py", l
    import gradio as gr
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    import gradio._simple_templates
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from .simpledropdown import SimpleDropdown
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio.components.base import FormComponent
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio.components.annotated_image import AnnotatedImage
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio.components.base import Component
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio.blocks import Block, BlockContext
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio import (
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from gradio.routes import App  # HACK: to avoid circular import # noqa: F401
  File "/home/ma-user/anaconda3/envs/PyTorch-2.2.0/lib/python3.10/site-packages/gradi
    from multipart.multipart import parse_options_header
ModuleNotFoundError: No module named 'multipart'
```

解决措施：可更新python-multipart为0.0.12版本，具体步骤如下：

- 启动训练任务前更新python-multipart版本：

```
pip install python-multipart==0.0.12
```

4.40 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.907）

4.40.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 支持FP16和BF16数据类型推理。
- Server驱动版本要求23.0.6。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-373 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-374**所示。

表 4-374 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-375**所示。

表 4-375 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
13	deepseek-coder-33b-instruct	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
27	qwen2-7b	√	√	x	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
40	llama3.1-8b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
        ├── build.sh               # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
        ├── Dockerfile             # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
  ├──llm_tools          # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
    ├──AutoAWQ             # W4A16量化工具
      ├──convert_awq_to_npu.py   # awq权重转换脚本
      ├──quantize.py   # 昇腾适配的量化转换脚本
      ├──build.sh      # 安装量化模块的脚本
    ├──llm_evaluation    # 推理评测代码包
      ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
      ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
```

```
├──install.sh            #安装opencompass脚本
├──vllm_api.py           #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型（PyTorch）基于Lite Server训练指导**。

4.40.2 部署推理服务

本章节介绍如何使用vLLM 0.5.0框架部署并启动推理服务。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数，用来确认对应卡数已经挂载
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-373**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip到主机中，包获取路径请参见**表4-374**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-375**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

3. 权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：
```
df -h
```

Step4 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name}
```

参数说明：

- ${base_image}为基础镜像地址。
- ${image_name}为推理镜像名称，可自行指定。

运行完后，会生成推理所需镜像。

Step5 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。docker启动失败会有对应的error提示，启动成功会有对应的docker id生成，并且不会报错。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，即第四步中生成的新镜像id，在宿主机上可通过docker images查询得到。

Step6 启动推理服务

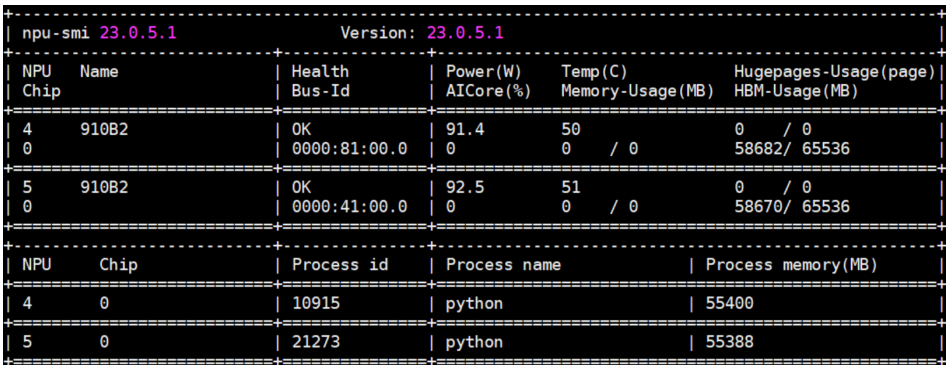
1. 进入容器。
```
docker exec -it -u ma-user ${container-name} /bin/bash
```
2. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。
```
npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

驱动版本要求是23.0.6。如果不符合要求请参考**安装固件和驱动**章节升级驱动。启动后容器默认端口是8080。
3. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-692 查询结果



4. 配置环境变量。
```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一
```



```
次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

5. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**或**使用SmoothQuant量化**章节对模型做量化处理。
6. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**

     在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   - **方式二：通过vLLM服务API接口启动服务**

     在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   推理服务基础参数说明如下：

   - --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
   - --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
   - --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-

model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。

- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。使用不同的dtype会影响模型精度。如果使用开源权重，建议不指定dtype，使用开源权重默认的dtype。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**3**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为None，举例：参数可以设置为0.0.0.0。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。如果模型长度>8192，不支持开启prefix-caching特性，否则会导致推理服务不可用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，如果未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。如果未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。如果未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，如果不使用该功能，则无需配置。注意：如果使用投机推理功能，必须开启此参数。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-376**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${container_model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

  下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

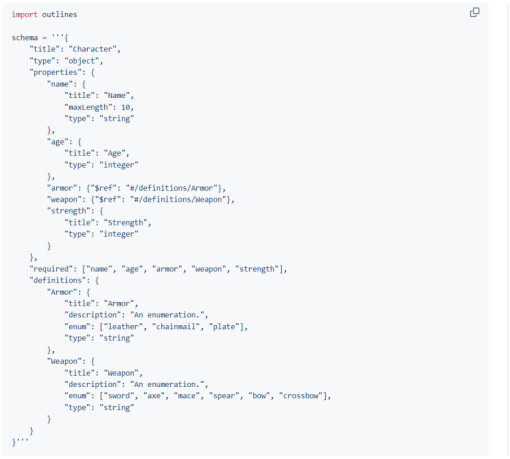
```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-376 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-693** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

4.40.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py    # 工具函数集
├── benchmark.py          # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试还不支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step4 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，切换一个conda环境。
   ```
   cd benchmark_tools
   conda activate python-3.9.10
   ```
3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host ${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口8080。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-694 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**

- Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

方法二：使用generate_dataset.py脚本生成数据集方法：

客户通过业务数据，在generate_dataset.py脚本，指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```

generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 进入benchmark_tools目录下，切换一个conda环境。

```
cd benchmark_tools
conda activate python-3.9.10
```

3. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
python benchmark_serving.py --backend vllm --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。

- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

4. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-695 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.40.4 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
```

3. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤4进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

4. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

5. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

6. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 4-377 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

7. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远(小于10）认为分数有效。

4.40.5 推理模型量化

4.40.5.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-375**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：per-group

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、在容器中使用ma-user用户运行以下命令下载并安装AutoAWQ源码。

```
bash build.sh
```

2、运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.40.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-375**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。

- --scale-input：量化系数输入路径，如果之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，如果指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，如果指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step6 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

4.40.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-375**。

Step1 使用 tensorRT 量化工具进行模型量化，必须在 GPU 环境

使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在--output_dir下生成kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：


```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.40.6 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-378 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16
18	qwen1.5-110b	--		8	128
19	qwen2-0.5b	1	128	1	256

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
38	falcon-11b	1	8	1	64

4.40.7 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

4.41 主流开源大模型基于 Standard+OBS 适配 PyTorch NPU 训练指导（6.3.907）

4.41.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表4-379所示。

表 4-379 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 4-696 操作流程图

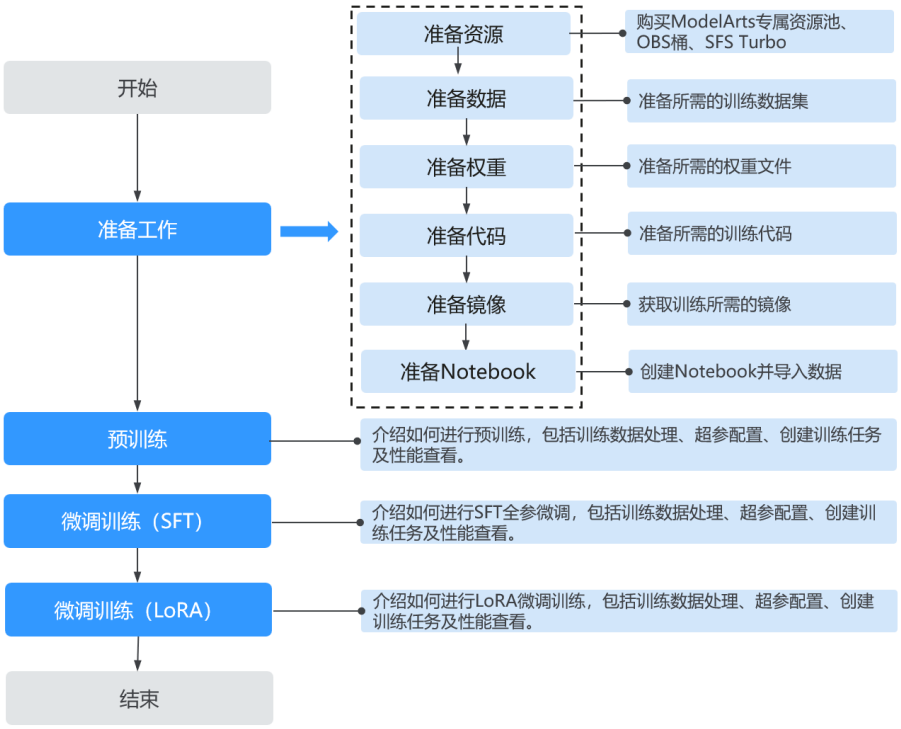


表 4-380 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.41.2 准备工作

4.41.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-386**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

4.41.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

```
{
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
}
```

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.41.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-379**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.41.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如表4-381所示。

表 4-381 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.907代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
                |——llama2          # llama2系列模型执行脚本的文件夹
                |——llama3          # llama3系列模型执行脚本的文件夹
                |——qwen            # Qwen系列模型执行脚本的文件夹
                |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                |—— ...
                |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |—— install.sh       # 环境部署脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

修改代码

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后。在上传代码前，需要对解压后的训练脚本代码进行修改。具体文件为：llm_train/AscendSpeed/scripts/obs_pipeline.sh，具体修改代码内容以及位置，如下所示。

1. 训练作业中存在2个代码目录，一个是从OBS上传到ModelArts Standard训练容器中的代码目录**OBS_CODE_DIR**，一个是后续构建新镜像步骤**ECS中构建新镜像（二选一）**中镜像的代码目录**CODE_DIR**。修改代码如图4-697。

   图 4-697 修改区分训练作业中 2 个代码目录

```
OBS_CODE_DIR=`pwd`
CODE_DIR=/home/ma-user/AscendSpeed

ModelLink_PATH=$CODE_DIR/ModelLink
MindSpeed_PATH=$CODE_DIR/MindSpeed
Scripts_PATH=$OBS_CODE_DIR/scripts
ASCENDCLOUD_PATCH_PATH=$CODE_DIR/ascendcloud_patch
INPUT_PROCESSED_DIR=${INPUT_PROCESSED_DIR:=$ROOT_DIR/processed_for_input/$MODEL_NAME}
OUTPUT_SAVE_DIR=${OUTPUT_SAVE_DIR:=$ROOT_DIR/saved_dir_for_output/${MODEL_NAME}_${RUN_TYPE}_${SEQ_LEN}}
```

2. 使用环境变量**SAVE_PATH**重新覆盖权重文件保存路径，作为最终的权重保存路径。修改代码如**图4-698**。

图 4-698 修改权重保存路径

```
# PTD args
cd $ASCENDCLOUD_PATCH_PATH
MODEL_TYPE=$MODEL_NAME
SAVE_PATH=$OUTPUT_SAVE_DIR
MODEL_PATH=${MODEL_PATH:=$CONVERT_MODEL_PATH}
CKPT_SAVE_PATH=${SAVE_PATH}/saved_models/${RUN_TYPE}
mkdir -p ${SAVE_PATH}/logs
mkdir -p ${SAVE_PATH}/saved_models/${RUN_TYPE}

SAVE_PATH=${SAVE_PATH}/saved_models/${RUN_TYPE}

if [[ $Instruction_Data =~ ${DATA_TYPE} ]]; then
    run_type_args="
        --is-instruction-dataset \
    "
fi
```

3. 多机训练场景下，需要将**CODE_DIR**修改为**OBS_CODE_DIR**目录，则可以使用scripts/tools/sync_with_obs.py工具将其它节点的权重文件同步上传到主节点。修改代码如**图4-699**。

图 4-699 多机同步权重文件

```
if [[ $MOUNT == "OBS" ]]; then
  shard=$((TP * PP))
  if [ "$shard" -gt 8 ]; then
    cd $OBS_CODE_DIR
    python scripts/tools/sync_with_obs.py --local_dir=$CKPT_SAVE_PATH
    CKPT_SAVE_PATH=${MA_JOB_DIR}/mg_weights
  else
    echo "skip sync with obs"
  fi
fi
sleep 5s
```

代码上传至 OBS

将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── models                                         #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
|── tokenizers                                     #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.41.2.5 准备镜像

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-382 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像（二选一）**、**ECS中构建新镜像（二选一）**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **使用基础镜像（二选一）**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像（二选一）**：在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。
  - 若用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

注意：训练作业的资源池以及ECS都需要联通外网，否则会安装和下载失败。

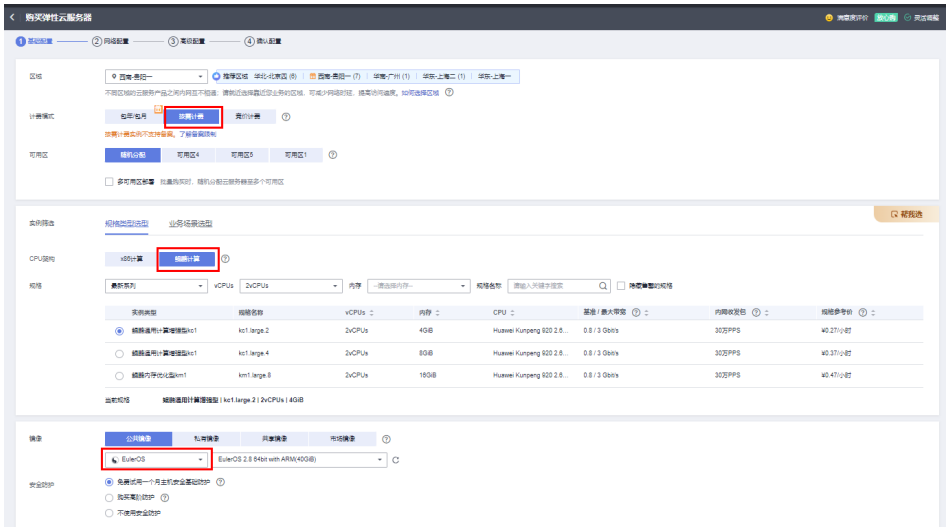
ECS 获取和上传基础镜像

步骤1 创建ECS。

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

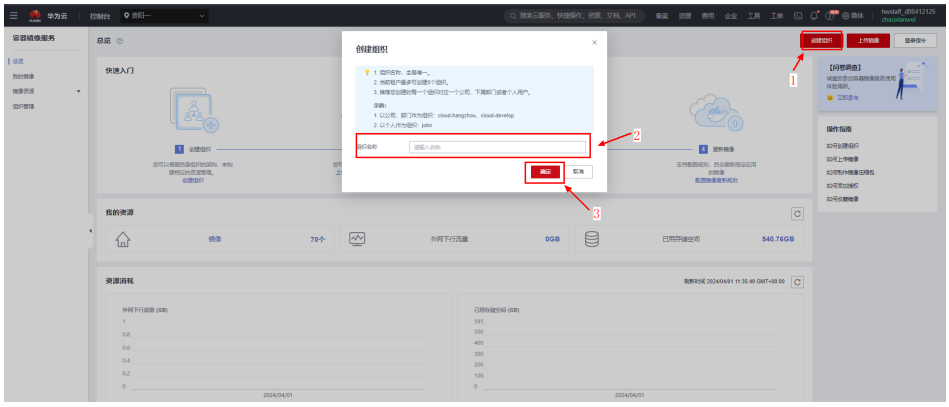
图 4-700 购买 ECS



步骤2 创建镜像组织。

在SWR服务页面创建镜像组织。

图 4-701 创建镜像组织



步骤3 安装Docker。

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤4 获取训练镜像。

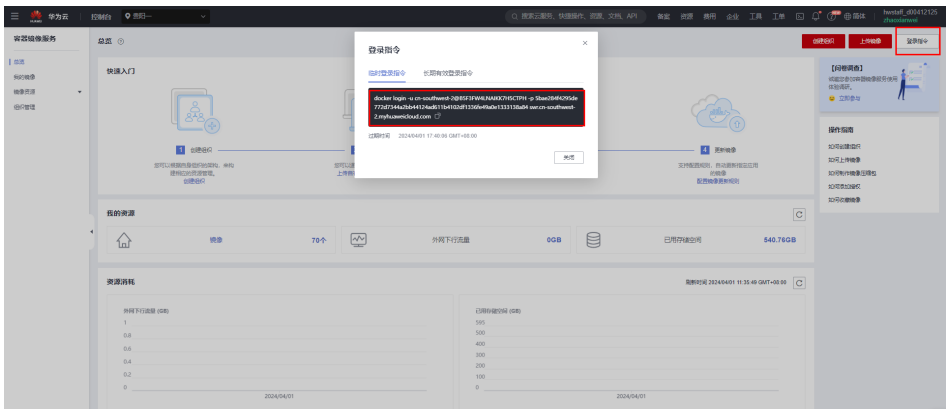
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-382**。

```
docker pull {image_url}
```

步骤5 在ECS中Docker登录。

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-702 复制登录指令



步骤6 修改并上传镜像。

1. 登录指令输入之后，使用下列示例命令：
   ```
   docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   参数说明：
   - <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
   - <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

   docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606
2. 上传镜像至镜像仓库。
   ```
   docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   示例：

   docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

----结束

使用基础镜像（二选一）

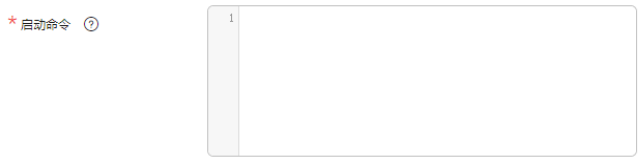
通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，若直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-703**中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。命令如下：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而 install.sh 则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-703 训练作业启动命令



ECS 中构建新镜像（二选一）

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

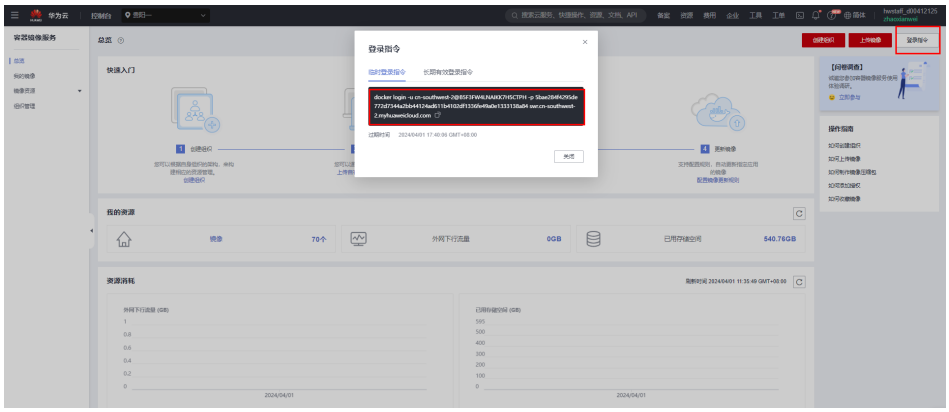
步骤1 获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-381**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.907-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   若无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤2 在ECS中Docker登录。

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-704 复制登录指令



步骤3 修改并上传镜像。

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：
   ```
   docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   参数说明：

   ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。

   <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

   <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group

   <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

----结束

4.41.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。

本案例中，若用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-705 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，若需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 4-706 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至云硬盘EVS）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

4.41.3 预训练

前提条件

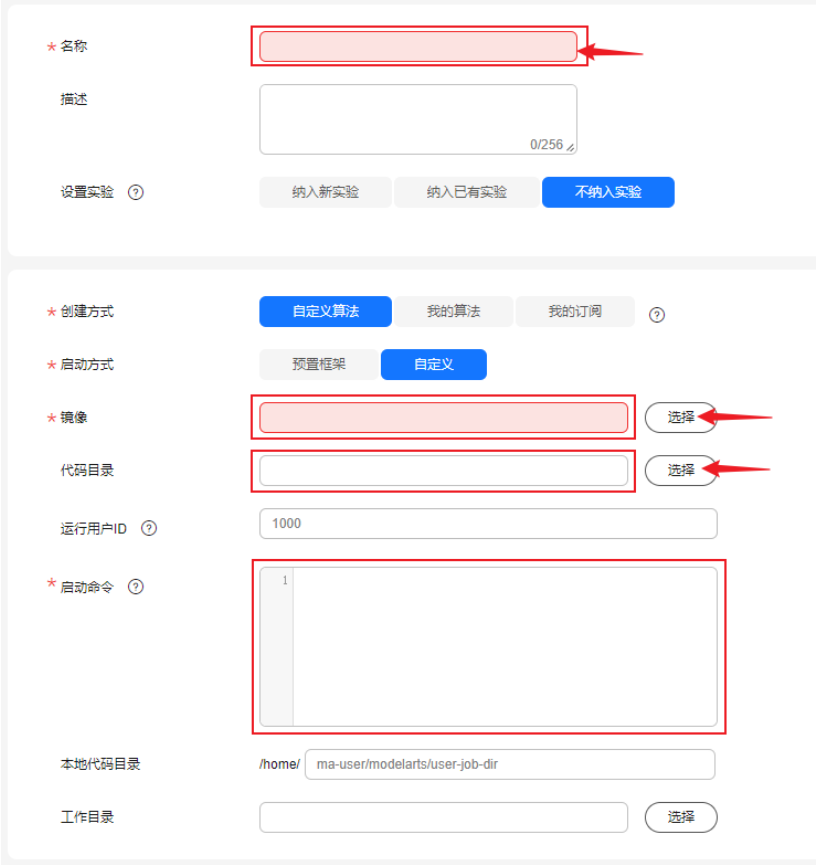
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 4-707 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-383**表格中的配置进行填写。

表 4-383 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-386**进行配置。

图 4-708 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

4.41.4 SFT 全参微调训练

前提条件

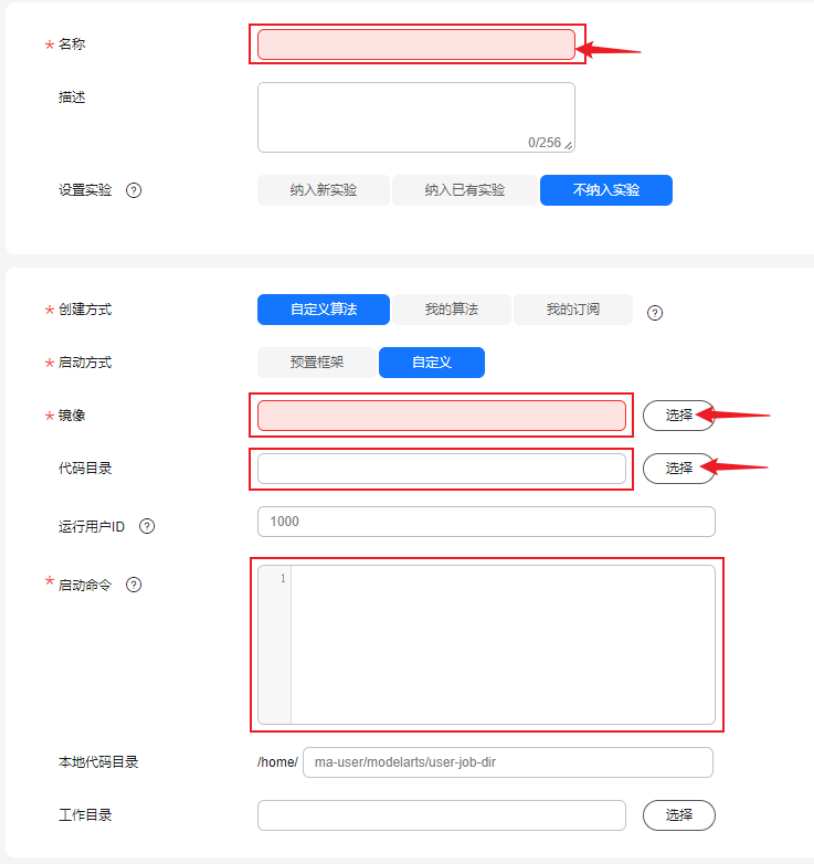
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 4-709 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-383**表格中的配置进行填写。

图 4-710 环境变量

表 4-384 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。

环境变量	示例值	参数说明
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-386**进行配置。

图 4-711 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

4.41.5 LoRA 微调训练

前提条件

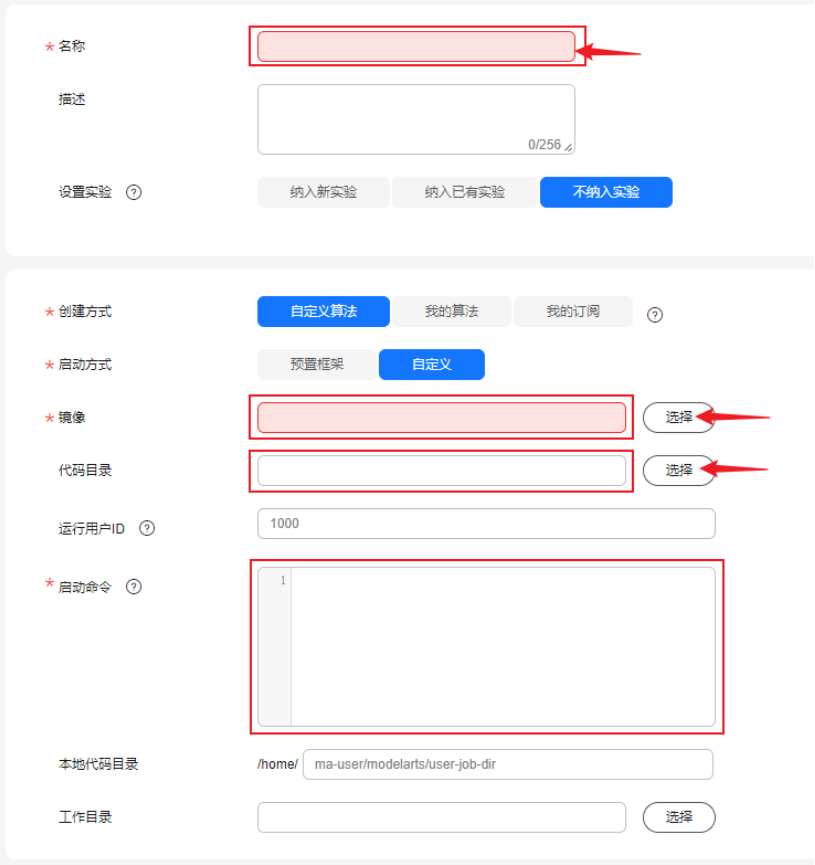
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 4-712 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表4-383**表格中的配置进行填写。

表 4-385 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
TRAIN_ITERS	100	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-386**进行配置。

图 4-713 选择资源池规格

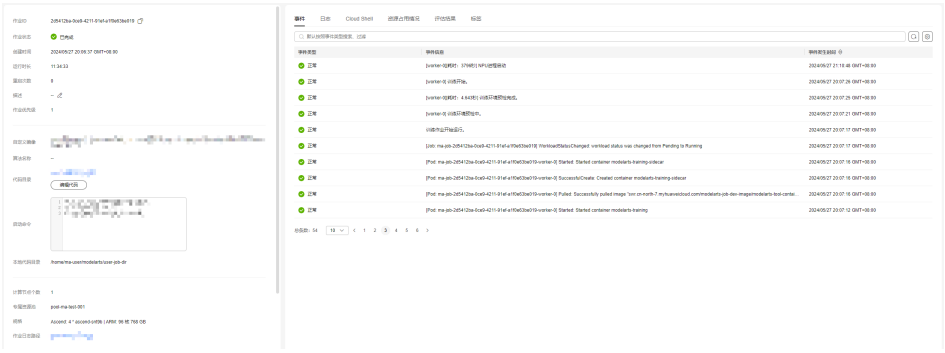
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

4.41.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

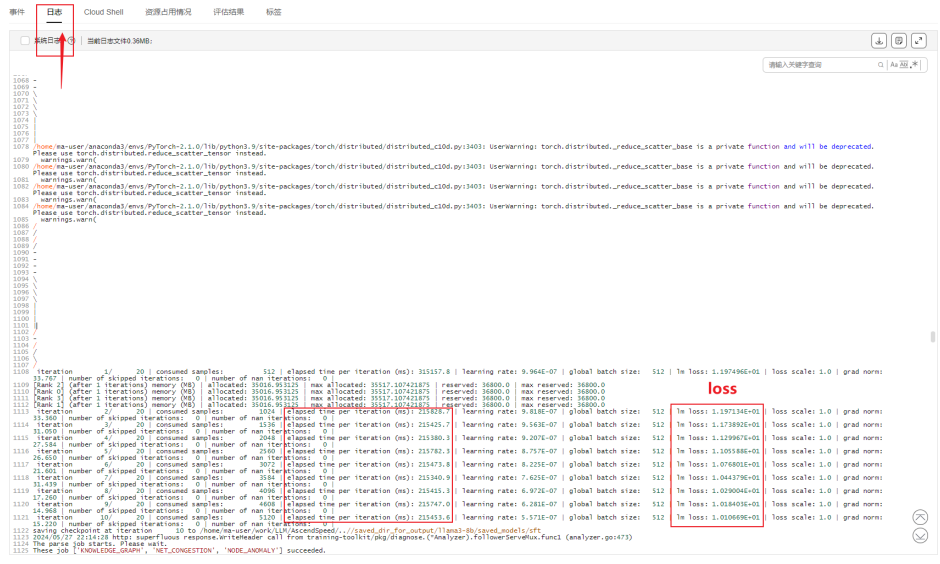
图 4-714 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-715 查看日志和性能



4.41.7 训练脚本说明

4.41.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

不同模型推荐的训练参数和计算规格要求如**表4-386**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-386 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

4.41.7.2 训练数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。

- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-387 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.41.7.3 训练权重转换说明

以 **llama2-13b** 举例，使用训练作业运行 **obs_pipeline.sh** 脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。

- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

若用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。若用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-388 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face

参数	示例	参数说明
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.41.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-716所示。

图 4-716 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-717 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-718 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-719 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
295
```

图 4-720 修改 ChatGLMv4-9B tokenizer 文件

```
        if needs_to_be_padded:
            difference = max_length - len(required_input)

            if "attention_mask" in encoded_inputs:
                encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
            if "position_ids" in encoded_inputs:
                encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
            encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference

        return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-721 修改 Qwen tokenizer 文件

```
from transformers.utils import logging

try:
    from einops import rearrange
except ImportError:
    rearrange = None
from torch import nn

SUPPORT_CUDA = torch.cuda.is_available()
SUPPORT_BF16 = SUPPORT_CUDA and True
SUPPORT_FP16 = SUPPORT_CUDA and True
SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2


from .configuration_qwen import QWenConfig
from .qwen_generation_utils import (
    HistoryType,
```

4.41.8 常见错误原因和解决方法

4.41.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-386**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

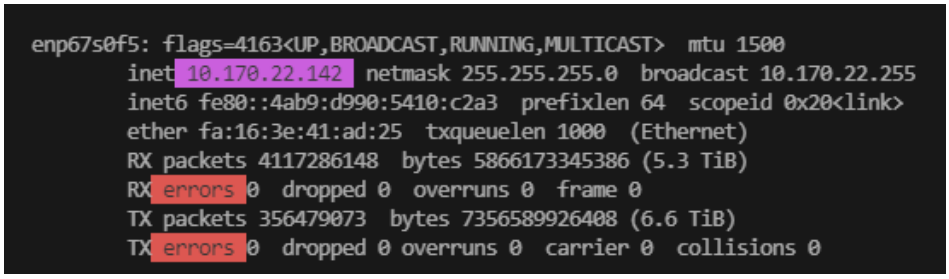
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.41.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-722 网卡名称错误



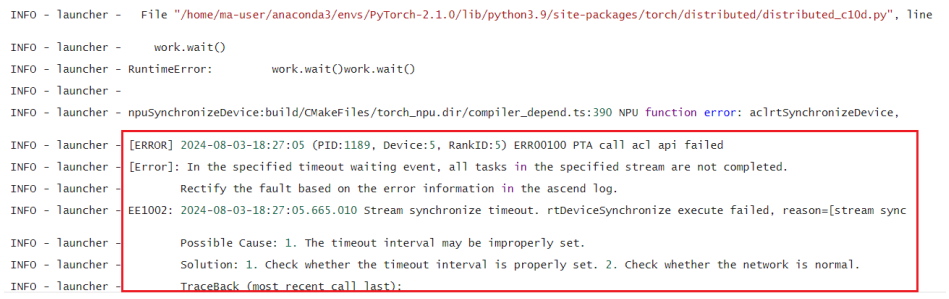

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.41.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-723 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.42 主流开源大模型基于 Standard+OBS+SFS 适配 PyTorch NPU 训练指导（6.3.907）

4.42.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅使用OBS的存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现数据灵活管理、高性能读取数据等。通过OBS上传训练所需的模型文件、训练数据等，再将OBS中的数据文件导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表4-389所示。

表 4-389 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

序号	支持模型	支持模型参数量	权重文件获取地址
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 4-724 操作流程图

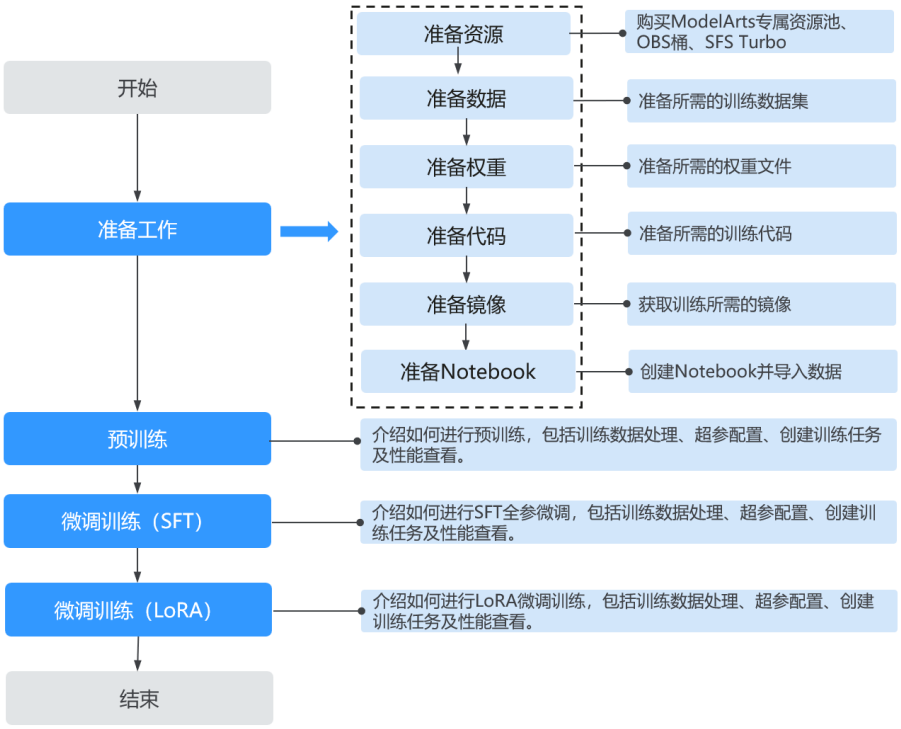


表 4-390 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.42.2 准备工作

4.42.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-397**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

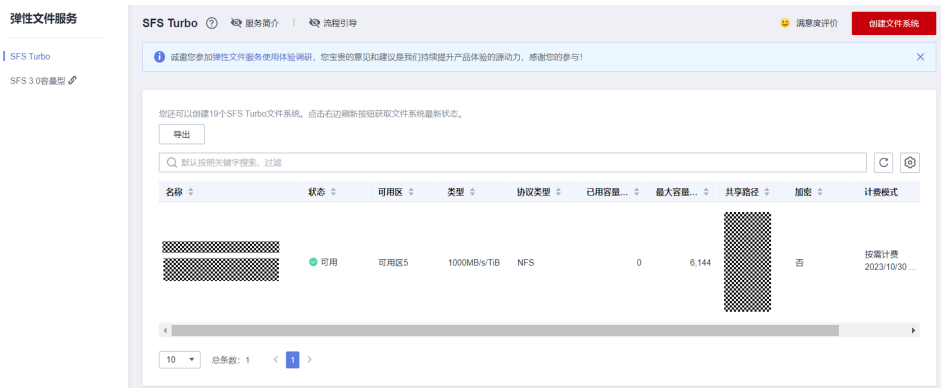
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-725 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

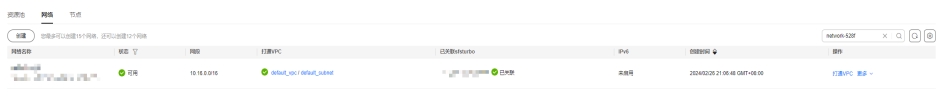
图 4-726 SFS 类型和容量选择

ModelArts 网络关联 SFS Turbo

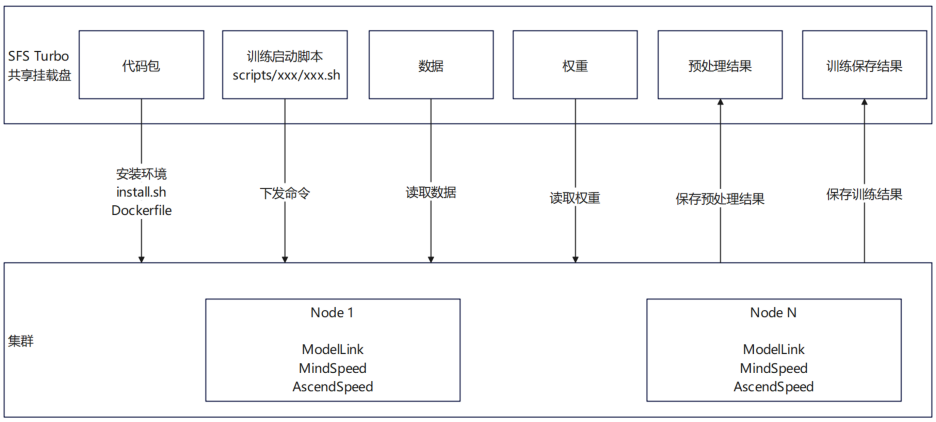
OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

若ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-727 ModelArts 网络关联 SFS Turbo



SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过SSH连接ECS将代码包上传至SFS Turbo中。
2. 在**表4-392**获取基础镜像，随后通过**镜像方案说明**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

4.42.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

请注意huggingface中的数据集具有如下this格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

```
{
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
}
```

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.42.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-389**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

4.42.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-391**所示。

表 4-391 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.907代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
              |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |——scripts/             # 训练需要的启动脚本
                   |——llama2          # llama2系列模型执行脚本的文件夹
                   |——llama3          # llama3系列模型执行脚本的文件夹
                   |——qwen            # Qwen系列模型执行脚本的文件夹
                   |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   |—— ...
                   |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   |—— install.sh       # 环境部署脚本
              |———src/                # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

代码上传至 OBS

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|——llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |—— AscendSpeed               # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |—— processed_for_input          #目录结构会自动生成，无需用户创建
          |—— ${model_name}             # 模型名称
                |—— data                # 预处理后数据
            |—— pretrain   # 预训练加载的数据
            |—— finetune   # 微调加载的数据
       |——converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |—— saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
          |—— ${model_name}             # 模型名称
```

```
                    |── logs              # 训练过程中日志（loss、吞吐性能）
                      |─── saved_models
                    |── lora              # lora微调输出权重
                    |── sft               # 增量训练输出权重
                    |── pretrain          # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── models                                         #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
|── tokenizers                                     #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.42.2.5 准备镜像

4.42.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-392 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**、**Notebook中构建新镜像**的方式（**三选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  若用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。
- **Notebook中构建新镜像方案：**首先需要ECS将基础镜像上传至SWR中。随后在Notebook环境中，通过运行scripts/install.sh文件会安装必要的依赖包以及下载Megatron-LM、MindSpeed、ModelLink源码。若Notebook环境挂载了SFS

Turbo，则源码文件会下载至SFS Turbo中。最后选择Notebook中“保存镜像”，则可以得到新的镜像环境。

若用户希望修改源码，则需要在Notebook环境中直接访问并编辑源码文件。

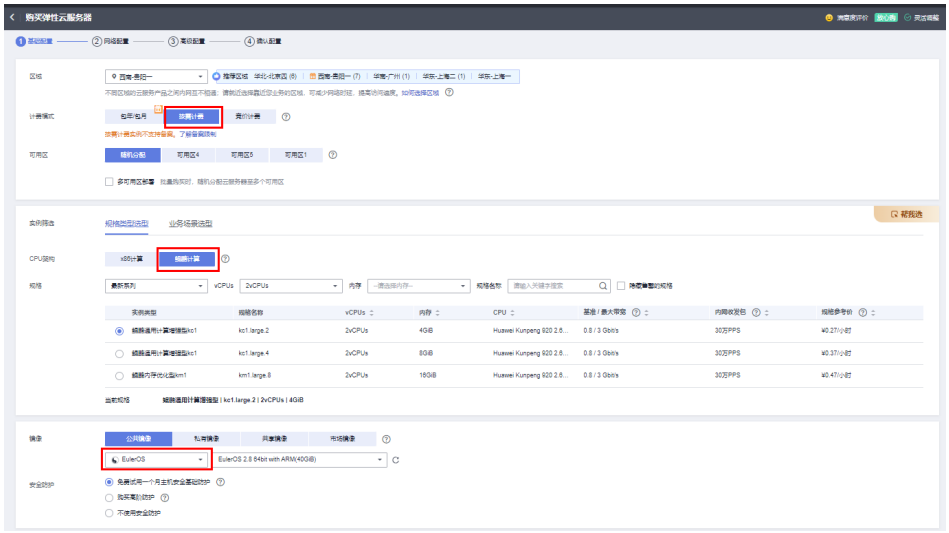
4.42.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考ECS文档购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

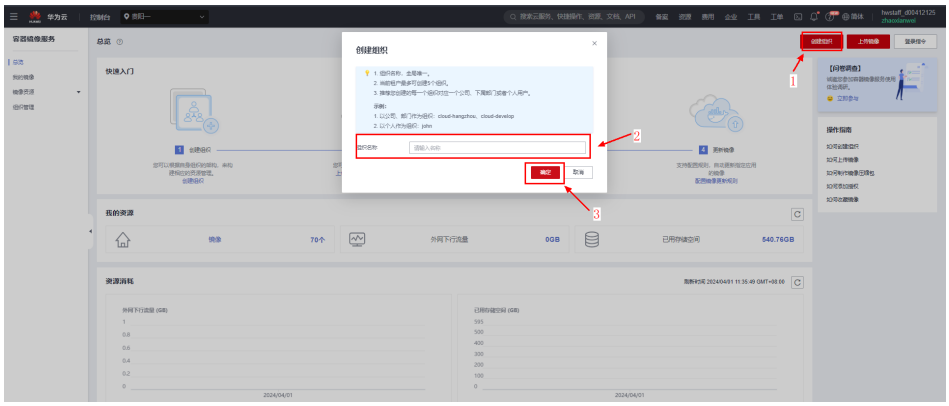
图 4-728 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-729 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

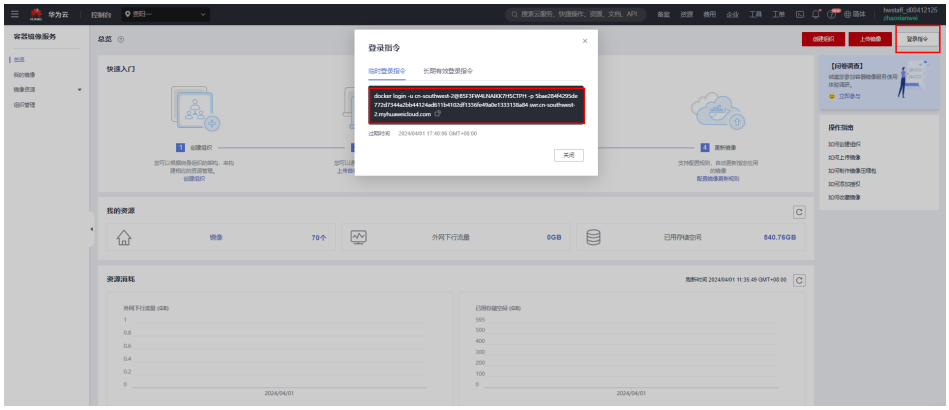
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-392**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-730 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

4.42.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

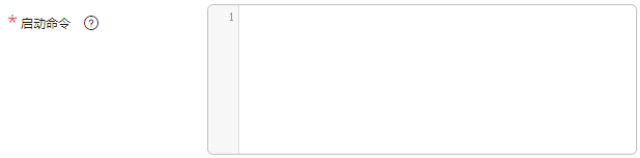
由于基础镜像内需要安装固定版本依赖包，若直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图4-731**中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而 install.sh 则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 4-731 训练作业启动命令



4.42.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-391**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.907-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网

```
docker build -t <镜像名称>:<版本名称> .
```

若无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

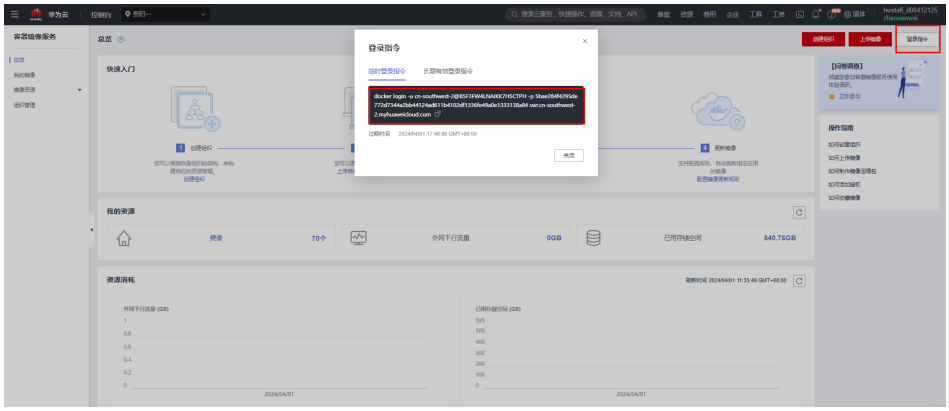
```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称> .
```

- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-732 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

```
docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/
pytorch_2_1_ascend:20240606
```

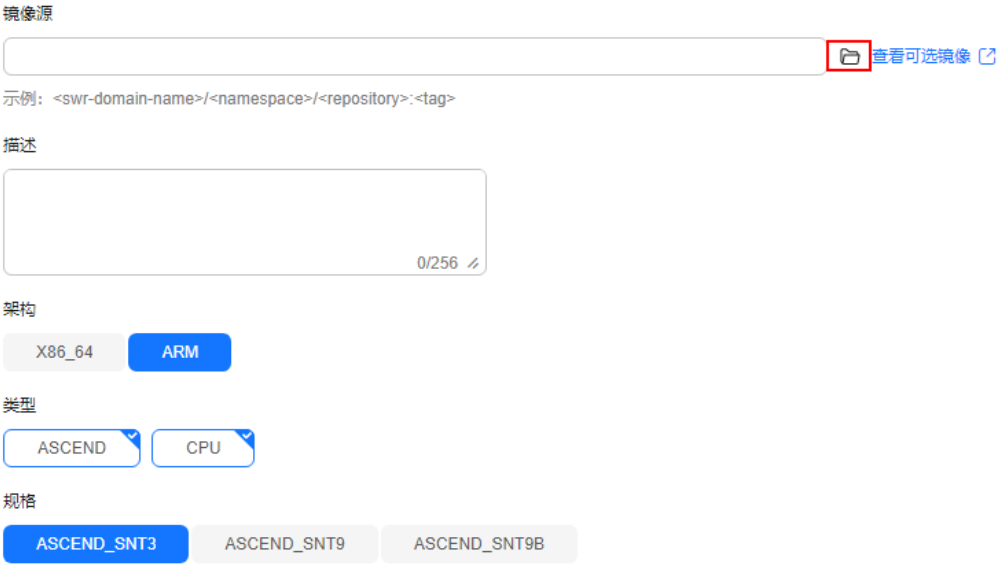
4.42.2.5.5 Notebook 中构建新镜像

ModelArts 中注册镜像

通过**ECS获取和上传基础镜像**将基础镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。镜像注册的操作步骤如下：

1. 登录ModelArts管理控制台，在左侧导航栏单击“镜像管理”。
2. 在“镜像管理”页面右上角，单击“注册镜像”。
3. 在“注册镜像”页面，选择已上传的镜像源，按需增加“描述”，“架构”选择“ARM”，“类型”选中“ASCEDN”和“CPU”，按需选择“规格”，然后单击“立即注册”。

图 4-733 配置镜像信息



Notebook 介绍

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

Step1 创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，资源规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-734 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例，子目录挂载可选择默认不填写。

如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 4-735 Notebook 中选择弹性文件服务

Step2 使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Step3 Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包。

通过运行install.sh脚本，会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
|——AscendCloud-LLM
  |——llm_train                # 模型训练代码包
      |——AscendSpeed          # 基于AscendSpeed的训练代码
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
          |——scripts/             # 训练需要的启动脚本
          |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
      |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
      |——MindSpeed/           # MindSpeed昇腾大模型加速库
      |——ModelLink/           # ModelLink端到端的大语言模型方案
          |——megatron/        # 注意：该文件夹从Megatron-LM中复制得到
          |——...
```

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

图 4-736 安装依赖包

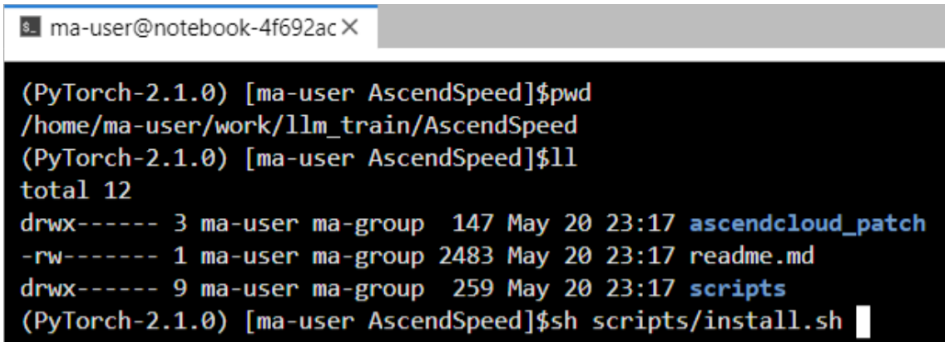


图 4-737 保存镜像

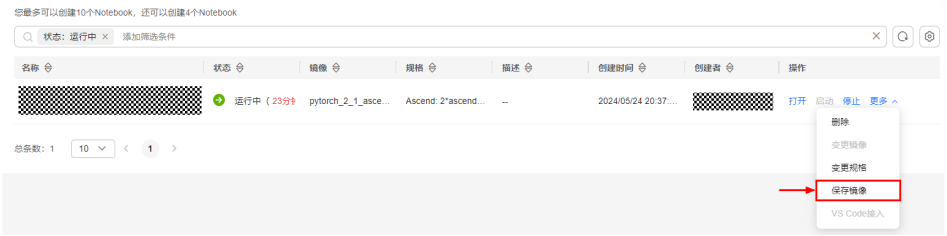


图 4-738 填写保存镜像相关参数

4.42.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如表4-393所示。其他超参均有默认值，可以参考表4-396按照实际需求修改。

表 4-393 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

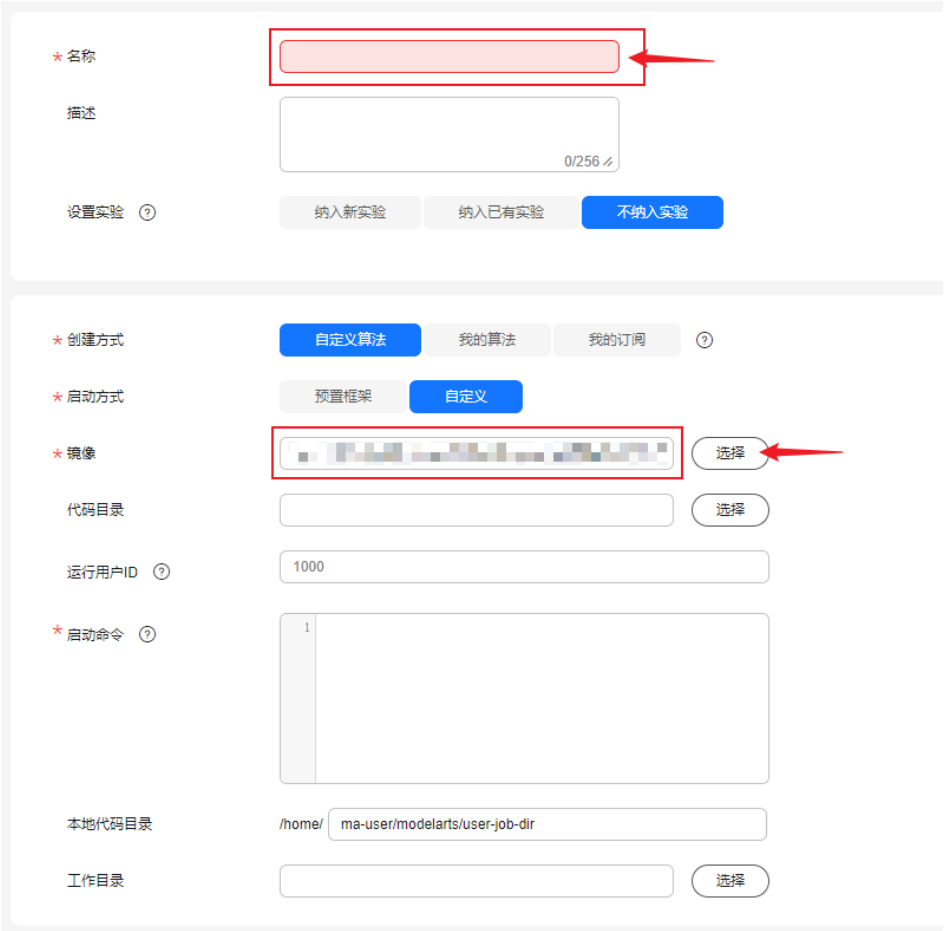
参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-739 选择镜像



若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

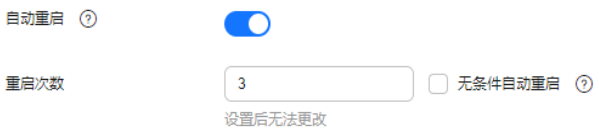
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-740 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

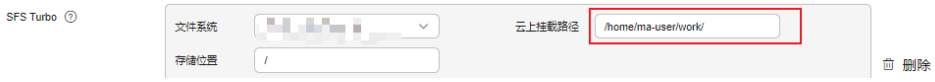
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-397**进行配置。

图 4-741 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 4-742 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.42.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表4-394所示。其他超参均有默认值，可以参考表4-396按照实际需求修改。

表 4-394 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-743 选择镜像

若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

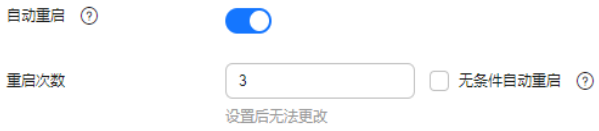
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-744 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-397**进行配置。

图 4-745 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 4-746 选择 SFS Turbo

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.42.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-395**所示。其他超参均有默认值，可以参考**表4-396**按照实际需求修改。

表 4-395 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/models/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/tokenizers/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/llm_train/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/llm_train/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。

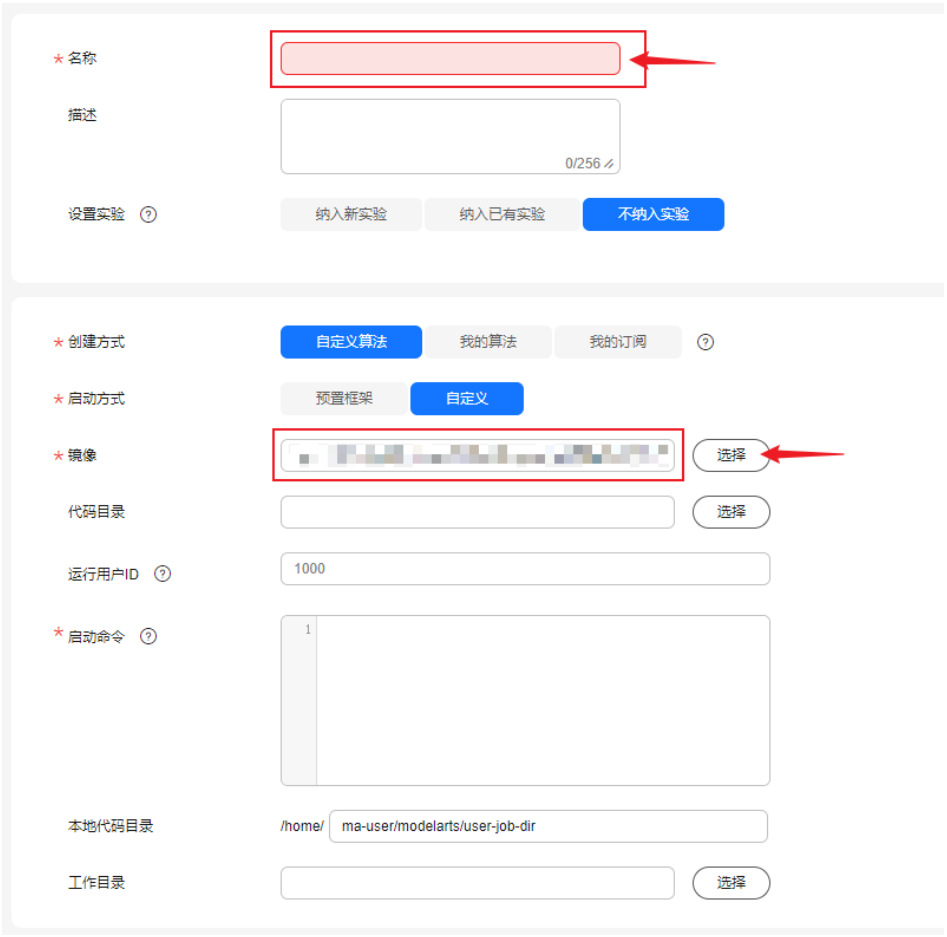
参数	示例值	参数说明
CKPT_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。
CONVERT_MG2HF	TRUE	训练完成的权重文件默认不会自动转换为Hugging Face格式权重。如果需要自动转换，则在运行脚本添加变量CONVERT_MG2HF并赋值TRUE。如果用户后续不需要自动转换，则在运行脚本中必须**删除**CONVERT_MG2HF变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-747 选择镜像



若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

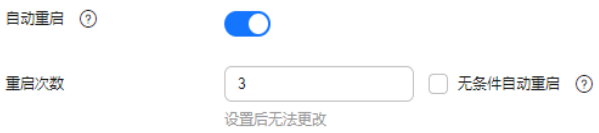
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-748 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint继续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置继续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

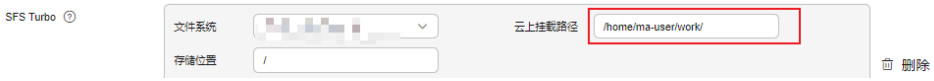
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-397**进行配置。

图 4-749 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 4-750 选择 SFS Turbo



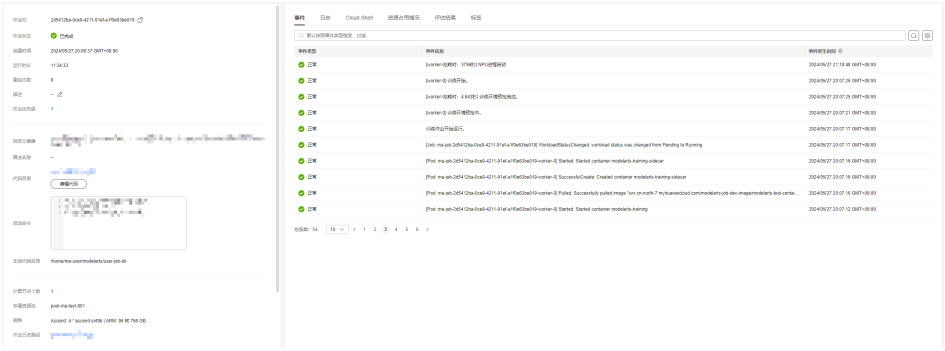
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.42.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

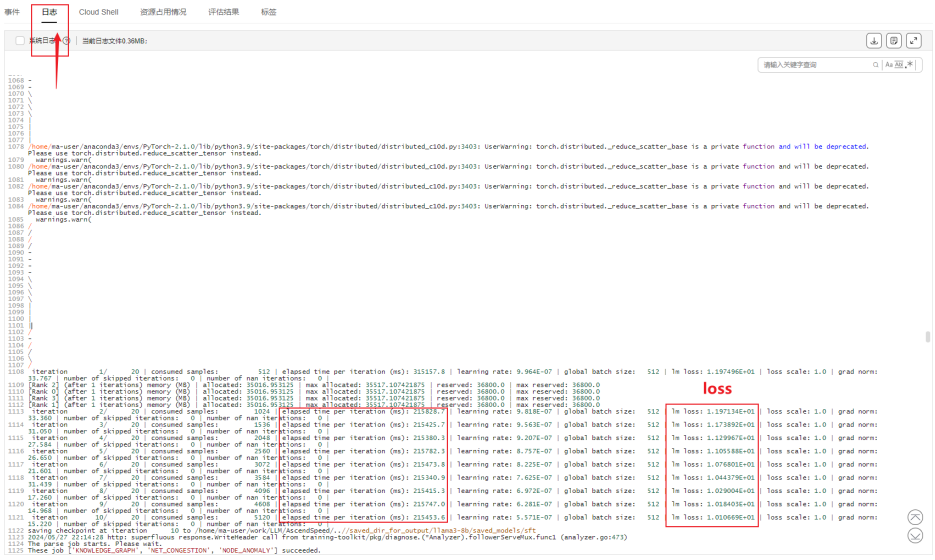
图 4-751 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-752 查看日志和性能



4.42.7 训练脚本说明

4.42.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**

环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-396 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。

参数	示例值	参数说明
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-397所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-397 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

4.42.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的文本数据集，用于预训练。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-398 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.42.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-399 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.42.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图4-753所示。

图 4-753 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-754 修改 ChatGLMv3-6B tokenizer 文件

```
295                 return_attention_mask:
296                     (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-755 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-756 修改 ChatGLMv4-9B tokenizer 文件

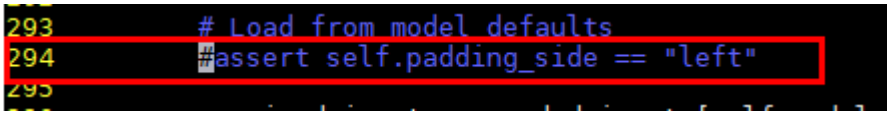
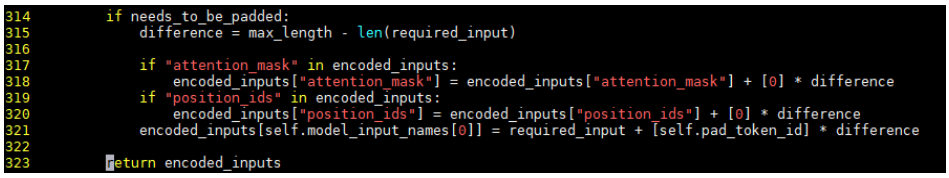


图 4-757 修改 ChatGLMv4-9B tokenizer 文件

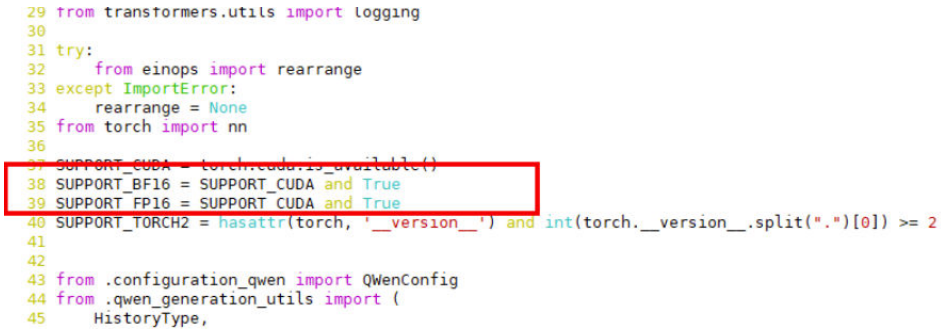


Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-758 修改 Qwen tokenizer 文件



4.42.8 常见错误原因和解决方法

4.42.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表4-397**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

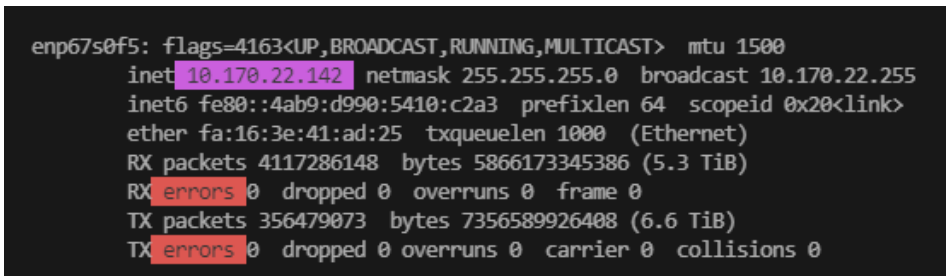
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

4.42.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 4-759 网卡名称错误



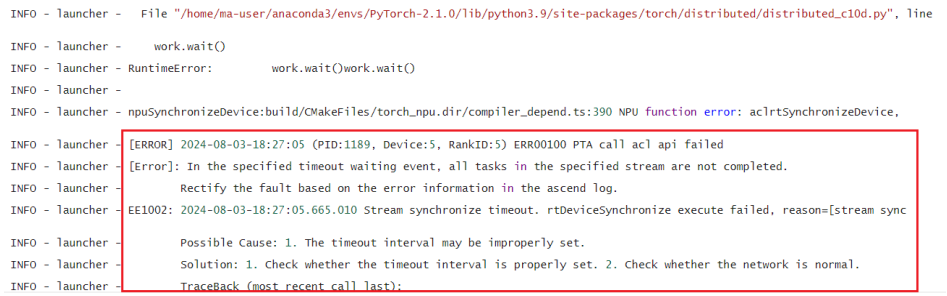

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

4.42.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 4-760 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

4.43 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.907）

4.43.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如**表4-400**所示。

表 4-400 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	https://huggingface.co/huggyllama/llama-7b
2	llama-13b	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
13	deepseek-coder-33b-instruct	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
27	qwen2-7b	√	√	x	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon-11b	√	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
40	llama3.1-8b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

操作流程

图 4-761 操作流程图

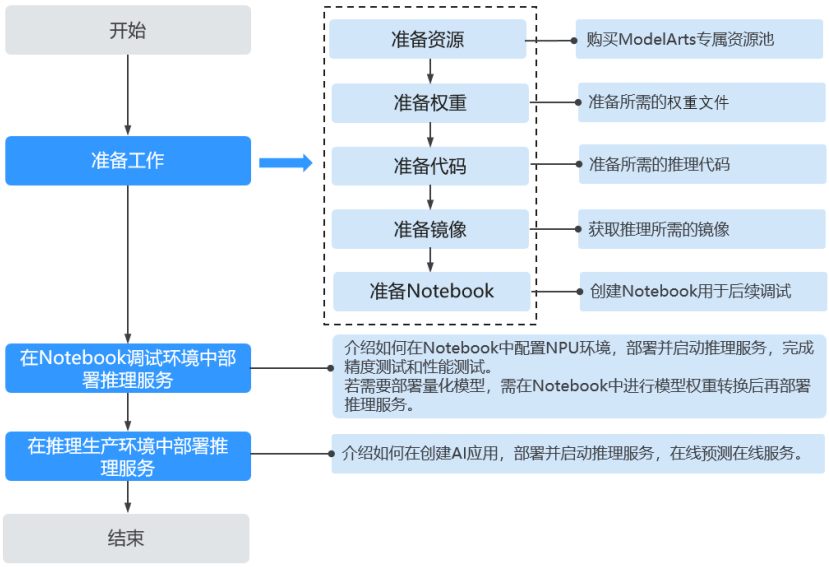


表 4-401 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.907-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 若需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何创建AI应用，部署模型并启动推理服务，在线预测服务。

4.43.2 准备工作

4.43.2.1 准备资源

创建专属资源池

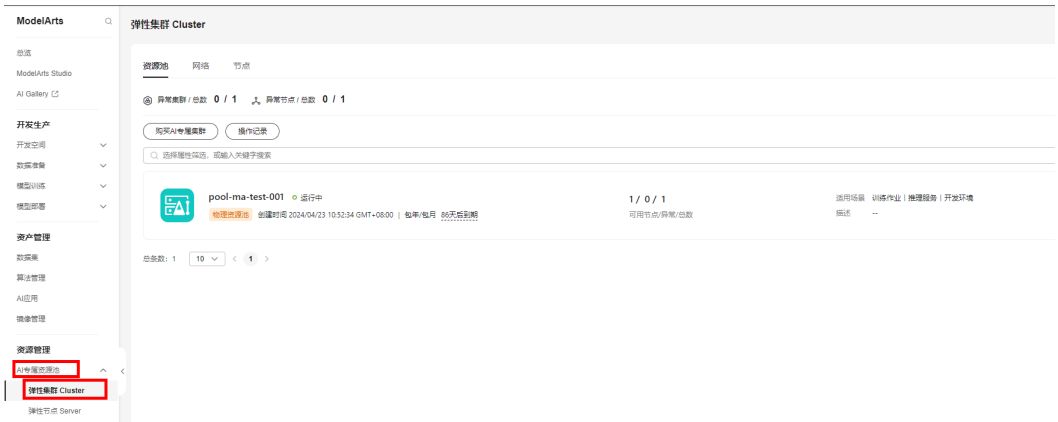
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 4-762 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 4-763 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.43.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.43.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-402**所示。

表 4-402 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
        ├── build.sh                # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
  ├──llm_tools          # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
    ├──AutoAWQ             # W4A16量化工具
        ├──convert_awq_to_npu.py   # awq权重转换脚本
        ├──quantize.py   # 昇腾适配的量化转换脚本
        ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

4.43.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-403 基础容器镜像地址

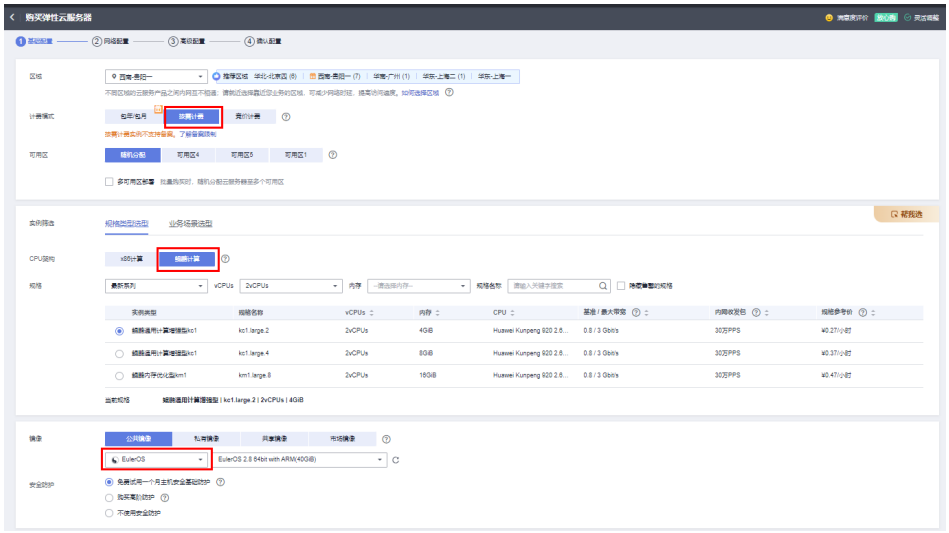
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-764 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-765 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-402**。

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网。

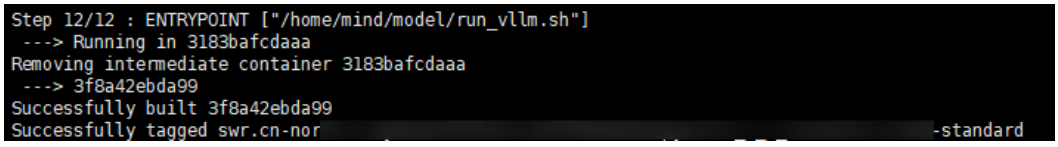
```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image} --image-name=${image_name} --specify-enrtypoint=True
```

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

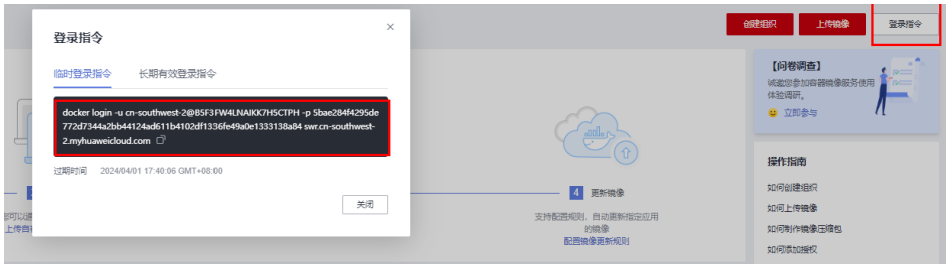
图 4-766 成功构建镜像



Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-767 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

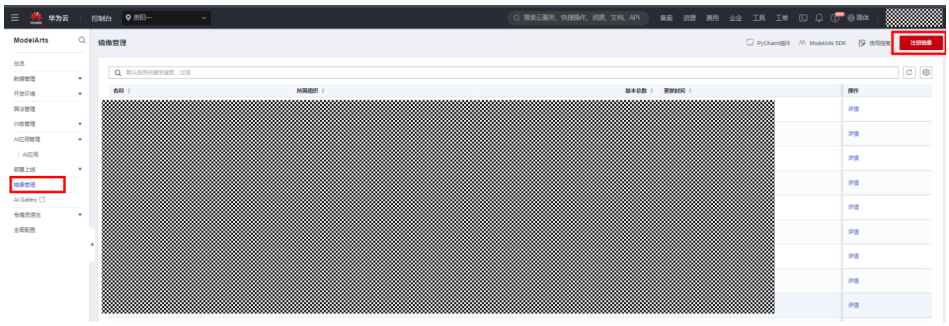
打印如下信息，表示上传镜像成功。

图 4-768 成功上传镜像

Step8 注册镜像

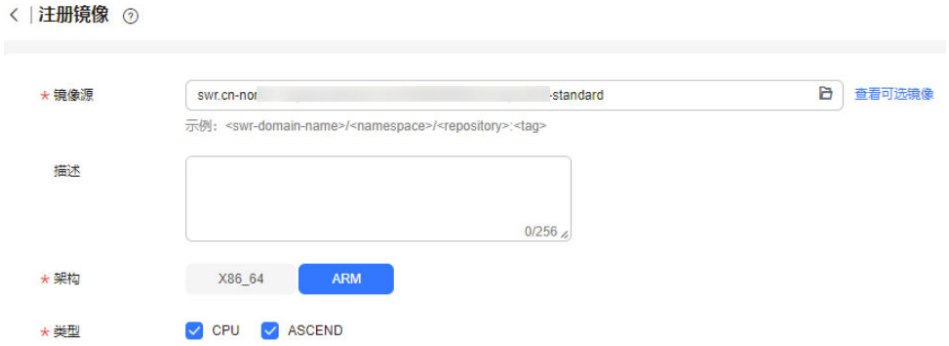
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中单击“注册镜像”。

图 4-769 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 4-770 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.43.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 4-771 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-772 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.43.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

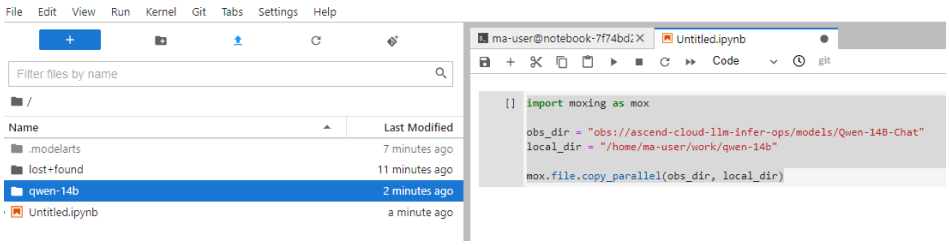
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-773 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

说明

通过命令npu-smi info查询NPU卡为容器中的第几张卡。例如下图查询出两张卡，如果希望使用第一和第二张卡，则“export ASCEND_RT_VISIBLE_DEVICES=0,1”，注意编号不是填4、5。

图 4-774 查询结果

```
+------------------------------------------------------------------------------------------+
| npu-smi 23.0.5.1                 Version: 23.0.5.1                                       |
+---------------------------+---------------+----------------------------------------------+
| NPU   Name                | Health        | Power(W)    Temp(C)           Hugepages-Usage(page)|
| Chip                      | Bus-Id        | AICore(%)   Memory-Usage(MB)  HBM-Usage(MB)        |
+===========================+===============+==============================================+
| 4     910B2               | OK            | 91.4        50                0    / 0         |
| 0                         | 0000:81:00.0  | 0           0    / 0          58682/ 65536         |
+===========================+===============+==============================================+
| 5     910B2               | OK            | 92.5        51                0    / 0         |
| 0                         | 0000:41:00.0  | 0           0    / 0          58670/ 65536         |
+===========================+===============+==============================================+
+---------------------------+---------------+----------------------------------------------+
| NPU     Chip              | Process id    | Process name             | Process memory(MB)      |
+===========================+===============+==============================================+
| 4       0                 | 10915         | python                   | 55400                   |
+===========================+===============+==============================================+
| 5       0                 | 21273         | python                   | 55388                   |
+===========================+===============+==============================================+
```

2. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **通过vLLM服务API接口启动服务**

  在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：/home/ma-user/work/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但

是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。

- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
        "prompt": "hello",
        "max_tokens": 100,
        "temperature": 0,
        "ignore_eos": false,
        "presence_penalty":2
 }'
```

vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**

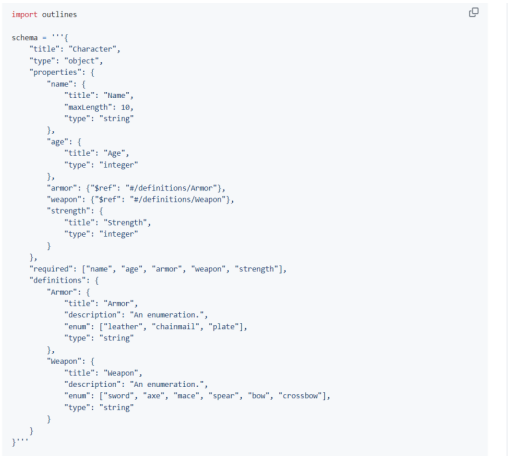
- 方式二：使用OpenAI接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
    "model": "${model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

表 4-404 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，如果需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必选属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 如果希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 4-775** guided_json 样例  如果想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\":

参数	是否必选	默认值	参数类型	描述
				{\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9。

表 4-405 不同模型推理支持的 max-model-len 长度

模型名	280T		313T	
	最小卡数	最大序列(K)	最小卡数	最大序列(K)
llama-7b	1	16	1	32
llama-13b	2	16	1	16
llama-65b	8	16	4	16
llama2-7b	1	16	1	32
llama2-13b	2	16	1	16
llama2-70b	8	32	4	64
llama3-8b	1	32	1	128
llama3-70b	8	32	4	64
qwen-7b	1	8	1	32
qwen-14b	2	16	1	16
qwen-72b	8	8	4	16

模型名	280T		313T	
	最小卡数	最大序列(K)	最小卡数	最大序列(K)
qwen1.5-0.5b	1	128	1	256
qwen1.5-7b	1	8	1	32
qwen1.5-1.8b	1	64	1	128
qwen1.5-14b	2	16	1	16
qwen1.5-32b	4	32	2	64
qwen1.5-72b	8	8	4	16
qwen1.5-110b	oom		8	128
qwen2-0.5b	1	128	1	256
qwen2-1.5b	1	64	1	128
qwen2-7b	1	32	1	64
qwen2-72b	8	32	4	64
chatglm2-6b	1	64	1	128
chatglm3-6b	1	64	1	128
glm-4-9b	1	32	1	128
baichuan-7b	1	16	1	32
baichuan-13b	2	4	1	4
baichuan2-7b	1	8	1	32
baichuan2-13b	2	4	1	4
yi-6b	1	64	1	128
yi-9b	1	32	1	64
yi-34b	4	32	2	64
deepseek-llm-7b	1	16	1	32
deepseek-coder-instruct-33b	4	32	2	64
deepseek-llm-67b	8	32	4	64
mistral-7b	1	32	1	128
mixtral-8x7b	4	8	2	32
gemma-2b	1	64	1	128
gemma-7b	1	8	1	32

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

4.43.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

  说明

  - 若需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
  - 权重文件夹不要以"model"命名，若以"model"命名会导致后续创建AI应用报错。
- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-776 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
python -m vllm.entrypoints.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

参数说明：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。--tensor-parallel-size：并行卡数。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。
```

Step2 部署模型

在ModelArts控制台的AI应用模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“资产管理 > AI应用 > 创建”，开始创建AI应用。
2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-777 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

   首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

   说明

   若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“模型部署 > 在线服务 > 部署”，开始部署在线服务。
2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-778 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

注：若部署在线服务出现报错starting container process caused "exec: \"/home/mind/model/run_vllm.sh\": permission denied"，请参考**附录：大模型推理standard常见问题**问题6重新构建镜像。

Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

若以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-779 预测-vllm

若以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-780 预测-openai

在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.43.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
```

3. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤4进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

4. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

5. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。


```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
```

output_path: 要保存的结果路径。

6. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算这n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预置变量

   ```
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
   ```

   执行脚本如下：

   ```
   python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
   ```

   output_path 指定保存结果的路径。

   参考模型llama3系列模型，数据集 mmlu 为例，配置如下：

   表 4-406 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

7. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```


参数说明如下:

- --datasets, 评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法
- --hf-type, HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择
- --hf-path, 本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B
- --max-seq-len, 模型的最大序列长度
- --max-out-len, 模型的最大输出长度
- --hf-num-gpus, 需要使用的卡数
- --batch-size, 推理每次处理的输入数目
- -w 存放输出结果的目录

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

```
npu：
mmlu：46.6
gpu：
mmlu：47
```

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

4.43.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，若静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。

- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

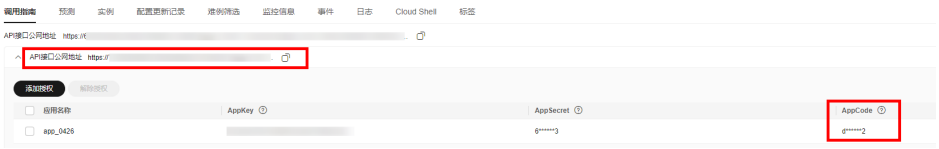
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-781 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。

- --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-782 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_dataset.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

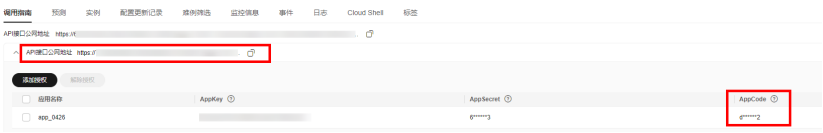
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-783 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。

- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-784 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每token平均时延（ms）	首token平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.8237294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.43.7 推理模型量化

4.43.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化，量化方法为per-group。

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst，注意需指定到val.jsonl的上一级目录。

  详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.43.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。

2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数

- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

4.43.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

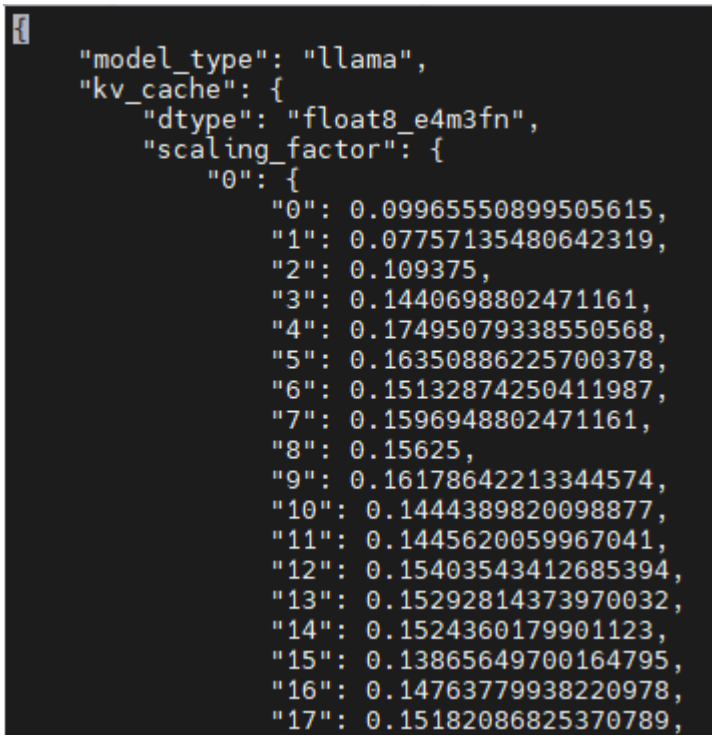
使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
```

```
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-785 抽取 kv-cache 量化系数



注意：

1、抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。

2、当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.43.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

以下值是在gpu-memory-utilization为0.9时测试得出，为服务部署所需的最小昇腾卡数及该卡数下推荐的最大max-model-len长度，不代表最佳性能。

以llama2-13b为例，NPU卡显存为32GB时，至少需要2张卡运行推理业务，2张卡运行的情况下，推荐的最大序列max-model-len长度最大是16K，此处的单位K是1024，即16*1024。

测试方法：gpu-memory-utilization为0.9下，以4k、8k、16k递增max-model-len，直至达到能执行静态benchmark下的最大max-model-len。

表 4-407 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
1	llama-7b	1	16	1	32
2	llama-13b	2	16	1	16
3	llama-65b	8	16	4	16
4	llama2-7b	1	16	1	32
5	llama2-13b	2	16	1	16
6	llama2-70b	8	32	4	64
7	llama3-8b	1	32	1	128
8	llama3-70b	8	32	4	64
9	qwen-7b	1	8	1	32
10	qwen-14b	2	16	1	16
11	qwen-72b	8	8	4	16
12	qwen1.5-0.5b	1	128	1	256
13	qwen1.5-7b	1	8	1	32
14	qwen1.5-1.8b	1	64	1	128
15	qwen1.5-14b	2	16	1	16
16	qwen1.5-32b	4	32	2	64
17	qwen1.5-72b	8	8	4	16

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
18	qwen1.5-110b	--		8	128
19	qwen2-0.5b	1	128	1	256
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32
38	falcon-11b	1	8	1	64

4.43.9 附录：大模型推理 standard 常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

  解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

- 问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len。

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

  config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

  ```
  from vllm import LLM, SamplingParams
  llm = LLM(model="facebook/opt-125m", block_size=128)
  ```

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmoothQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：部署在线服务报错starting container process caused "exec: \"/home/mind/model/run_vllm.sh\": permission denied"

  解决方法：修改AscendCloud-6.3.907-xxx.zip压缩包中llm_inference/ascend_vllm/build_image.sh内容，将'ENTRYPOINT ["/home/mind/model/run_vllm.sh"]'修改为'ENTRYPOINT sh /home/mind/model/run_vllm.sh'，并重新构建镜像。

  见如下示例：

图 4-786 修改 build_images.sh

```
#!/bin/bash

OPTIONS=$(getopt -n "$0" -o i:e:n --long base-image:,specify-enrtypoint:,image-name: -- "$@")

if [ $? != 0 ]; then
    echo "args error"
    exit 1
fi

eval set -- "$OPTIONS"

while true; do
    case "$1" in
        -i|--base-image)
            base_image="$2"
            shift 2
            ;;
        -e|--specify-enrtypoint)
            specify_enrtypoint="$2"
            shift 2
            ;;
        -n|--image-name)
            image_name="$2"
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "unknown options"
            exit 1
            ;;
    esac
done

if [ -z "$base_image" ]; then
    echo "--base-image not specified"
    exit 1
fi

if [ -z "$image_name" ]; then
    echo "--image-name not specified"
    exit 1
fi

if [ -n "$specify_enrtypoint" ]; then
    echo 'ENTRYPOINT sh /home/mind/model/run_vllm.sh' >> Dockerfile
fi

cd ../../../../
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile ./
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/.dockerignore ./

docker build -t $image_name --build-arg BASE_IMAGE=$base_image .

rm -f Dockerfile
rm -f .dockerignore

```

4.44 主流开源大模型基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.906）

4.44.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考表4-410获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如表4-408所示。

表 4-408 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 4-787 操作流程图

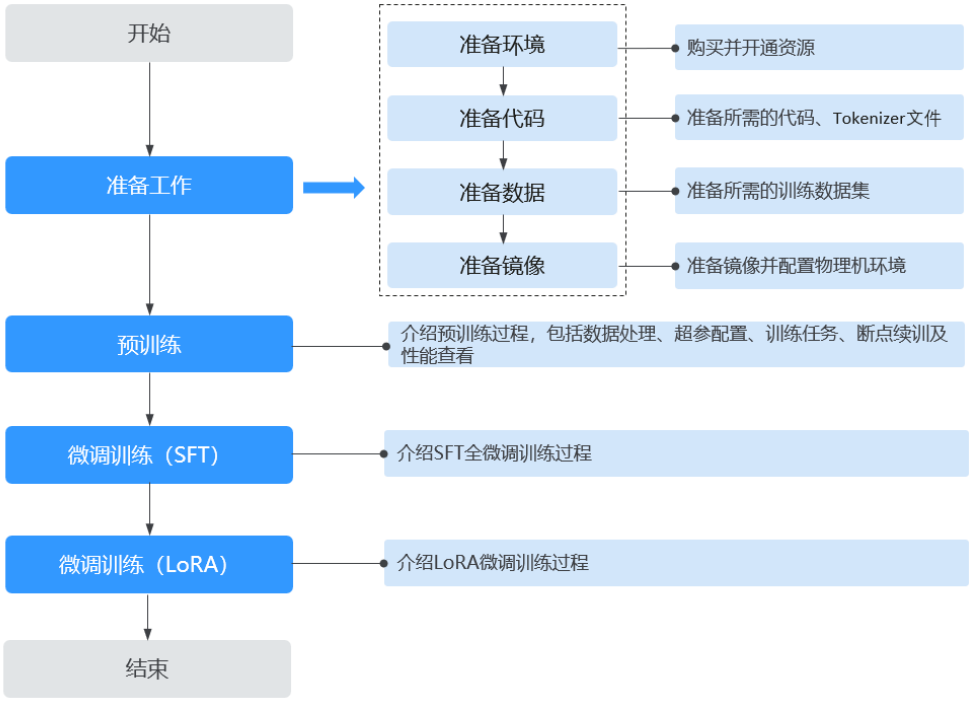


表 4-409 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

4.44.2 准备工作

4.44.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-418**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

4.44.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-410**所示，模型列表、对应的开源权重获取地址如**表4-411**所示。

表 4-410 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

表 4-411 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

模型软件包结构说明

AscendCloud代码包结构介绍如下：

```
├──AscendCloud-LLM
  ├──llm_train                # 模型训练代码包
     ├──AscendSpeed           # 基于AscendSpeed的训练代码
        ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
        ├──scripts/             # 训练需要的启动脚本
           ├──llama2            # llama2系列模型执行脚本的文件夹
           ├──llama3            # llama3系列模型执行脚本的文件夹
           ├──qwen              # Qwen系列模型执行脚本的文件夹
           ├──qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
           ├── ...
           ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
           ├── install.sh       # 环境部署脚本
        ├──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  ├──llm_inference              # 推理代码包
```

```
   |──llm_tools                    # 推理工具
|──AscendCloud-OPP                 #依赖算子包
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── data              # 预处理后数据
          |── pretrain   # 预训练加载的数据
          |── finetune   # 微调加载的数据
      |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs              # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizer                    #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── model                        #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                  #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-*.zip
   ```

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

4.44.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下this格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key 标志来选择用于训练的列。

```
{
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
}
```

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

4.44.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-412 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

表 4-413 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

步骤 1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤 2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤 3 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
```



```
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

1. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
3. 使用ma-user用户安装依赖包。
   ```
   #进入scripts目录换
   cd /home/ma-user/ws/llm_train/AscendSpeed
   #执行安装命令
   sh scripts/install.sh
   ```
4. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：
   ```
   |——AscendCloud-LLM
     |——llm_train                # 模型训练代码包
       |——AscendSpeed            # 基于AscendSpeed的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——scripts/             # 训练需要的启动脚本
         |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/           # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/             # MindSpeed昇腾大模型加速库
       |——ModelLink/             # ModelLink端到端的大语言模型方案
         |——megatron/            # 注意：该文件夹从Megatron-LM中复制得到
         |——...
   ```

4.44.3 预训练任务

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-414**所示。其他超参均有默认值，可以参考**表4-417**按照实际需求修改。

表 4-414 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤 3 启动训练脚本

请根据**步骤2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
示例:
sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-788 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

4.44.4 SFT 全参微调训练任务

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-414**所示。其他超参均有默认值，可以参考**表4-417**按照实际需求修改。

表 4-415 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B、ChatGLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤 3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.44.5 LoRA 微调训练

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-414**所示。其他超参均有默认值，可以参考**表4-417**按照实际需求修改。

表 4-416 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B、ChatGLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-418**所示。

步骤 3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
```

```
#第一台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

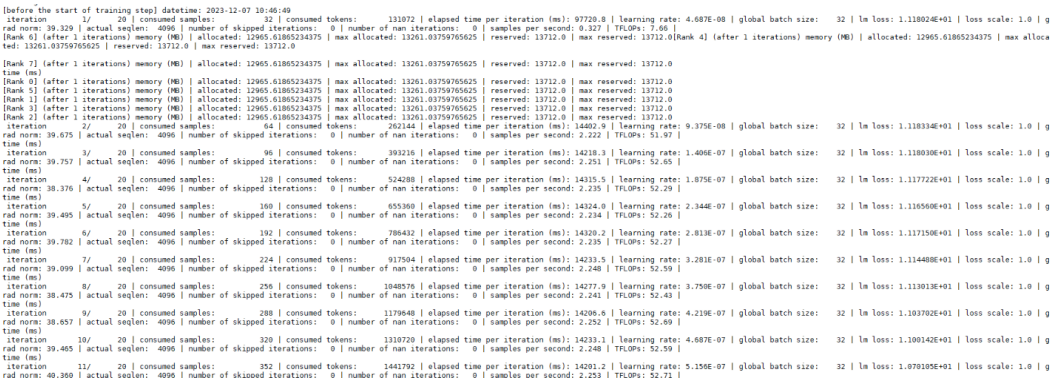
训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

4.44.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-789 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

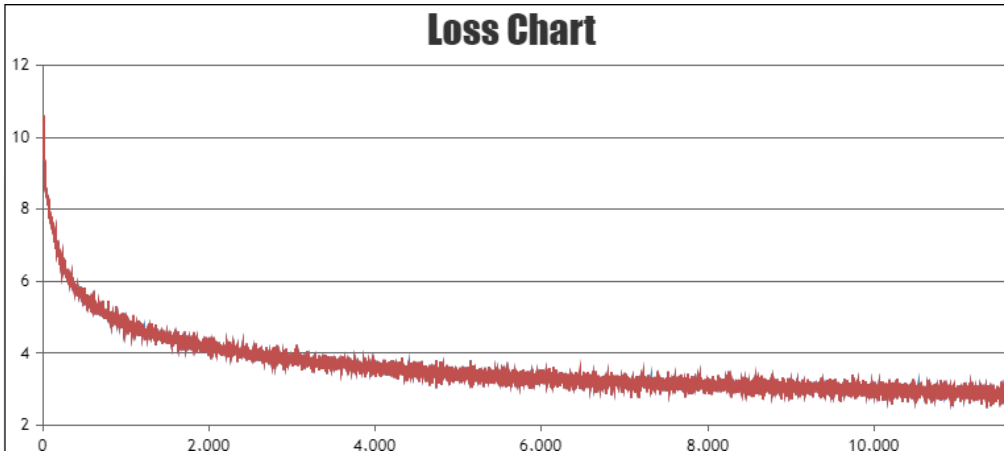
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-417**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-790**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-790 Loss 收敛情况（示意图）



4.44.7 训练脚本说明

4.44.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 4-417 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-418所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-418 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

4.44.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/finetune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的handler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

- 先调用self.get_tokenized_data()对数据集进行encode
- self.get_tokenized_data()中调用self._filter方法处理每一个sample
- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 对数据集 full_prompt 中的 user_prompt 进行 mask 操作。

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
```

```
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

   i. 对user和assistant的文本进行清洗

   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assistant_ids

   iii. input_ids是user_ids和assistant_ids的拼接

   iv. labels与input_ids对应，用-100替换user_ids的token，只保留assistant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-419 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**

环境变量	示例	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.44.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 4-420 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.44.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-791 修改 ChatGLMv3-6B tokenizer 文件

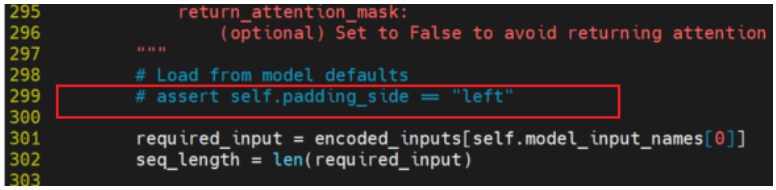
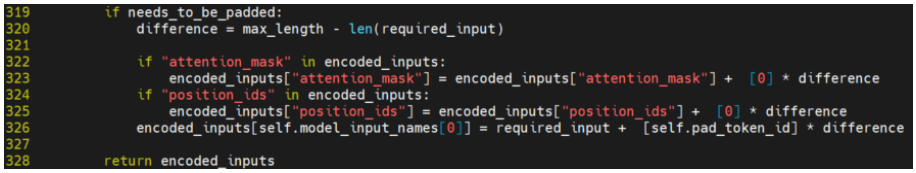


图 4-792 修改 ChatGLMv3-6B tokenizer 文件



GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-793 修改 ChatGLMv4-9B tokenizer 文件

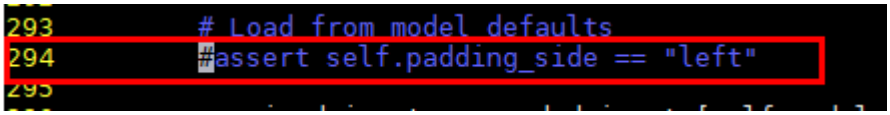
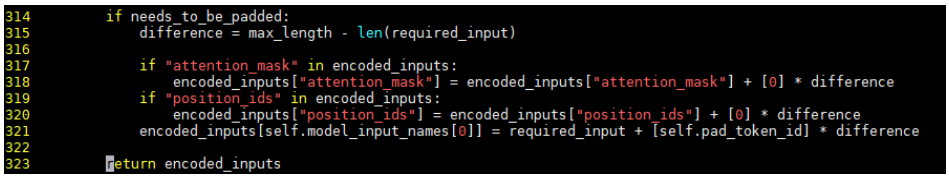


图 4-794 修改 ChatGLMv4-9B tokenizer 文件



Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下代码信息进行查找，修改后如图所示。

图 4-795 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41
42
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.45 主流开源大模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.906）

4.45.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.4.2版本。
- 支持FP16和BF16数据类型推理。
- Server驱动版本要求23.0.5。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-421 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	cann_8.0.rc2

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表4-422所示。

表 4-422 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.4.2版本。不同vLLM版本支持的模型列表有差异，具体如表4-423所示。

表 4-423 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
2	llama-13b	√	√	√	√	https://huggingface.co/huggyllama/llama-13b
3	llama-65b	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.906中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
      ├── vllm_npu       # 推理源码
      ├── ascend_vllm-0.4.2-py3-none-any.whl   # 推理安装包
      ├── build.sh              # 推理构建脚本
      ├── vllm_install.patch       # 社区昇腾适配的补丁包
 ├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
      ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
      ├── autosmoothquant               # 量化代码
      ├── build.sh                    # 安装量化模块的脚本
  ├──awq            # W4A16量化工具
    ├──convert_awq_to_npu.py   # awq权重转换脚本
  ├──llm_evaluation    # 推理评测代码包
    ├──benchmark_tools    #性能评测
      ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
      ├── benchmark_parallel.py   # 评测静态性能脚本
      ├── benchmark_serving.py    # 评测动态性能脚本
      ├── benchmark_utils.py      # 抽离的工具集
      ├── generate_datasets.py    # 生成自定义数据集的脚本
      ├── requirements.txt        # 第三方依赖
    ├──benchmark_eval #精度评测
      ├──opencompass.sh       #运行opencompass脚本
      ├──start.sh            #安装opencompass脚本
      ├──vllm_api.py          #启动vllm api服务器
      ├──vllm.py             #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型基于LiteServer适配PyTorch NPU训练指导（6.3.906）**。

4.45.2 部署推理服务

本章节介绍如何使用vLLM 0.4.2框架部署并启动推理服务。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 安装过程需要连接互联网git clone，确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   #查看驱动和固件版本
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   驱动版本要求是23.0.5。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-421**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.906-xxx.zip和算子包AscendCloud-OPP-6.3.906-xxx.zip到主机中，包获取路径请参见**表4-422**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-423**。

   如果使用模型训练后的权重文件进行推理，需要上传训练后的权重文件和开源的原始权重文件。模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step4 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

Step5 进入容器安装推理依赖软件

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
```
docker exec -it ${container_name} bash
```
2. 上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```
3. 解压算子包并将相应算子安装到环境中。
```
unzip AscendCloud-OPP-*.zip
pip install ascend_cloud_ops-1.0.0-py3-none-any.whl
pip install cann_ops-1.0.0-py3-none-any.whl
```
4. 解压软件推理代码并安装依赖包。安装过程需要连接互联网git clone，请确保容器环境可以访问公网。
```
unzip AscendCloud-LLM-*.zip
cd llm_inference/ascend_vllm
bash build.sh
```
运行完后，会安装适配昇腾的vllm-0.4.2版本。

Step6 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```

说明

NPU卡编号可以通过命令npu-smi info查询。

2. 配置环境变量。
```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
```

```
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**或**使用SmoothQuant量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **方式一：通过OpenAI服务API接口启动服务**

     在llm_inference/ascend_vllm/vllm-gpu-0.4.2目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --dtype=float16 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   - **方式二：通过vLLM服务API接口启动服务**

     在llm_inference/ascend_vllm/vllm-gpu-0.4.2目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.api_server --model ${container_model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --dtype=float16 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   推理服务基础参数说明如下：

   - --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。若使用了量化功能，则使用**推理模型量化**章节转换后的权重。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
   - --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
   - --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。
   - --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
   - --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考1。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择awq或smoothquant方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即Step3 上传代码包和权重文件上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见表4-424。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
```

```
    "ignore_eos": false,
    "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-424 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。

参数	是否必选	默认值	参数类型	描述
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

4.45.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*xxx*.zip的llm_tools/llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态、动态性能评测脚本
├── requirements.txt      # 第三方依赖
```

目前性能测试还不支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，切换一个conda环境，执行如下命令安装性能测试的关依赖。

```
conda activate python-3.9.10
pip install -r requirements.txt
```

3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
--parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。上面命令中使用vllm举例。
- --host ${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口8080。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。

4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-796 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

方法一：使用公开数据集

- ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
- Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

方法二：使用generate_dataset.py脚本生成数据集方法：

generate_dataset.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```

generate_dataset.py脚本执行参数说明如下：

- --dataset：数据集保存路径，如custom_datasets.json
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python benchmark_serving.py --backend vllm --host ${docker_ip} --port 8080 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如tgi，vllm，mindspore，openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是huggingface的权重路径。backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。

- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。

3. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-797 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.45.4 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。

```
benchmark_eval
├──opencompass.sh      #运行opencompass脚本
├──install.sh          #安装opencompass脚本
├──vllm_api.py         #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
```

2. 确保容器内通网，未通网需要配置$config_proxy_str，$config_pip_str设置对应的代理和pip源，来确保当前代理和pip源可用。
3. 精度评测新建一个conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。命令中的$work_dir 是benchmark_eval的绝对路径。

```
conda activate python-3.9.10 #如果没有该conda环境需要手动建立一个
export work_dir=${work_dir} #指定work_dir路径
bash install.sh
```

4. 在benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

5. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤6进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
```

```
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                       exec(check_program, exec_globals)      #第58行
```

6. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen 。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：评测数据集类型，分为eval、static、awq，也就是精度、静态和量化数据集，默认eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

7. 这一步可以在客户端显示运行过程，通过run.py运行。如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
```

output_path: 要保存的结果路径。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1,）认为NPU精度和GPU对齐。

4.45.5 推理模型量化

4.45.5.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-423**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：per-group

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 在容器中使用ma-user用户运行以下命令下载并安装AutoAWQ源码。
   ```
   git clone -b v0.2.5 https://github.com/casper-hansen/AutoAWQ.git AutoAWQ-0.2.5
   cd ./AutoAWQ-0.2.5
   export PYPI_BUILD=1
   pip install -e .
   ```
2. 需要编辑“examples/quantize.py”文件，针对NPU进行如下适配工作，以支持在NPU上进行量化。
   a. 添加import。
      ```
      import torch_npu
      from torch_npu.contrib import transfer_to_npu
      ```
   b. 指定模型输入、输出路径。
      ```
      model_path = **
      quant_path = **
      ```
   c. 可以指定校准数据集路径，如calib_data="/path/to/pile-val"，如不指定，默认数据集是“mit-han-lab/pile-val-backup”。
      ```
      model.quantize(tokenizer, quant_config=quant_config, calib_data="/path/to/pile-val",
      split="validation")
      ```
3. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。
   ```
   pip install transformers sentencepiece #安装量化工具依赖
   export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
   python examples/quantize.py
   ```
   详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python  awq/convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

4.45.5.2 使用 SmoothQuant 量化

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-423**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置环境。
   ```
   cd llm_tools/AutoSmoothQuant/
   sh build.sh
   ```
2. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。
3. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

   参数说明:

   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。

- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

4. 启动smoothQuant量化服务。

   参考**Step6 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

4.45.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-423**。

Step1 使用 tensorRT 量化工具进行模型量化

使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

量化脚本convert_checkpoint.py存放在TensorRT-LLM/examples路径对应的模型文件夹下，例如：llama模型对应量化脚本的路径是examples/llama/convert_checkpoint.py。

执行convert_checkpoint.py脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache**。

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf    \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

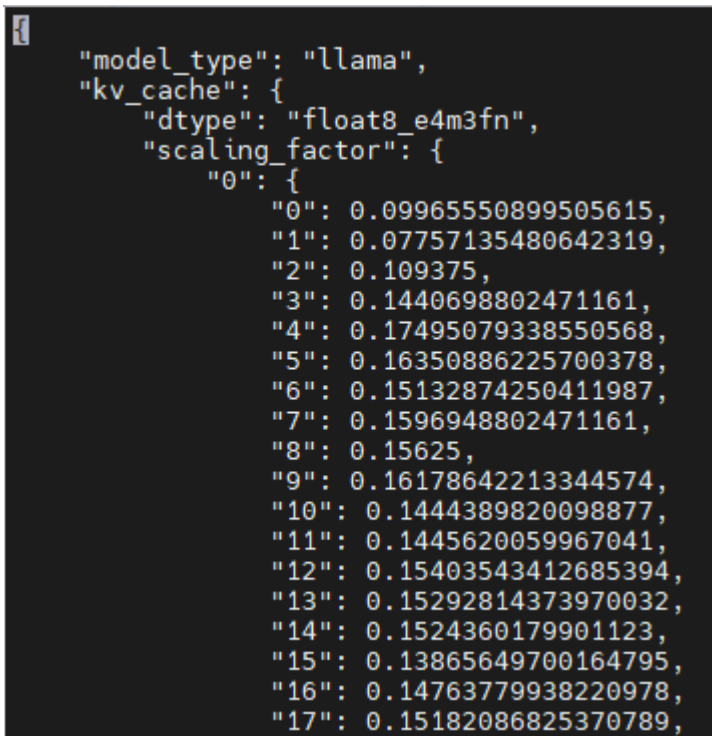
Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：



```
{
    "model_type": "llama",
    "kv_cache": {
        "dtype": "float8_e4m3fn",
        "scaling_factor": {
            "0": {
                "0": 0.09965550899505615,
                "1": 0.07757135480642319,
                "2": 0.109375,
                "3": 0.1440698802471161,
                "4": 0.17495079338550568,
                "5": 0.16350886225700378,
                "6": 0.15132874250411987,
                "7": 0.1596948802471161,
                "8": 0.15625,
                "9": 0.16178642213344574,
                "10": 0.1444389820098877,
                "11": 0.1445620059967041,
                "12": 0.15403543412685394,
                "13": 0.15292814373970032,
                "14": 0.1524360179901123,
                "15": 0.13865649700164795,
                "16": 0.14763779938220978,
                "17": 0.15182086825370789,
```


注意：

1、 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。

2、当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.45.6 附录：大模型推理常见问题

问题1：在推理预测过程中遇到NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

4.46 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.906）

4.46.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表4-425所示。

表 4-425 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 4-798 操作流程图

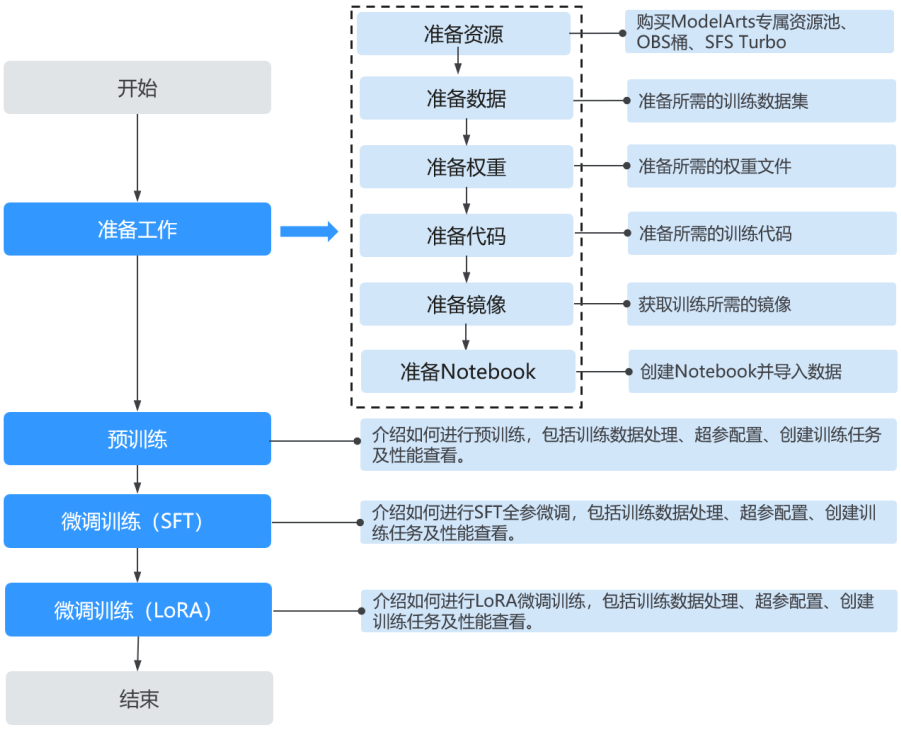


表 4-426 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.46.2 准备工作

4.46.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-433**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统，详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-799 创建 SFS Turbo

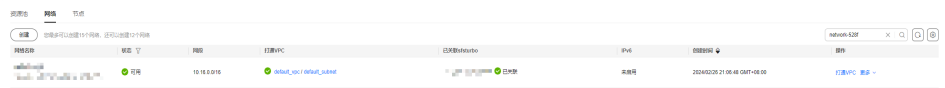
其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-800 SFS 类型和容量选择

ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。 具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-801 ModelArts 网络关联 SFS Turbo



4.46.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

```
{
   'id': '1',
   'url': 'https://simple.wikipedia.org/wiki/April',
   'title': 'April',
   'text': 'April is the fourth month...'
}
```

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.46.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-425**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
```

```
├── tokenizer.model
└── USE_POLICY.md
```

4.46.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-427**所示。

表 4-427 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.906代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|──llm_train                # 模型训练代码包
        |──AscendSpeed        # 基于AscendSpeed的训练代码
              |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |──scripts/             # 训练需要的启动脚本
                  |──llama2           # llama2系列模型执行脚本的文件夹
                  |──llama3           # llama3系列模型执行脚本的文件夹
                  |──qwen             # Qwen系列模型执行脚本的文件夹
                  |──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                  |── ...
                  |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                  |── install.sh       # 环境部署脚本
              |──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|──llm_inference              # 推理代码包
|──llm_tools                  # 推理工具
```

代码上传至 OBS

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed               # 代码目录
          |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── data              # 预处理后数据
                |── pretrain    # 预训练加载的数据
```

```
            |── finetune   # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
    |── saved_dir_for_output          # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs            # 训练过程中日志（loss、吞吐性能）
              |──── saved_models
            |── lora            # lora微调输出权重
            |── sft             # 增量训练输出权重
            |── pretrain        # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── tokenizer                  #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13B-chat-hf
|── model                      #原始权重与tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── llama2-13B-chat-hf
```

4.46.2.5 准备镜像

准备训练Llama2-13B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-428 基础容器镜像地址

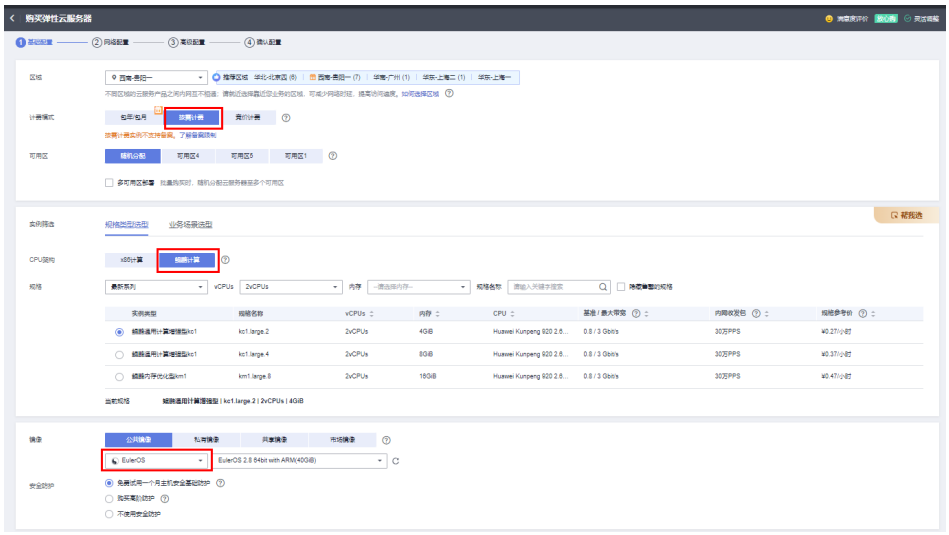
镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-802 购买 ECS



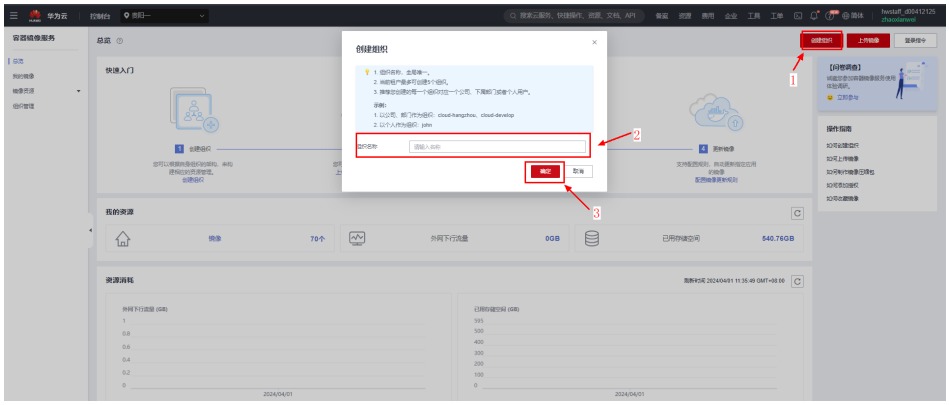
Step2 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

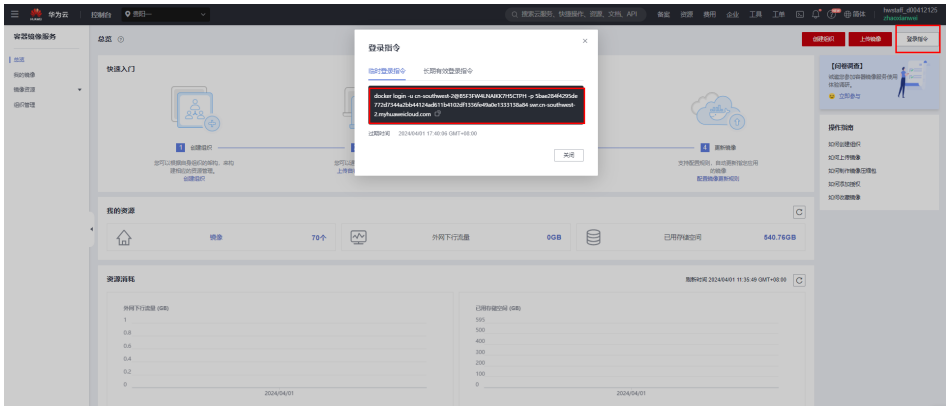
图 4-803 创建镜像组织



Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-804 复制登录指令



Step5 获取训练镜像

请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-428**。

```
docker pull {image_url}
```

Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

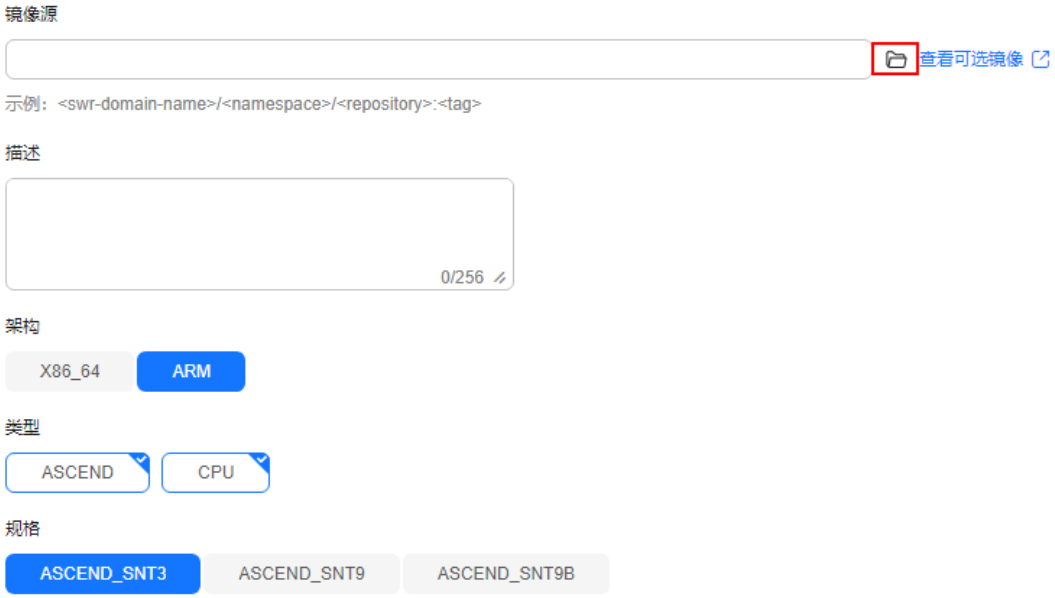
docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

Step7 ModelArts 中注册镜像

镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。

1. 登录ModelArts管理控制台，在左侧导航栏选择“资产管理 > 镜像管理”，然后在“镜像管理”页面右上角单击“注册镜像”。
2. 在“注册镜像”页面，选择已上传的镜像源，“架构”选择“ARM”，“类型”选中“ASCEDN”和“CPU”，按需选择规格，然后单击“立即注册”。

图 4-805 选择已上传的镜像源



4.46.2.6 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-806 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例，子目录挂载可选择默认不填写。

如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 4-807 Notebook 中选择弹性文件服务

使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包，每次启动训练作业时会执行该命令安装。

通过运行install.sh脚本，会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
|——AscendCloud-LLM
  |——llm_train                # 模型训练代码包
      |——AscendSpeed          # 基于AscendSpeed的训练代码
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
          |——scripts/             # 训练需要的启动脚本
          |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
      |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
      |——MindSpeed/           # MindSpeed昇腾大模型加速库
      |——ModelLink/           # ModelLink端到端的大语言模型方案
          |——megatron/        # 注意：该文件夹从Megatron-LM中复制得到
          |——...
```

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

由于训练启动命令也会执行sh scripts/install.sh安装依赖包，因此Notebook保存镜像为可选操作。

图 4-808 安装依赖包

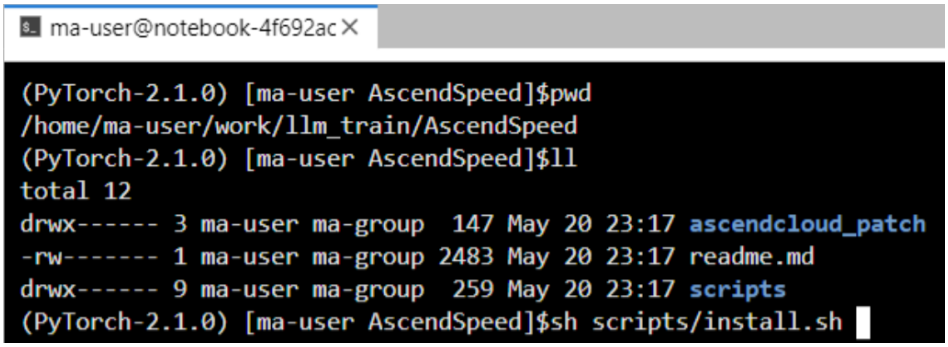


图 4-809 保存镜像

图 4-810 填写保存镜像相关参数

4.46.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-429**所示。其他超参均有默认值，可以参考**表4-432**按照实际需求修改。

表 4-429 必须修改的训练超参配置

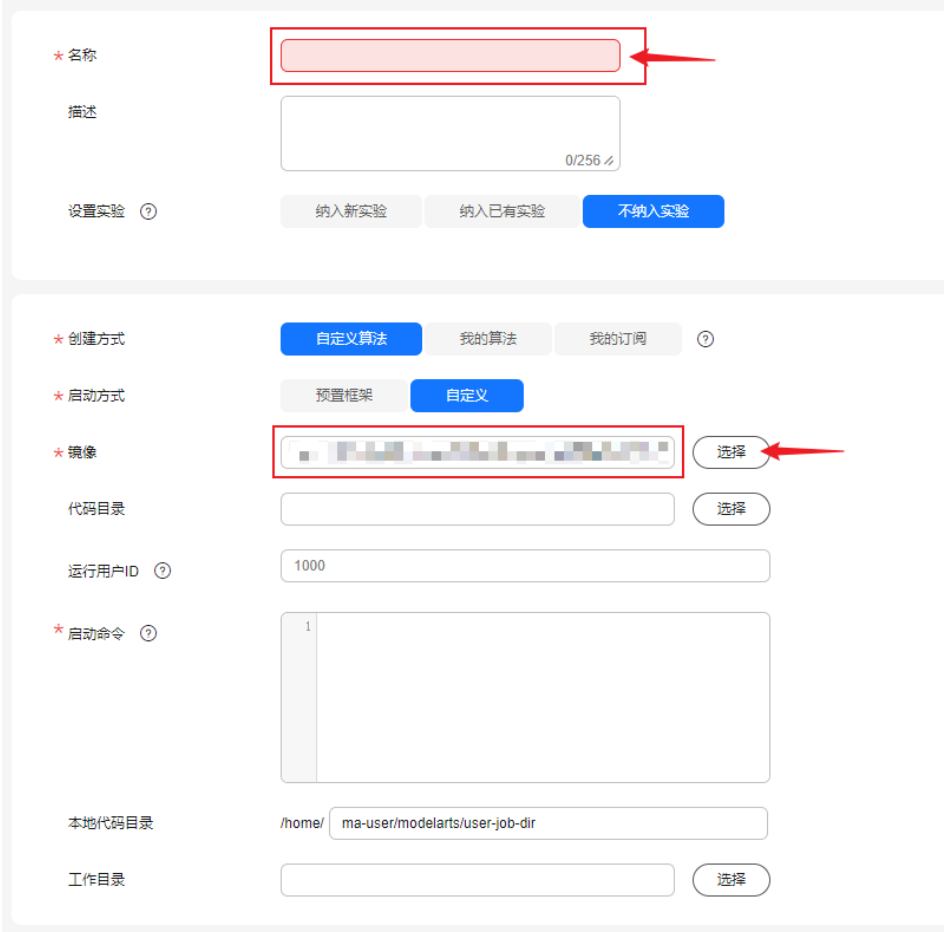
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-811 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

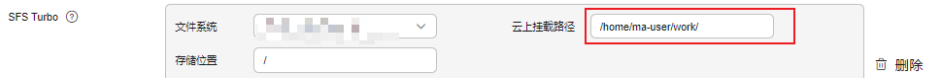
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-433**进行配置。

图 4-812 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-813 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.46.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-430**所示。其他超参均有默认值，可以参考**表4-432**按照实际需求修改。

表 4-430 必须修改的训练超参配置

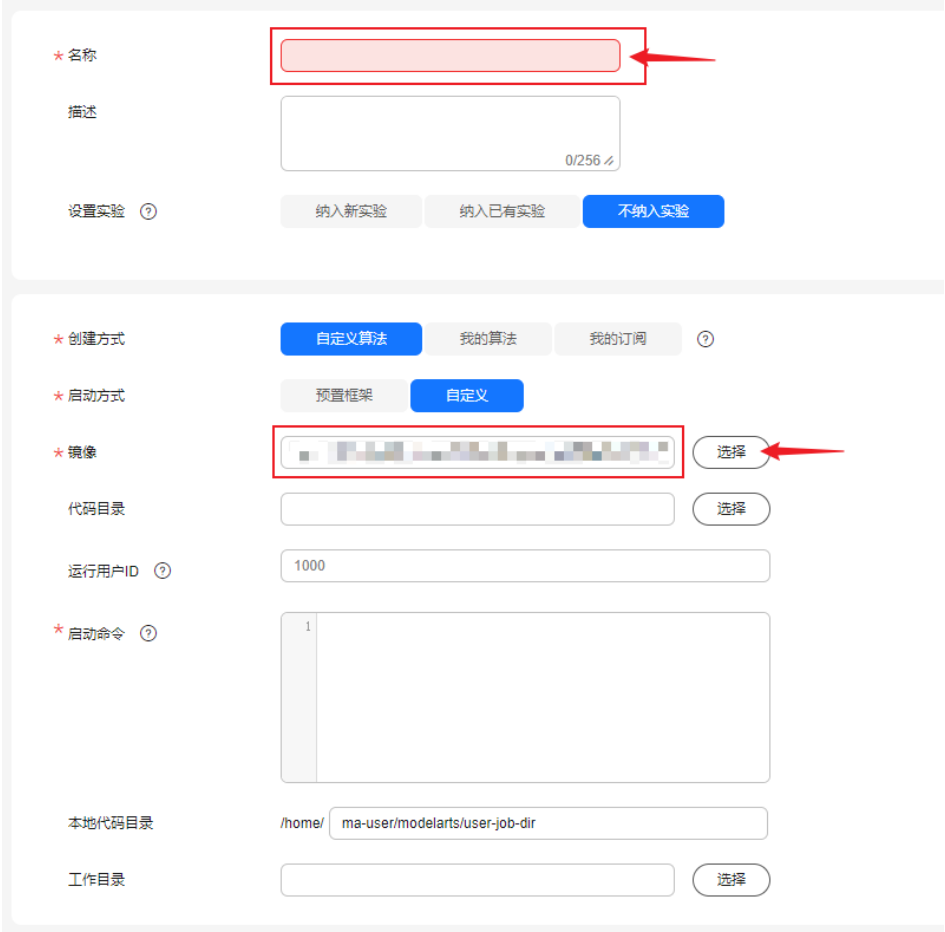
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-814 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-433**进行配置。

图 4-815 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-816 选择 SFS Turbo

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.46.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-431**所示。其他超参均有默认值，可以参考**表4-432**按照实际需求修改。

表 4-431 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

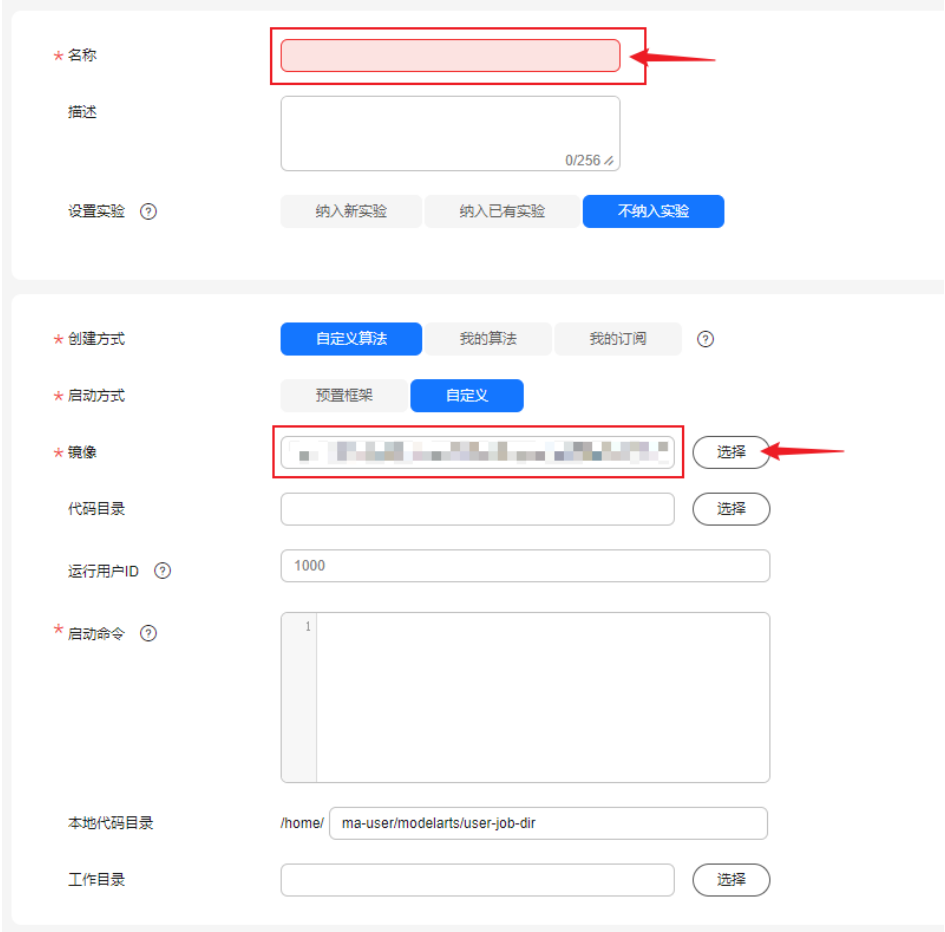
说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表4-433**所示。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-817 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

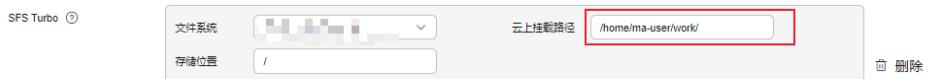
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-433**进行配置。

图 4-818 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-819 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

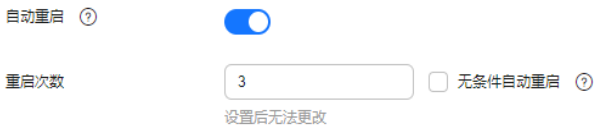
提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.46.6 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 4-820 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数（支持预训练、LoRA微调、SFT微调）。

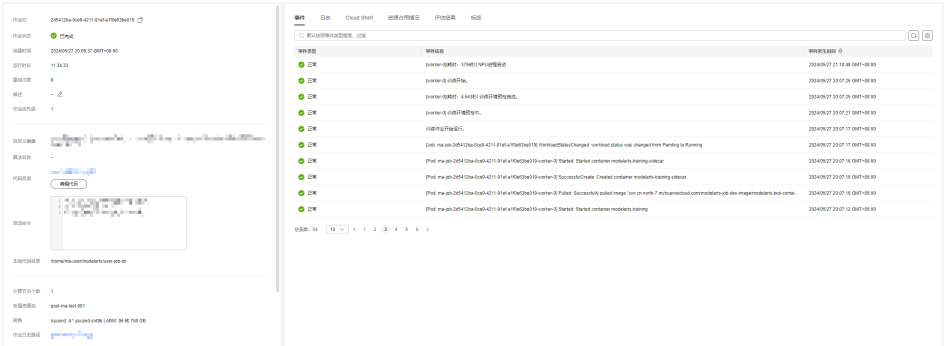
说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

4.46.7 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

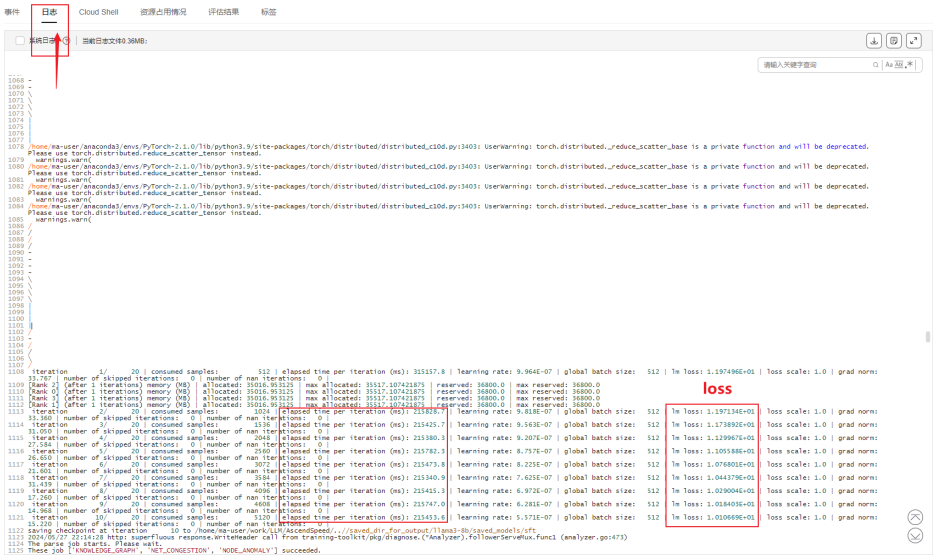
图 4-821 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-822 查看日志和性能



4.46.8 训练脚本说明

4.46.8.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-432 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-433所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-433 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

4.46.8.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。

- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-434 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.46.8.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。

- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 4-435 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.46.8.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-823 修改 ChatGLMv3-6B tokenizer 文件

```
295             return_attention_mask:
296                 (optional) Set to False to avoid returning attention
297         """
298         # Load from model defaults
299         # assert self.padding_side == "left"
300
301         required_input = encoded_inputs[self.model_input_names[0]]
302         seq_length = len(required_input)
303
```

图 4-824 修改 ChatGLMv3-6B tokenizer 文件

```
319         if needs_to_be_padded:
320             difference = max_length - len(required_input)
321 
322             if "attention_mask" in encoded_inputs:
323                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] +  [0] * difference
324             if "position_ids" in encoded_inputs:
325                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] +  [0] * difference
326             encoded_inputs[self.model_input_names[0]] = required_input +  [self.pad_token_id] * difference
327 
328         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

文件最后几处代码中需要修改，具体位置可根据上下文代码信息进行查找，修改后如图所示。

图 4-825 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295 
```

图 4-826 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件，具体位置可根据上下代码信息进行查找，修改后如图所示。

图 4-827 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.47 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.906）

4.47.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.4.2版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持的模型列表、对应的开源权重获取地址如**表4-436**所示。

表 4-436 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
3	llama-65b	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-33b-instruct	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

操作流程

图 4-828 操作流程图

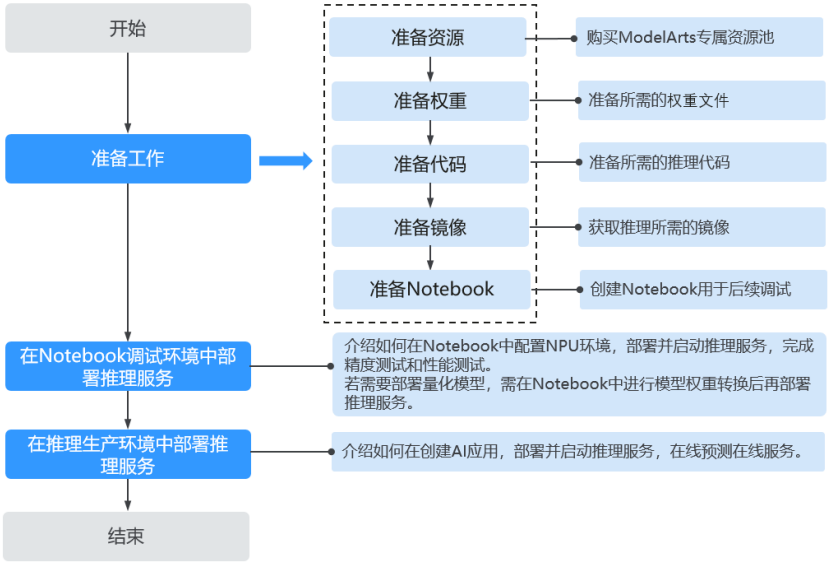


表 4-437 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.906-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 若需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.47.2 准备工作

4.47.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.47.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-436**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.47.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-438**所示。

表 4-438 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.906中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.4.2-py3-none-any.whl   # 推理安装包
        ├── build.sh                # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
  ├──llm_tools           # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
    ├──awq              # W4A16量化工具
      ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──llm_evaluation    # 推理评测代码包
      ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
      ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──start.sh             #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

4.47.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-439 基础容器镜像地址

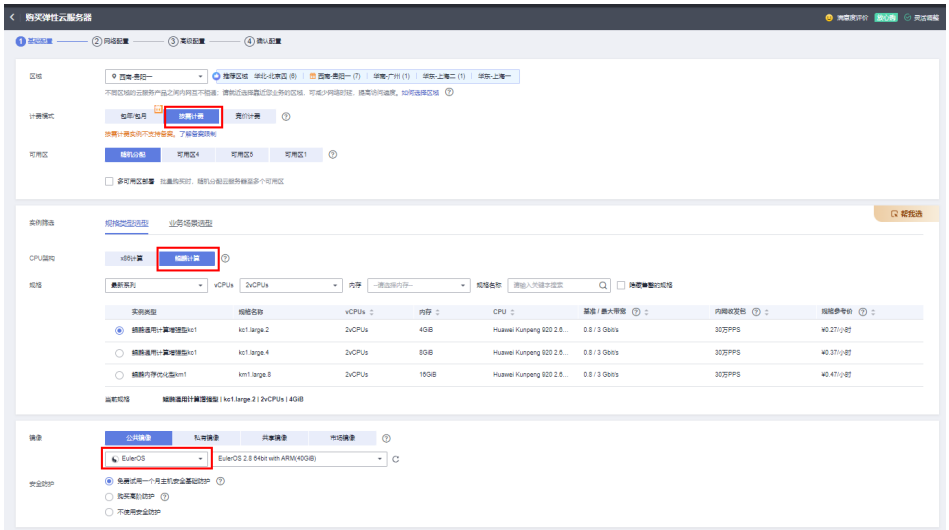
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-829 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-830 创建镜像组织

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包和依赖包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-438**。

在ModelArts官方提供的基础镜像上，构建一个用于ModelArts Standard推理部署的镜像。

在模型软件包和依赖包的同层目录下，创建并编辑Dockerfile。

```
vim Dockerfile
```

Dockerfile内容如下：

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

USER ma-user
COPY AscendCloud-*.zip /home/ma-user/
RUN unzip -o /home/ma-user/AscendCloud-*.zip
RUN unzip -o /home/ma-user/AscendCloud-LLM-*.zip
RUN unzip -o /home/ma-user/AscendCloud-OPP-*.zip

RUN pip install /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN cd /home/ma-user/llm_inference/ascend_vllm && bash /home/ma-user/llm_inference/ascend_vllm/build.sh

ENTRYPOINT sh /home/mind/model/run_vllm.sh
```

构建镜像。

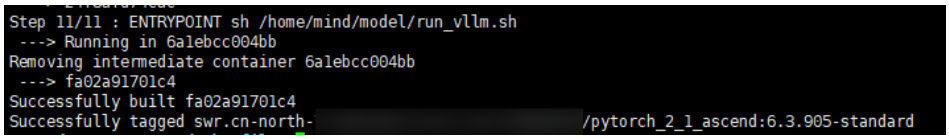
```
docker build -t swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag> .
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示构建镜像成功。

图 4-831 成功构建镜像

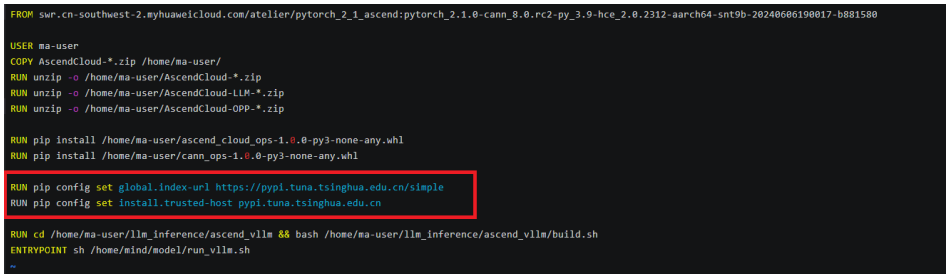


注：若构建镜像时报错pip超时，可在Dockerfile中添加如下命令设置pip源

```
RUN pip config set global.index-url https://xxx/simple
RUN pip config set install.trusted-host xxx
```

如下图所示：

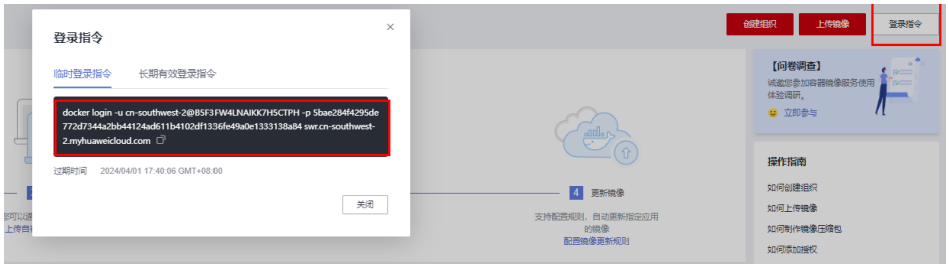
图 4-832 dockerfile 添加 pip 源



Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-833 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示上传镜像成功。

图 4-834 成功上传镜像

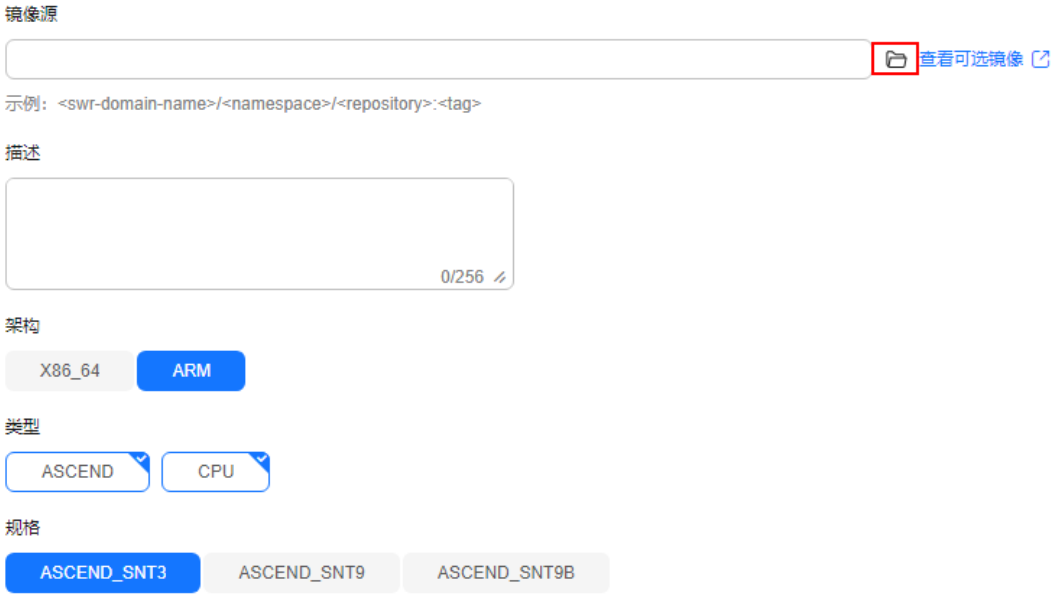
Step8 注册镜像

镜像上传至SWR成功后，在ModelArts控制台注册镜像。

1. 登录ModelArts管理控制台，在左侧导航栏选择“资产管理 > 镜像管理”，然后在“镜像管理”页面右上角单击“注册镜像”。

2. 在“注册镜像”页面，“镜像源”选择上一步上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，“架构”选择“ARM”，“类型”选中“ASCEDN”和“CPU”，按需选择规格，单击“立即注册”。

图 4-835 选择已上传的镜像源



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.47.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-836 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

4.47.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

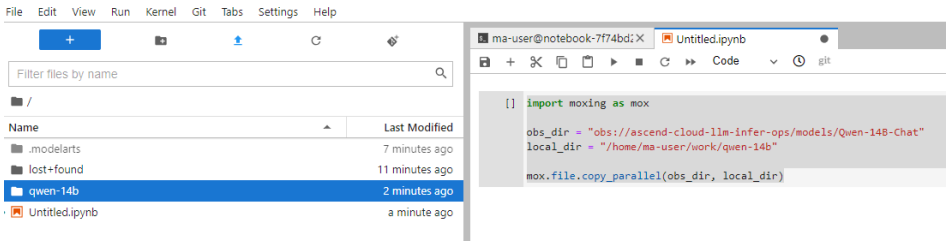
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-837 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。

2. 配置环境变量。

   ```
   export DEFER_DECODE=1
   # 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

   export DEFER_MS=10
   # 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

   export USE_VOCAB_PARALLEL=1
   ```

```
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

   说明

   以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

   - **通过vLLM服务API接口启动服务**

     在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.api_server --model ${model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --dtype=float16 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   - **通过OpenAI服务API接口启动服务**

     在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

     ```
     python -m vllm.entrypoints.openai.api_server --model ${model_path} \
     --max-num-seqs=256 \
     --max-model-len=4096 \
     --max-num-batched-tokens=4096 \
     --dtype=float16 \
     --tensor-parallel-size=1 \
     --block-size=128 \
     --host=${docker_ip} \
     --port=8080 \
     --gpu-memory-utilization=0.9 \
     --trust-remote-code
     ```

   具体参数说明如下：

   - --model ${model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。
   - --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
   - --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：/home/ma-user/work/chatglm3-6b/config.json。
   - --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。

- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
        "prompt": "hello",
        "max_tokens": 100,
        "temperature": 0,
        "ignore_eos": false,
        "presence_penalty":2
 }'
```

vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**

- 方式二：使用OpenAI接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/v1/chat/completions \
-H "Content-Type: application/json" \
```

```
-d '{
    "model": "${model_path}",
    "messages": [
        {
            "role": "user",
            "content": "hello"
        }
    ],
    "max_tokens": 100,
    "top_k": -1,
    "top_p": 1,
    "temperature": 0,
    "ignore_eos": false,
    "stream": false
}'
```

表 4-440 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.47.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**表4-436**。

  说明

  - 若需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。
  - 权重文件夹不要以"model"命名，若以"model"命名会导致后续创建AI应用报错。

- 推理启动脚本run_vllm.sh制作请参见**•创建推理脚本文件run_vllm.sh**。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-838 准备模型文件和权重文件

对象名称	存储类别	大小
cert.pem	标准存储	912 bytes
key.pem	标准存储	1.66 KB
run_vllm.sh	标准存储	458 bytes
chatglm3-6b	--	--

- **创建推理脚本文件run_vllm.sh**

  run_vllm.sh脚本示例如下。

  - **通过vLLM服务API接口启动服务**

    ```
    source /home/ma-user/.bashrc
    export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
    python -m vllm.entrypoints.api_server --model ${model_path} \
    --ssl-keyfile="/home/mind/model/key.pem" \
    --ssl-certfile="/home/mind/model/cert.pem" \
    --max-num-seqs=256 \
    --max-model-len=4096 \
    --max-num-batched-tokens=4096 \
    --dtype=float16 \
    --tensor-parallel-size=1 \
    --block-size=128 \
    --host=0.0.0.0 \
    --port=8080 \
    --gpu-memory-utilization=0.9 \
    --trust-remote-code
    ```

  - **通过OpenAI服务API接口启动服务**

    ```
    source /home/ma-user/.bashrc
    export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
    ```

```
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

参数说明：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。

  说明

  /home/mind/model路径为推理平台固定路径，部署服务时会将**Step1 准备模型文件和权重文件**OBS路径下的文件传输至/home/mind/model路径下。

- --tensor-parallel-size：并行卡数。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。

- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

Step2 部署模型

在ModelArts控制台的AI应用模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“资产管理 > AI应用 > 创建”，开始创建AI应用。
2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-839 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

说明

若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“模型部署 > 在线服务 > 部署”，开始部署在线服务。
2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-840 部署在线服务-专属资源池

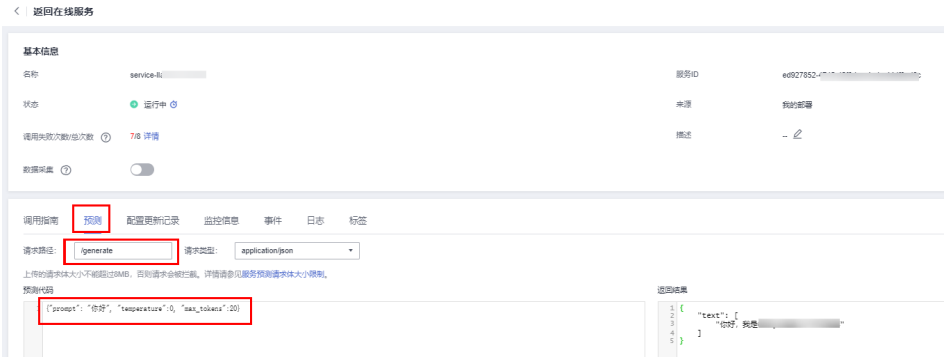
3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

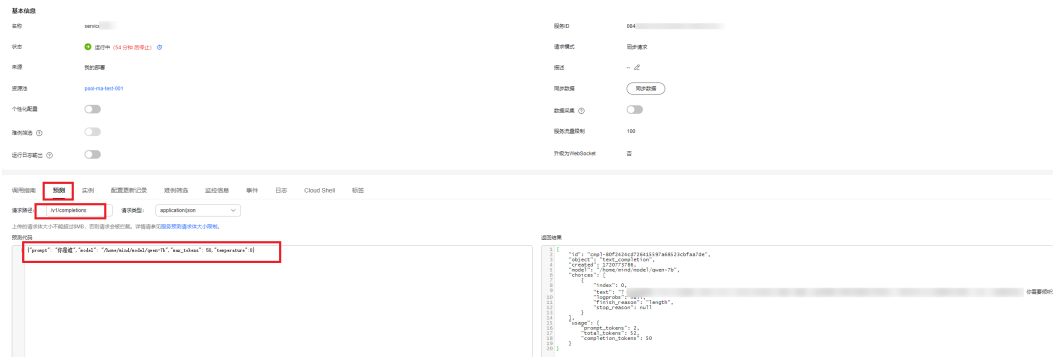
若以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。

图 4-841 预测-vllm



若以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”即可看到预测结果。

图 4-842 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

4.47.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh            #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
```

2. 确保Notebook内通网，已通网可以跳过这一步，未通网需要配置$config_proxy_str，$config_pip_str设置对应的代理和pip源，来确保当前代理和pip源可用。
3. 精度评测新建一个conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。命令中的$work_dir 是benchmark_eval的绝对路径。

```
conda activate python-3.9.10 #如果没有该conda环境需要手动建立一个
export work_dir=${work_dir} #指定work_dir路径
bash install.sh
```

4. 在benchmark_eval目录下安装依赖。

```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下 （ 可选，如果选择使用humaneval数据集 ）
pip install -e .  # 可选，如果选择使用humaneval数据集
```

5. （ 可选 ）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**6**进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                       exec(check_program, exec_globals)      #第58行
```

6. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（ 生成式回答是对整体进行评测，少一个字符就可能会导致判断错误 ）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen 。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：评测数据集类型，分为eval、static、awq，也就是精度、静态和量化数据集，默认eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

7. 客户端显示运行过程，通过run.py运行。如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

   ```
   cd opencompass
   python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
   ```

   output_path: 要保存的结果路径。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

4.47.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，若静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本
```

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

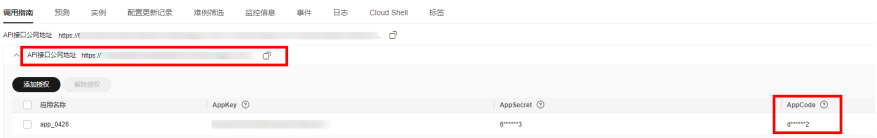
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-843 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。

- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-844 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_datasets.py脚本生成数据集方法：

   generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_datasets.py --datasets custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_datasets.py脚本执行参数说明如下：

   - --datasets：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。

- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   Notebook中进行测试：

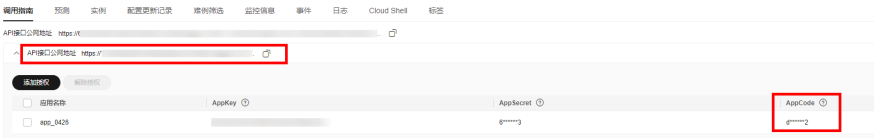
   ```
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-845 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --datasets：数据集路径。
   - --datasets-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。

- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-846 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.47.7 推理模型量化

4.47.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表4-436**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化，量化方法为per-group。

Step1 模型量化

可以在Huggingface开源社区获取AWQ量化后的模型权重；或者获取FP16/BF16的模型权重之后，通过autoAWQ工具进行量化。

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 在Notebook中运行以下命令下载并安装AutoAWQ源码。
   ```
   git clone -b v0.2.5 https://github.com/casper-hansen/AutoAWQ.git AutoAWQ-0.2.5
   cd ./AutoAWQ-0.2.5
   export PYPI_BUILD=1
   pip install -e .
   ```
2. 需要编辑“examples/quantize.py”文件，针对NPU进行如下适配工作，以支持在NPU上进行量化。
   a. 添加import。
      ```
      import torch_npu
      from torch_npu.contrib import transfer_to_npu
      ```
   b. 指定模型输入、输出路径。
      ```
      model_path = **
      quant_path = **
      ```
   c. 可以指定校准数据集路径，如calib_data="/path/to/pile-val"，如不指定，默认数据集是“mit-han-lab/pile-val-backup”。
      ```
      model.quantize(tokenizer, quant_config=quant_config, calib_data="/path/to/pile-val",
      split="validation")
      ```
3. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。

```
pip install transformers sentencepiece #安装量化工具依赖
export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
python examples/quantize.py
```

详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools代码目录下执行以下脚本：

执行时间预计10分钟。执行完成后会将权重路径下的原始权重替换成转换后的权重。如需保留之前权重格式，请在转换前备份。

```
python  awq/convert_awq_to_npu.py --model /home/ma-user/Qwen1.5-72B-Chat-AWQ
```

参数说明：

--model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

4.47.7.2 使用 SmoothQuant 量化工具转换权重

SmoothQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmoothQuant(W8A8)量化的模型列表请参见**表4-436**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

SmoothQuant量化工具使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_tools目录下。

代码目录如下:

```
AutoSmoothQuant  #量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                    # 安装量化模块的脚本
        ...
```

具体操作如下：

1. 配置环境。
   ```
   cd llm_tools/AutoSmoothQuant/
   sh build.sh
   ```
2. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。
3. 执行权重转换。

```
cd autosmoothquant/examples/
python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
```

参数说明:

- --model-path：原始模型权重路径。
- --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
- --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

4. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

4.47.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表4-436**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

使用tensorRT 0.9.0版本工具进行模型量化，工具下载使用指导请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/v0.9.0**。

执行如下脚本进行权重转换生成量化系数，详细参数解释请参见**https://github.com/NVIDIA/TensorRT-LLM/tree/main/examples/llama#int8-kv-cache** ）

```
python convert_checkpoint.py \
--model_dir ./llama-models/llama-7b-hf   \
--output_dir ./llama-models/llama-7b-hf/int8_kv_cache/ \
--dtype float16  \
--int8_kv_cache
```

运行完成后，会在output_dir下生成量化后的权重。量化后的权重包括原始权重和kvcache的scale系数。

Step2 抽取 kv-cache 量化系数

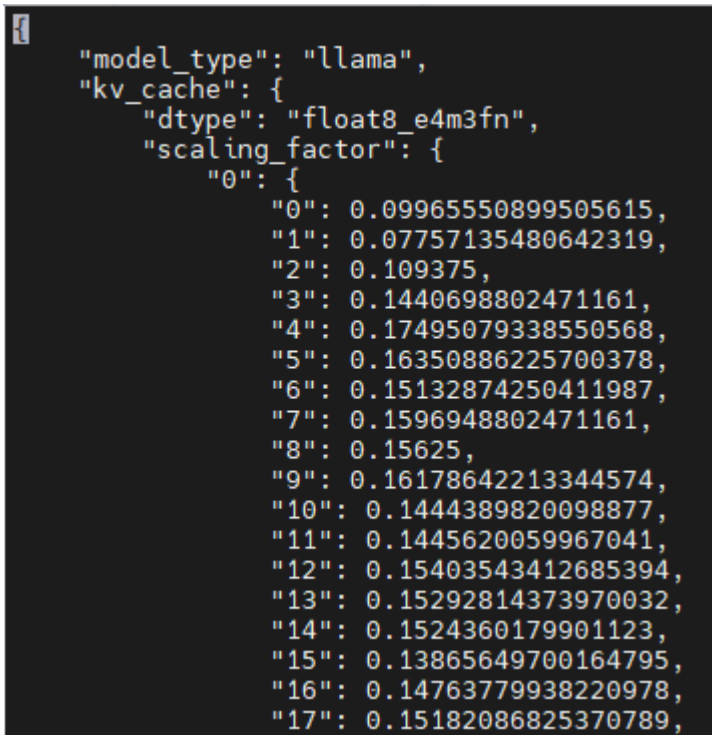
该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

```
python3 examples/fp8/extract_scales.py \
--quantized_model <QUANTIZED_MODEL_DIR> \
--tp_size <TENSOR_PARALLEL_SIZE> \
--output_dir <PATH_TO_OUTPUT_DIR>
```

运行后在 --output_dir下生成 kv_cache_scales.json文件，里面是提取的per-tensor的scale值。内容示例如下：

图 4-847 抽取 kv-cache 量化系数



注意：

1、抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。

2、当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

4.48 主流开源大模型基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.905）

4.48.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite Server上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表4-443**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如**表4-441**所示。

表 4-441 支持的模型

序号	支持模型	支持模型参数量
1	llama2	llama2-7b
2		llama2-13b
3		llama2-70b
4	llama3	llama3-8b
5		llama3-70b
6	Qwen	qwen-7b
7		qwen-14b
8		qwen-72b
9	Qwen1.5	qwen1.5-7b
10		qwen1.5-14b
11		qwen1.5-32b

序号	支持模型	支持模型参数量
12		qwen1.5-72b
13	Yi	yi-6b
14		yi-34b
15	ChatGLMv3	glm3-6b
16	Baichuan2	baichuan2-13b

操作流程

图 4-848 操作流程图

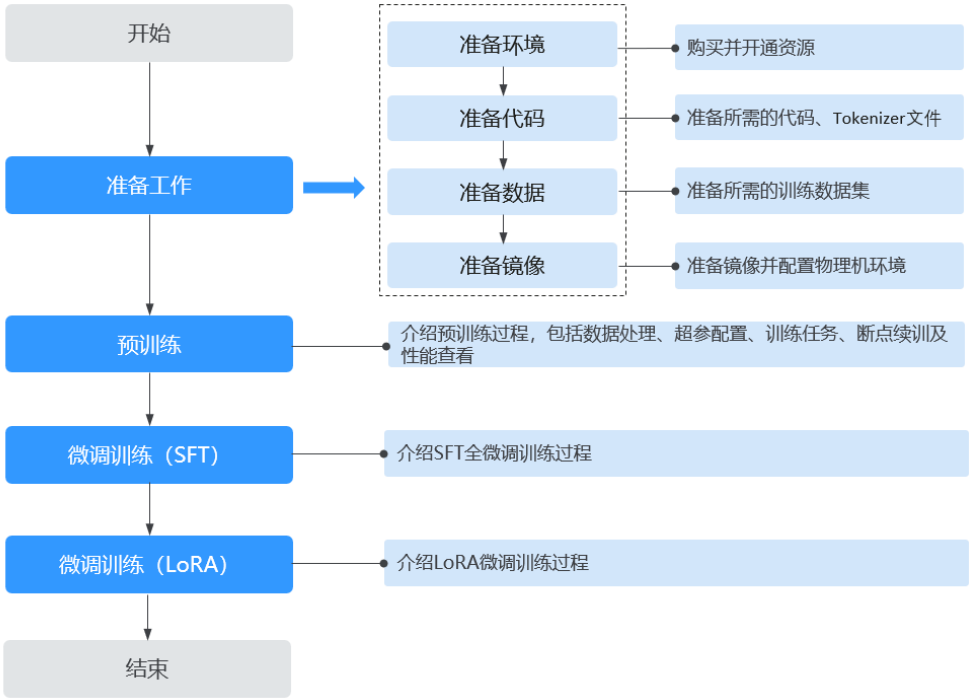


表 4-442 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite Server运行的，需要购买并开通Server资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。

阶段	任务	说明
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、断点续训及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调。
	LoRA微调训练	介绍如何进行LoRA微调训练。

4.48.2 准备工作

4.48.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的Server。请参考本文档要求准备资源环境。

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表4-451**。

硬盘空间：至少200GB。

Ascend资源规格：

- Ascend: 1*ascend-snt9b表示Ascend单卡。
- Ascend: 8*ascend-snt9b表示Ascend 8卡。

购买并开通资源

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

4.48.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表4-443**所示，模型列表、对应的开源权重获取地址如**表4-444**所示。

表 4-443 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

表 4-444 支持的模型类型和权重获取地址

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat

模型软件包结构说明

AscendCloud-3rdLLM代码包结构介绍如下：

```
|──llm_train                # 模型训练代码包
        |──AscendSpeed        # 基于AscendSpeed的训练代码
            |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |──scripts/             # 训练需要的启动脚本
                |──llama2           # llama2系列模型执行脚本的文件夹
                |──llama3           # llama3系列模型执行脚本的文件夹
                |──qwen             # Qwen系列模型执行脚本的文件夹
                |──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                |── ...
                |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                |── install.sh       # 环境部署脚本
|──llm_inference                  # 推理代码包
|──llm_tools                     # 推理工具
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。训练脚本以分类的方式集中在 scripts 文件夹中。

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed             # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
# 数据目录结构
|── processed_for_input          #目录结构会自动生成，无需用户创建
    |── ${model_name}             # 模型名称
        |── data              # 预处理后数据
            |── pretrain    # 预训练加载的数据
            |── finetune    # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换megatron格式后权重文件
|── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |── ${model_name}             # 模型名称
        |── logs              # 训练过程中日志（loss、吞吐性能）
            |── saved_models
            |── lora              # lora微调输出权重
            |── sft               # 增量训练输出权重
            |── pretrain          # 预训练输出权重
|── tokenizers                    #原始权重及tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                 #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录Server。
2. 将AscendCloud代码包AscendCloud-3rdLLM-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。
   ```
   unzip AscendCloud-3rdLLM-*.zip
   ```
3. 上传代码之后需要修改llm_train/AscendSpeed/scripts/install.sh文件。具体为删除install.sh 的第43行 "git cherrypick 171ba0b3"。该问题会导致代码安装失败，会在后续版本修复。
4. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Llama2-70B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Llama2-70B
   ```

4.48.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key 标志来选择用于训练的列。

```
{
    'id': '1',
    'url': 'https://simple.wikipedia.org/wiki/April',
    'title': 'April',
    'text': 'April is the fourth month...'
}
```

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。
```
mkdir training_data
```
数据存放参考目录结构如下：
```
${workdir}（例如/home/ma-user/ws）
 |── training_data
     |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
     |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.48.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-445 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0

表 4-446 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
```
npu-smi info                  # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```

4.48.3 预训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-447**所示。其他超参均有默认值，可以参考**表4-450**按照实际需求修改。

表 4-447 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

请根据**Step2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为8机64卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  1
...
...
# 第八台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  7
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NODE_RANK、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
示例:
sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 4-849 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

4.48.4 SFT 全参微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-447**所示。其他超参均有默认值，可以参考**表4-450**按照实际需求修改。

表 4-448 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8>
<NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  1
# 第三台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  2
```

```
# 第四台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
```

训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

4.48.5 LoRA 微调训练

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-447**所示。其他超参均有默认值，可以参考**表4-450**按照实际需求修改。

表 4-449 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8> <NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  1
# 第三台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  2
# 第四台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
```

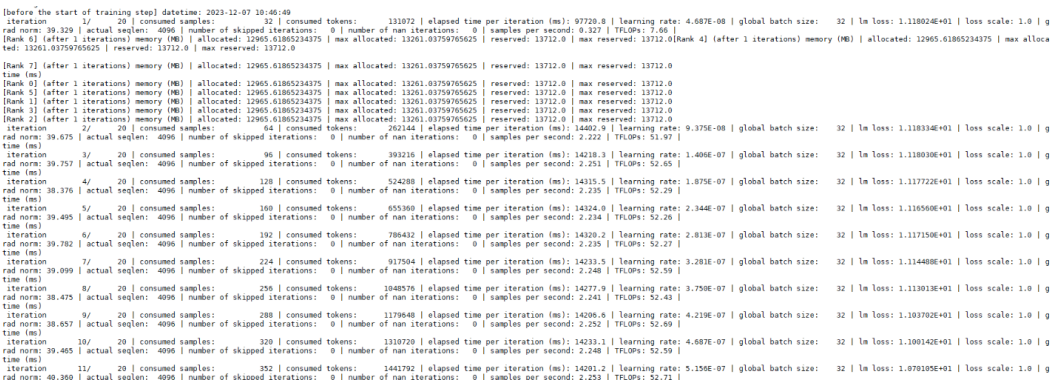
训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

4.48.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 4-850 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

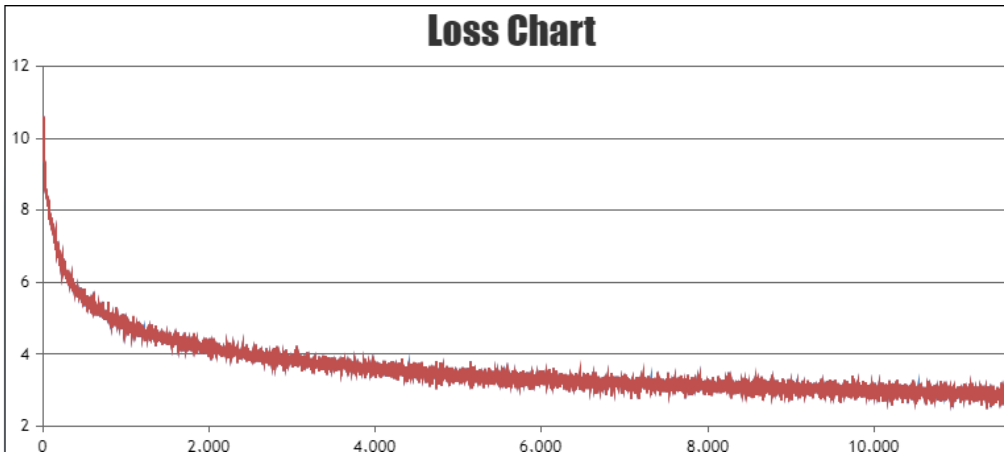
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表4-450**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图4-851**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 4-851 Loss 收敛情况（示意图）



4.48.7 训练脚本说明

4.48.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例：

表 4-450 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-451所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-451 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

4.48.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/finetune/**

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/1_preprocess_data.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-452 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.48.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 4-453 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.48.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图4-852所示。

图 4-852 修改 ChatGLMv3-6B tokenizer 文件（1）

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图4-853所示。

图 4-853 修改 ChatGLMv3-6B tokenizer 文件（2）

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图4-854所示。

图 4-854 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.49 主流开源大模型基于 LIte Server 适配 PyTorch NPU 推理指导（6.3.905）

4.49.1 推理场景介绍

方案概览

本方案介绍了在ModelArts Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.3.2。
- 支持FP16和BF16数据类型推理。

资源规格要求

本文档中的模型运行环境是ModelArts Lite的Server。推荐使用“西南-贵阳一”Region上的资源和Ascend Snt9B。

如果使用Server资源，请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-454**所示。

表 4-454 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的vLLM 0.3.2推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	6.3.905版本获取路径：**Support-E**（推荐） **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
AscendCloud-OPP-6.3.905-xxx.zip	推理依赖的算子包。	

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-455 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	Cann版本
6.3.905版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	cann_8.0.rc2

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

支持的模型列表和权重文件

本方案支持vLLM的v0.3.2版本。不同vLLM版本支持的模型列表有差异，具体如**表4-456**所示。

表 4-456 支持的模型列表和权重获取地址

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
1	llama-7b	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	**https://huggingface.co/huggyllama/llama-13b**

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
3	llama-65b	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
21	qwen1.5-14b	√	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	baichuan2-7b	√	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
26	baichuan2-13b	√	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
27	chatglm2-6b	√	https://huggingface.co/THUDM/chatglm2-6b
28	chatglm3-6b	√	https://huggingface.co/THUDM/chatglm3-6b
29	gemma-2b	√	https://huggingface.co/google/gemma-2b
30	gemma-7b	√	https://huggingface.co/google/gemma-7b
31	mistral-7b	√	https://huggingface.co/mistralai/Mistral-7B-v0.1

模型软件包结构说明

本教程需要使用到的AscendCloud-3rdLLM-xxx.zip软件包中的关键文件介绍如下。

```
├──llm_tools            #推理工具包
  ├──llm_evaluation  #推理评测代码包
    ├──benchmark_eval   # 精度评测
      ├── config
        ├── config.json  # 请求的参数，根据实际启动的服务来调整
        ├── mmlu_subject_mapping.json  # 数据集配置
        ├── ...
      ├── evaluators
        ├── evaluator.py # 数据集数据预处理方法集
        ├── model.py  # 发送请求的模块，在这里修改请求响应。目前支持vllm.openai，atb的tgi模板
        ├── ...
      ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
      ├── service_predict.py  # 发送请求的服务。支持vllm的openai，atb的tgi模板
      ├── ...
    ├──benchmark_tools  #性能评测
      ├── benchmark.py    # 可以基于默认的参数跑完静态benchmark和动态benchmark
      ├── benchmark_parallel.py  # 评测静态性能脚本
      ├── benchmark_serving.py   # 评测动态性能脚本
      ├── benchmark_utils.py     # 抽离的工具集
      ├── generate_datasets.py    # 生成自定义数据集的脚本
```

```
        ├── requirements.txt       # 第三方依赖
        ├── ...
├──llm_inference  #推理代码
        ├── ascend_vllm_adapter  #昇腾vLLM使用的算子模块
        ├── ascend.txt   #基于开源vLLM适配过NPU的patch脚本
        ├── autosmoothquant_ascend.txt   #基于开源autosmoothquant适配过NPU的patch脚本
        ├── build.sh     #推理构建脚本
        ├── requirements.txt       # 第三方依赖
```

相关文档

和本文档配套的模型训练文档请参考《**主流开源大模型基于Lite Server适配PyTorch NPU训练指导**》。

4.49.2 部署推理服务

本章节介绍如何使用vLLM 0.3.2框架部署并启动推理服务。

前提条件

- 已准备好Server环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的Server和昇腾Snt9b资源。
- 确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

2. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表4-455**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-3rdLLM-xxx.zip和算子包AscendCloud-OPP-xxx.zip到容器中，包获取路径请参见**表4-454**。
2. 将权重文件上传到Server机器中。权重文件的格式要求为Huggingface格式。开源权重文件获取地址请参见**表4-456**。

Step4 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- -v ${dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，dir为宿主机中文件目录，${container_work_dir}为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- {image_id} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

Step5 进入容器安装推理依赖软件

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
3. 解压算子包并将相应算子安装到环境中。
   ```
   unzip AscendCloud-OPP-*.zip
   pip install ascend_cloud_ops-1.0.0-py3-none-any.whl
   pip install cann_ops-1.0.0-py3-none-any.whl
   ```

4. 解压软件推理代码并安装依赖包。
```
unzip AscendCloud-3rdLLM-*.zip
cd llm_inference
pip install -r requirements.txt
```
5. 运行推理构建脚本build.sh文件，会自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。
```
cd llm_inference
bash build.sh
```
运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

Step6 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```
说明

NPU卡编号可以通过命令npu-smi info查询。
2. 配置PYTHONPATH。
```
export PYTHONPATH=$PYTHONPATH:${vllm_path}
```
${vllm_path} 填写ascend_vllm文件夹绝对路径。
3. 高阶配置（可选）。
   a. 词表切分。

      在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量：
```
export USE_VOCAB_PARALLEL=1   #打开词表切分开关
unset USE_VOCAB_PARALLEL      #关闭词表切分开关
```
      配置后重启服务生效。
   b. Matmul_all_reduce融合算子。

      使用Matmul_all_reduce融合算子能提升全量推理性能；该算子要求驱动和固件版本为Ascend HDK 24.1.RC1.B011及以上，默认不开启。如需开启，配置以下环境变量：
```
export USE_MM_ALL_REDUCE_OP=1   #打开Matmul_all_reduce融合算子
unset  USE_MM_ALL_REDUCE_OP     #关闭Matmul_all_reduce融合算子
```
      配置后重启服务生效。
   c. 查看详细日志。

      查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量：
```
export DETAIL_TIME_LOG=1   #打开打印详细日志
export RAY_DEDUP_LOGS=0    #打开打印详细日志
unset  DETAIL_TIME_LOG     #关闭打印详细日志
```
      配置后重启服务生效。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **通过vLLM服务API接口启动服务**

  在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。如果使用的是训练后模型转换为HuggingFace格式的地址，还需要有Tokenizer原始文件。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。

- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表4-457**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。${container_model_path}请替换为实际使用的模型名称。

```
curl -X POST http://${docker_ip}:8080/v1/chat/completions \
-H "Content-Type: application/json" \
-d '{
   "model": "${container_model_path}",
   "messages": [
      {
         "role": "user",
         "content": "hello"
      }
   ],
   "max_tokens": 100,
   "top_k": -1,
   "top_p": 1,
   "temperature": 0,
   "ignore_eos": false,
   "stream": false
}'
```

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http:/${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 4-457 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.3.2）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9，qwen系列、qwen1.5系列、llama3系列模型还需打开词表切分配置export USE_VOCAB_PARALLEL=1。

序号	模型名称	4*64GB	8*32GB
1	qwen1.5-72b	24576	8192
2	qwen-72b	24576	8192
3	llama3-70b	32768	8192
4	llama2-70b	98304	32768
6	llama-65b	24576	8192

序号	模型名称	2*64GB	4*32GB
1	qwen1.5-32b	65536	24576

序号	模型名称	1*64GB	1*32GB
1	qwen1.5-7b	49152	16384
2	qwen-7b	49152	16384
3	llama3-8b	98304	32768
4	llama2-7b	126976	16384
5	chatglm3-6b	126976	65536
6	chatglm2-6b	126976	65536

序号	模型名称	1*64GB	2*32GB
1	qwen1.5-14b	24576	24576
2	qwen-14b	24576	24576
3	llama2-13b	24576	24576

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

4.49.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-3rdLLM-xxx.zip的llm_tools/llm_evaluation（6.3.905版本）目录中。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
```

```
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本、
├── requirements.txt      # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-3rdLLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，执行如下命令安装性能测试的关依赖。
   ```
   pip install -r requirements.txt
   ```
3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   cd benchmark_tools
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```
   参数说明
   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存路径，如benchmark_parallel.csv。
4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-855 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

本章节介绍如何进行动态benchmark验证。

1. 获取数据集。动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   方法二：使用generate_dataset.py脚本生成数据集方法：

   generate_dataset.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python generate_dataset.py --dataset custom_datasets.json --tokenizer /path/to/tokenizer \
   --min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
   --min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
   ```

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。

- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python benchmark_serving.py --backend vllm --host${docker_ip} --port 8085 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如"tgi"，vllm"，"mindspore"
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务端口
- --dataset：数据集路径
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是huggingface的权重路径
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer
- --benchmark-csv：结果保存路径，如benchmark_serving.csv

3. 脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-856 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

4.49.4 推理精度测试

本章节介绍如何进行推理精度测试。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-3rdLLM-xxx.zip的llm_tools/llm_evaluation（6.3.905版本）目录中。代码目录结构如下。精度测试使用到的mmlu和ceval数据集已经提前打包在代码中。

```
benchmark_eval
├──apig_sdk       # ma校验包
├──cpu_npu        # 检测资源消耗
```

```
├── config
│   ├── config.json  # 服务的配置模板，已配置了ma-standard，tgi示例
│   ├── mmlu_subject_mapping.json  # mmlu数据集学科信息
│   ├── ceval_subject_mapping.json  # ceval数据集学科信息
├── evaluators
│   ├── evaluator.py # 数据集数据预处理方法集
│   ├── chatglm.py  # 处理请求相应模块, 一般和chatglm的官方评测数据集ceval搭配
│   ├── llama.py  # 处理请求相应模块, 一般和llama的评测数据集mmlu搭配
├── mmlu-exam, mmlu数据集
├── ceval-exam, ceval数据集
├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
├── readme.md     # 说明文档
├── requirements.txt   # 第三方依赖
├── service_predict.py  # 发送请求的服务
```

2. 上传精度测试代码到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-3rdLLM-*x.x.x*.zip并解压，无需重复执行。
3. 进入benchmark_eval目录下，执行如下命令安装性能测试的关依赖。

```
pip install -r requirements.txt
```

4. 执行精度测试启动脚本eval_test.py，具体操作命令如下，可以根据参数说明修改参数。

```
python eval_test.py \
 --max_workers=1 \
 --service_name=llama2-13b-chat-test \
 --eval_dataset=ceval \
 --service_url=http://${docker_ip}:8080/v1/completions \
 --few_shot=3 \
 --is_devserver=True \
 --model_name=llama2 \
 --deploy_method=vllm \
 --vllm_model=${model_path}
```

参数说明:

- max_workers：请求的最大线程数，默认为1。
- service_name：服务名称，保存评测结果时创建目录，示例为：llama2-13b-chat-test。
- eval_dataset：评测使用的评测集（枚举值），目前仅支持mmlu、ceval。
- service_url：成功部署推理服务后的服务预测地址，示例：http://${docker_ip}:8080/generate。此处的${docker_ip}替换为宿主机实际的IP地址，端口号8080来自前面配置的服务端口。
- few_shot：开启少量样本测试后添加示例样本的个数。默认为3，取值范围为0~5整数。
- is_devserver： 是否Server部署方式，True表示Server模式。False表示ModelArts Standard模式。
- model_name：评测模型名称，llama2。
- deploy_method：部署方法，不同的部署方式api参数输入、输出解析方式不同，目前支持tgi、ma_standard、vllm等方式。
- vllm_model：deploy_method为vllm时，服务以openai的方式启动，vllm_model为启动服务时传入的model_path。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{service_name}/{eval_dataset}-{timestamp} 的目录结果保存到对应的测试工程。执行多少次，则会在{service_name}下生成多少次结果。

单独的评测结果如下：


```
{eval_dataset}-{timestamp}  # 例如: mmlu-20240205093257
├── accuracy
│   ├── evaluation_accuracy.xlsx  # 测试的评分结果，包含各个学科数据集的评分和总和评分。
├── infer_info
│   ├── xxx1.csv # 单个数据集的评测结果
│   ├── ......
│   ├── xxxn.csv # 单个数据集的评测结果
├── summary_result
│   ├── answer_correct.xlsx  # 回答正确的结果
│   ├── answer_error.xlsx  # 保存回答了问题的选项，但是回答结果错误
│   ├── answer_result_unknow.xlsx  # 保存未推理出结果的问题，例如超时、系统错误
│   ├── system_error.xlsx  # 保存推理结果，但是可能答非所问，无法判断是否正确，需要人工判断进行纠偏。
```

4.49.5 附录：大模型推理常见问题

问题1：在推理预测过程中遇到NPU out of memory

解决方法：调整推理服务启动时的显存利用率，将--gpu-memory-utilization的值调小。

问题2：在推理预测过程中遇到ValueError:User-specified max_model_len is greater than the drived max_model_len

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

4.50 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.905）

4.50.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Standard上的预训练和全量微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

约束限制

本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如**表4-458**所示。

表 4-458 代码包中适配的模型

序号	支持模型	支持模型参数量	权重文件获取地址
1	Llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	Llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**

操作流程

图 4-857 操作流程图

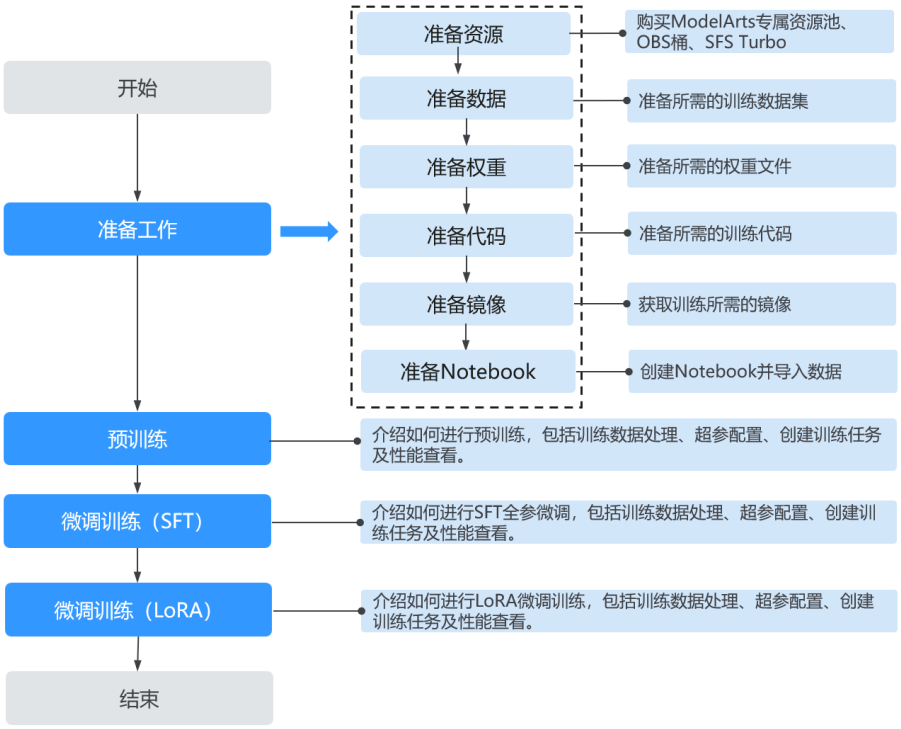


表 4-459 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

4.50.2 准备工作

4.50.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表4-466**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统，详细操作指导请参考**创建SFS Turbo文件系统**。

图 4-858 创建 SFS Turbo

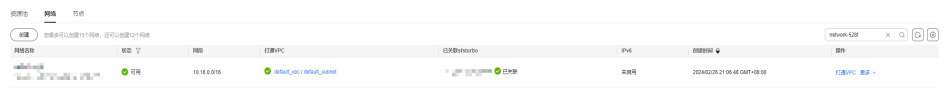
其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 4-859 SFS 类型和容量选择

ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。 具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 4-860 ModelArts 网络关联 SFS Turbo



4.50.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

Alpaca数据集是由OpenAI的text-davinci-003引擎生成的包含52k条指令和演示的数据集。这些指令数据可以用来对语言模型进行指令调优，使语言模型更好地遵循指令。

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

请注意huggingface中的数据集具有如下**this**格式。可以使用–json-key标志更改数据集文本字段的名称，默认为text。在维基百科数据集中，它有四列，分别是id、url、title和text。可以指定–json-key标志来选择用于训练的列。

```
{
  'id': '1',
  'url': 'https://simple.wikipedia.org/wiki/April',
  'title': 'April',
  'text': 'April is the fourth month...'
}
```

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

4.50.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-458**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
```

```
├── tokenizer.model
└── USE_POLICY.md
```

4.50.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表4-460**所示。

表 4-460 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-3rdLLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
├──llm_train                # 模型训练代码包
        ├──AscendSpeed         # 基于AscendSpeed的训练代码
              ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              ├──scripts/             # 训练需要的启动脚本
                   ├──llama2          # llama2系列模型执行脚本的文件夹
                   ├──llama3          # llama3系列模型执行脚本的文件夹
                   ├──qwen            # Qwen系列模型执行脚本的文件夹
                   ├──qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   ├── ...
                   ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   ├── install.sh       # 环境部署脚本
├──llm_inference                # 推理代码包
├──llm_tools                    # 推理工具
```

注意

下载代码之后需要修改llm_train/AscendSpeed/scripts/install.sh文件。具体为删除install.sh的第43行 "git cherrypick 171ba0b3"。该问题会导致代码安装失败，会在后续版本修复。

代码上传至 OBS

将AscendSpeed代码包AscendCloud-3rdLLM-905-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                           #微调数据文件
|── model                                          #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
```

4.50.2.5 准备镜像

准备训练Llama2-13B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-461 基础容器镜像地址

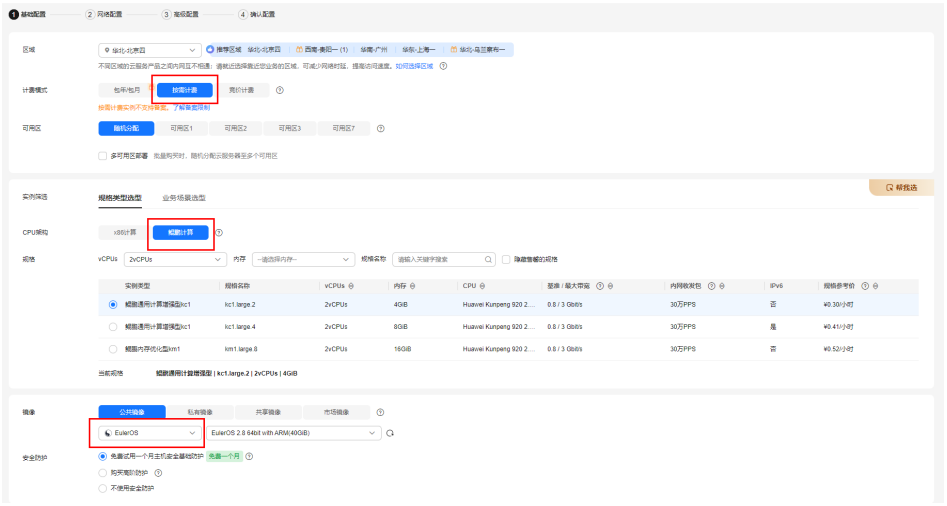
镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-861 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

图 4-862 创建镜像组织

Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-863 复制登录指令

Step5 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表4-461**。

```
docker pull {image_url}
```

Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240528

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240528

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

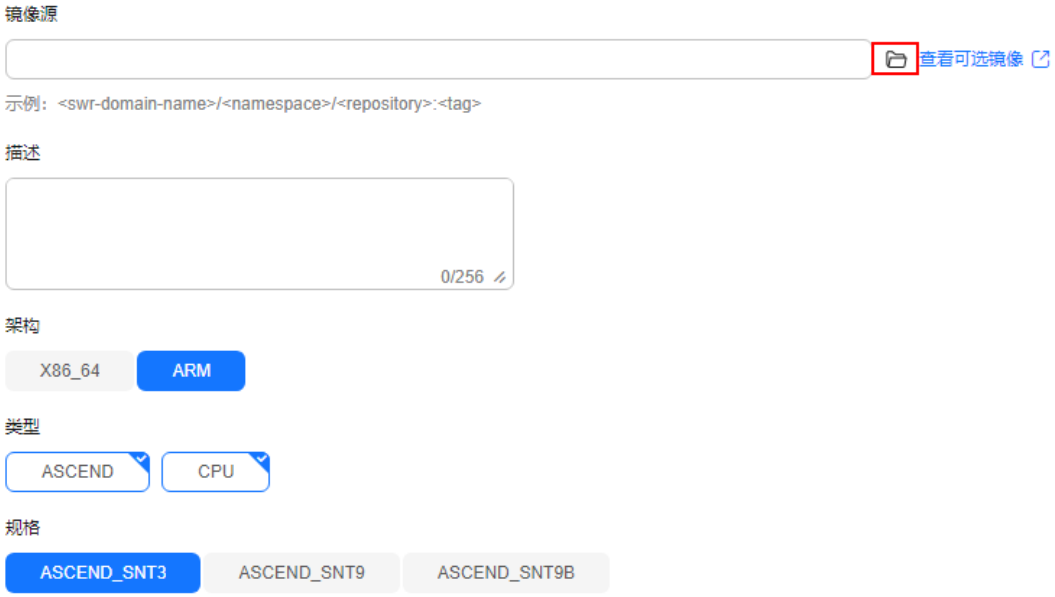
docker push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240528

Step7 ModelArts 中注册镜像

镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。

1. 登录ModelArts管理控制台，在左侧导航栏选择“资产管理 > 镜像管理”，然后在“镜像管理”页面右上角单击“注册镜像”。
2. 在“注册镜像”页面，选择已上传的镜像源，“架构”选择“ARM”，“类型”选中“ASCEDN”和“CPU”，按需选择规格，单击“立即注册”。

图 4-864 选择已上传的镜像源



4.50.2.6 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 4-865 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例。如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 4-866 Notebook 中选择弹性文件服务

使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包，每次启动训练作业时会执行该命令安装。

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

由于训练启动命令也会执行sh scripts/install.sh安装依赖包，因此Notebook保存镜像为可选操作。

图 4-867 安装依赖包

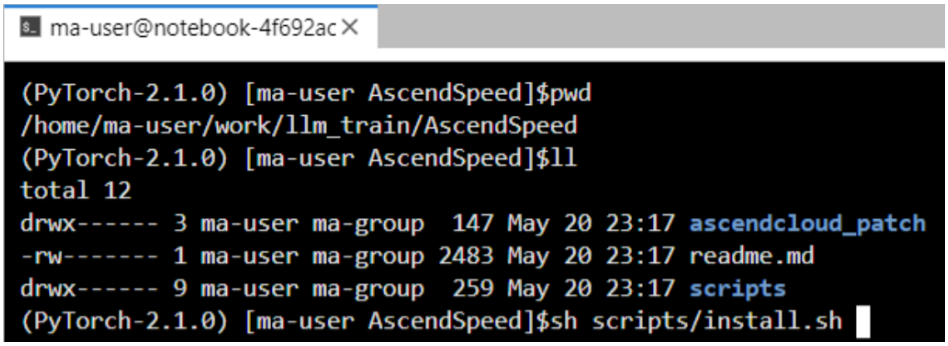


图 4-868 保存镜像

图 4-869 填写保存镜像相关参数

4.50.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-462**所示。其他超参均有默认值，可以参考**表4-465**按照实际需求修改。

表 4-462 必须修改的训练超参配置

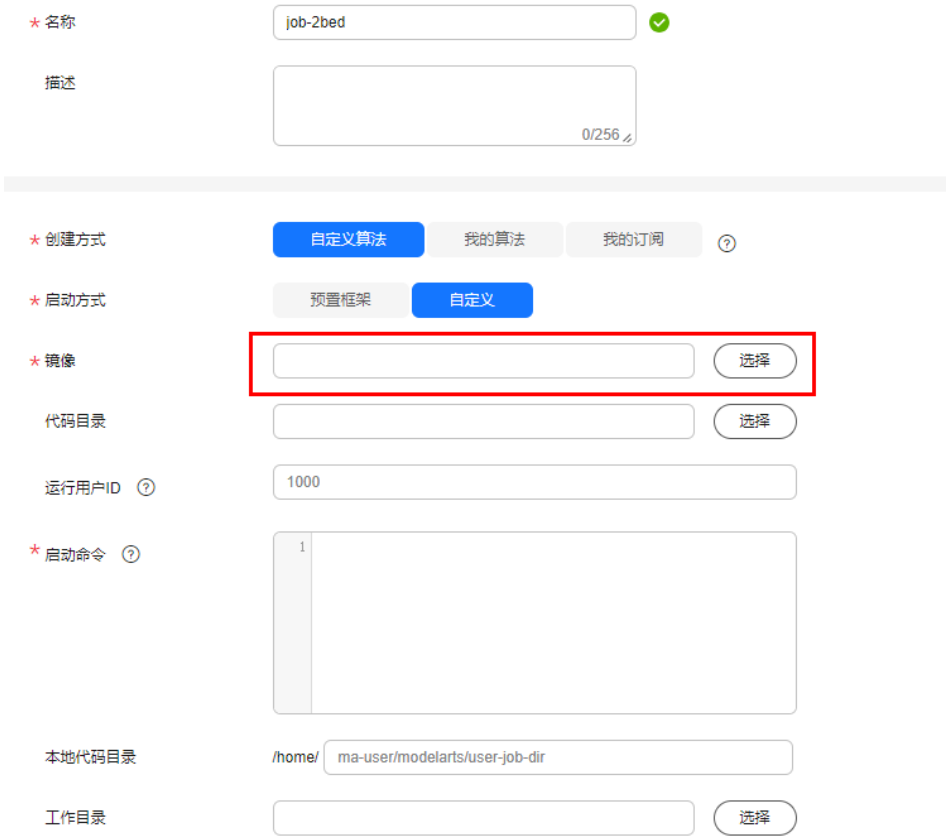
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-870 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

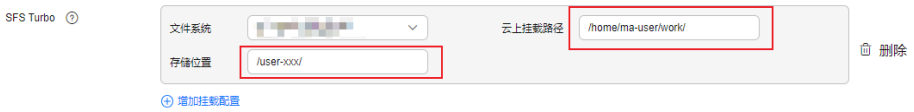
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-466**进行配置。

图 4-871 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-872 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.50.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-463**所示。其他超参均有默认值，可以参考**表4-465**按照实际需求修改。

表 4-463 必须修改的训练超参配置

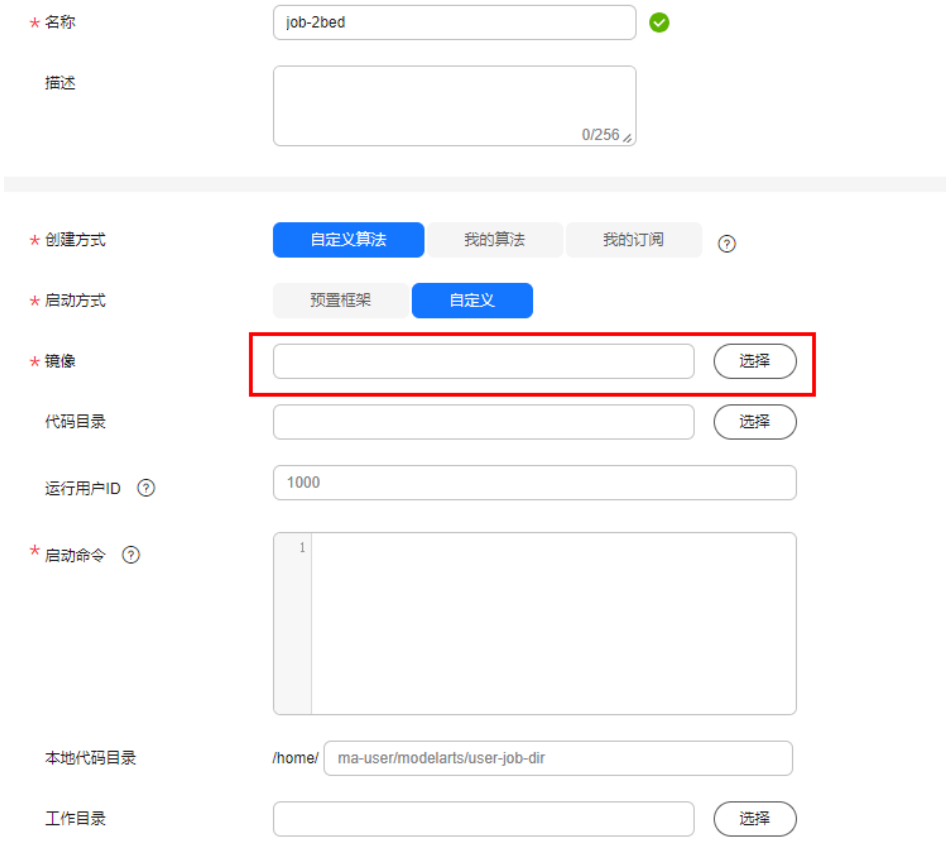
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-873 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

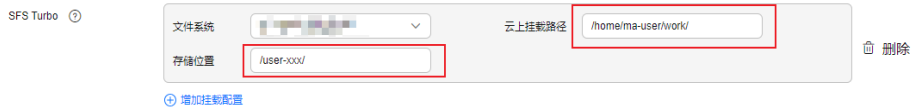
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-466**进行配置。

图 4-874 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-875 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.50.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表4-464**所示。其他超参均有默认值，可以参考**表4-465**按照实际需求修改。

表 4-464 必须修改的训练超参配置

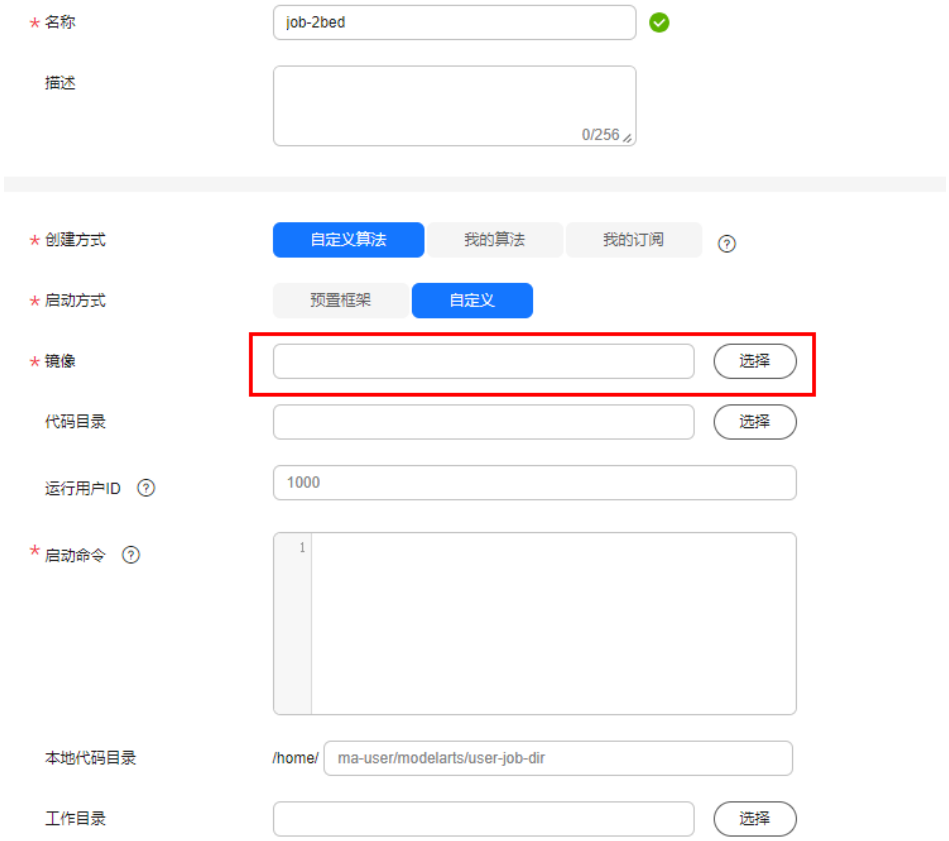
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 4-876 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

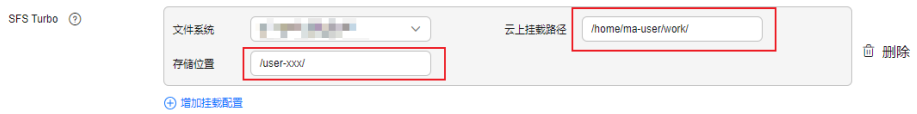
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表4-466**进行配置。

图 4-877 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 4-878 选择 SFS Turbo



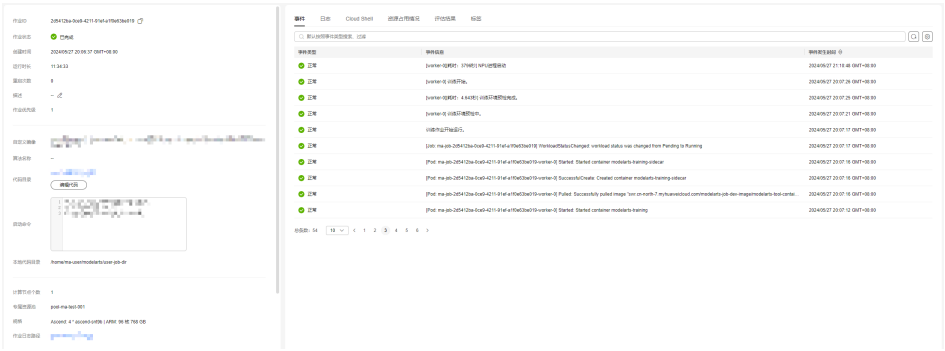
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

4.50.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

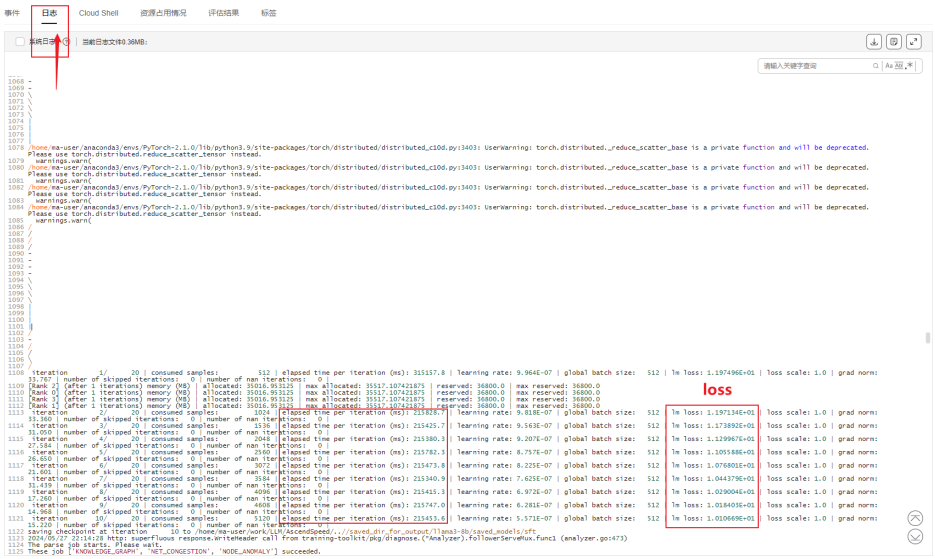
图 4-879 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数 ，lm loss的值随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 4-880 查看日志和性能



4.50.7 训练脚本说明

4.50.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 4-465 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler]	示例值需要根据数据集的不同，选择其一。 GeneralPretrainHandler：使用预训练的alpaca数据集； GeneralInstructionHandler：使用微调的alpaca数据集；
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表4-466所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 4-466 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN =4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
2		llama2-13b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN =4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
15	ChatGLM v3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

4.50.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理时，要执行的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认值。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理时，要执行的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/finetune/**

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在**Notebook**中运行该脚本。其中环境变量详细介绍如下：

表 4-467 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

4.50.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在**Notebook**运行该脚本。其中环境变量详细介绍如下：

表 4-468 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

4.50.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图4-881所示。

图 4-881 修改 ChatGLMv3-6B tokenizer 文件（1）

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图4-882所示。

图 4-882 修改 ChatGLMv3-6B tokenizer 文件（2）

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图4-883所示。

图 4-883 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

4.51 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.905）

4.51.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

约束限制

- 推理部署使用的服务框架是vLLM（官网地址：https://github.com/vllm-project/vllm/tree/v0.3.2，版本：v0.3.2）。
- 仅支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持的模型列表、对应的开源权重获取地址如表4-469所示。

表 4-469 支持的模型列表和权重获取地址

序号	支持模型	支持模型参数量	开源权重获取地址
1	Llama	llama-7b	https://huggingface.co/huggyllama/llama-7b
2		llama-13b	https://huggingface.co/huggyllama/llama-13b
3		llama-65b	https://huggingface.co/huggyllama/llama-65b
4	Llama 2-	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

序号	支持模型	支持模型参数量	开源权重获取地址
6		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
7	Llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
8		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
9	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
10		yi-9b	**https://huggingface.co/01-ai/Yi-9B**
11		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
12	Deepseek	deepseek-llm-7b	**https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat**
13		deepseek-coder-instruct-33b	**https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct**
14		deepseek-llm-67b	**https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat**
15	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
16		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
17		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
18	Qwen1.5	qwen1.5-0.5b	**https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat**
19		qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
20		qwen1.5-1.8b	**https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat**
21		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
22		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B/tree/main**

序号	支持模型	支持模型参数量	开源权重获取地址
23	Baichuan	qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24		qwen1.5-110b	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25		baichuan2-7b	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
26		baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
27	ChatGLMv2	chatglm2-6b	https://huggingface.co/THUDM/chatglm2-6b
28		chatglm3-6b	https://huggingface.co/THUDM/chatglm3-6b
29	Gemma	gemma-2b	https://huggingface.co/google/gemma-2b
30		gemma-7b	https://huggingface.co/google/gemma-7b
31	Mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-v0.1

操作流程

图 4-884 操作流程图

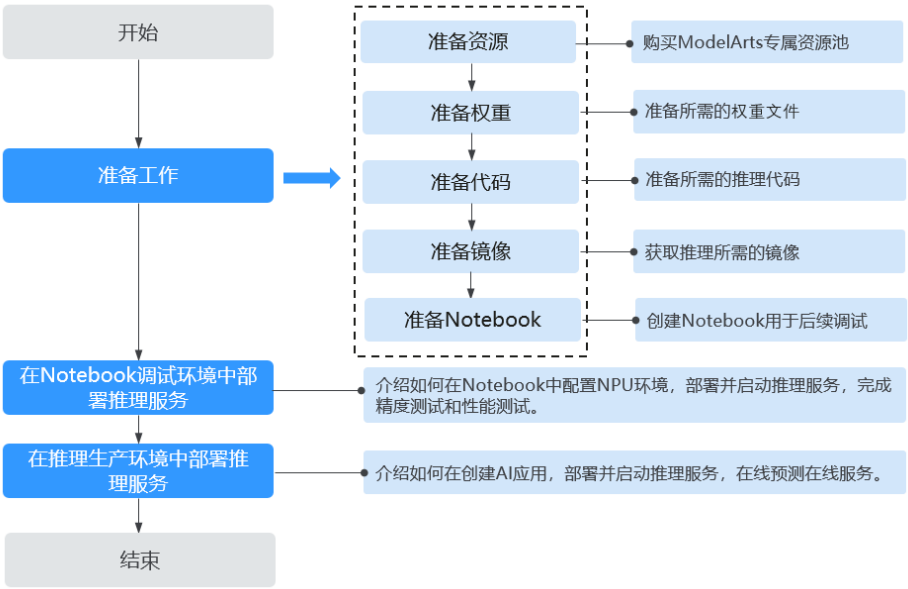


表 4-470 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-3rdLLM-6.3.905-xxx.zip和AscendCloud-OPP-6.3.905-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

4.51.2 准备工作

4.51.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

4.51.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**表4-469**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
```

```
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

4.51.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取配套版本

本方案支持的软件配套版本和依赖包获取地址如**表4-471**所示。

表 4-471 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的vLLM 0.3.2推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
AscendCloud-OPP-6.3.905-xxx.zip	推理依赖的算子包。	

模型软件包结构说明

本教程需要使用到的AscendCloud-3rdLLM-xxx.zip软件包中的关键文件介绍如下。

```
├──llm_tools          #推理工具包
  ├──llm_evaluation  #推理评测代码包
    ├──benchmark_eval   # 精度评测
      ├── config
          ├── config.json  # 请求的参数，根据实际启动的服务来调整
          ├── mmlu_subject_mapping.json  # 数据集配置
          ├── ...
      ├── evaluators
          ├── evaluator.py # 数据集数据预处理方法集
          ├── model.py  # 发送请求的模块，在这里修改请求响应。目前支持vllm.openai，atb的tgi模板
          ├── ...
      ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
      ├── service_predict.py  # 发送请求的服务。支持vllm的openai，atb的tgi模板
      ├── ...
    ├──benchmark_tools  #性能评测
      ├── benchmark.py    # 可以基于默认的参数跑完静态benchmark和动态benchmark
      ├── benchmark_parallel.py  # 评测静态性能脚本
      ├── benchmark_serving.py   # 评测动态性能脚本
      ├── benchmark_utils.py     # 抽离的工具集
      ├── generate_datasets.py    # 生成自定义数据集的脚本
```

```
        ├── requirements.txt       # 第三方依赖
        ├── ...
├──llm_inference  #推理代码
        ├── ascend_vllm_adapter  #昇腾vLLM使用的算子模块
        ├── ascend.txt   #基于开源vLLM适配过NPU的patch脚本
        ├── autosmoothquant_ascend.txt   #基于开源autosmoothquant适配过NPU的patch脚本
        ├── build.sh     #推理构建脚本
        ├── requirements.txt       # 第三方依赖
```

4.51.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-472 基础容器镜像地址

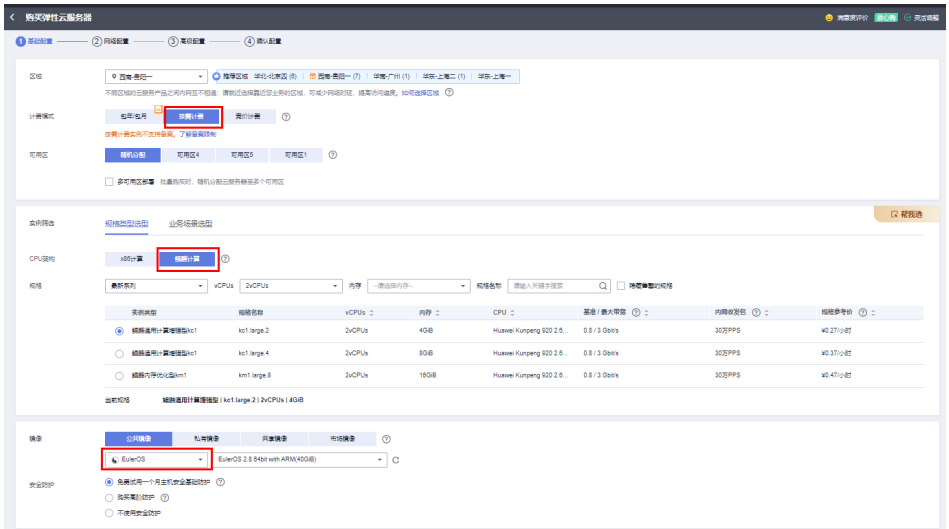
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 4-885 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step3 创建镜像组织

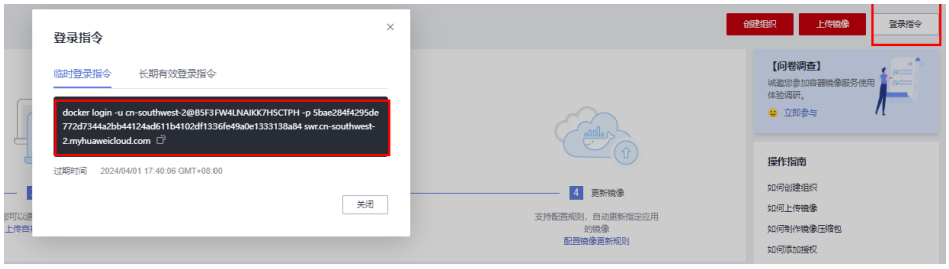
在SWR服务页面创建镜像组织。

图 4-886 创建镜像组织

Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-887 复制登录指令



Step5 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step6 构建 ModelArts Standard 推理镜像

获取模型软件包和依赖包，并上传到ECS的目录下（可自定义路径），获取地址参考**表4-471**。

在ModelArts官方提供的基础镜像上，构建一个用于ModelArts Standard推理部署的镜像。

在模型软件包和依赖包的同层目录下，创建并编辑Dockerfile。

```
vim Dockerfile
```

Dockerfile内容如下：

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0

USER root
COPY AscendCloud-*.zip /home/ma-user/
RUN unzip -o /home/ma-user/AscendCloud-3rdLLM-6.3.905-*.zip
RUN unzip -o /home/ma-user/AscendCloud-OPP-6.3.905-*.zip

RUN chmod 755 /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN pip install -r /home/ma-user/llm_inference/requirements.txt
RUN chmod -R 755 /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages
ENTRYPOINT sh /home/mind/model/run_vllm.sh
```

构建镜像。

```
docker build -t swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag> .
```

参数说明：

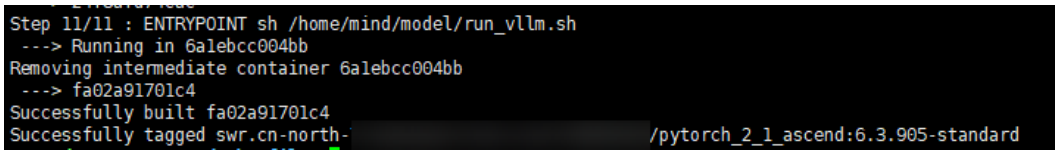
- <组织名称>：Step3中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

示例：

docker build -t swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示构建镜像成功。

图 4-888 成功构建镜像



Step7 上传镜像

在ECS服务器中输入Step4登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：Step3中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示上传镜像成功。

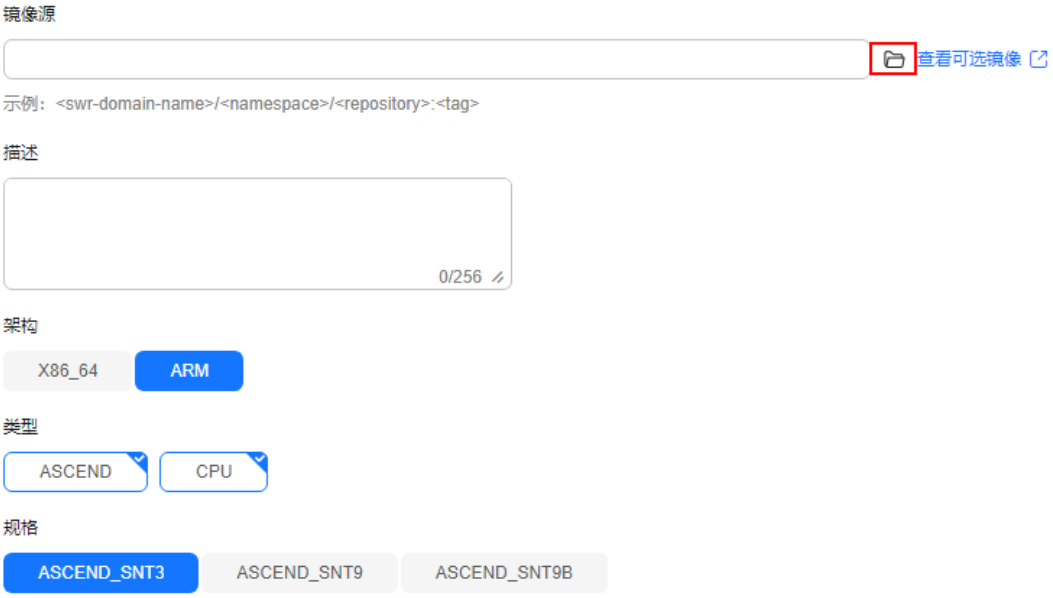
图 4-889 成功上传镜像

Step8 注册镜像

镜像上传至SWR成功后，在ModelArts控制台注册镜像。

1. 登录ModelArts管理控制台，在左侧导航栏选择“资产管理 > 镜像管理”，然后在“镜像管理”页面右上角单击“注册镜像”。
2. 在“注册镜像”页面，“镜像源”选择上一步上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，“架构”选择“ARM”，“类型”选中“ASCEDN”和“CPU”，按需选择规格，单击“立即注册”。

图 4-890 选择已上传的镜像源



Step9 构建推理代码

提前在ECS中构建推理代码，用于后续在推理生产环境中部署推理服务。

执行GIT安装命令。

```
sudo yum update
sudo yum install git
```

解压AscendCloud-3rdLLM-6.3.905-xxx.zip代码包。

```
unzip AscendCloud-3rdLLM-6.3.905-*.zip
```

运行推理构建脚本build.sh文件，自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。

```
cd llm_inference
bash build.sh
```

运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

将生成的ascend_vllm文件夹从ECS中取出并上传至OBS中。

Step10 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

4.51.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，单击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 4-891 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Ascend snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，**在Notebook调试环境中部署推理服务**。

4.51.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

Step2 准备模型代码包和权重文件

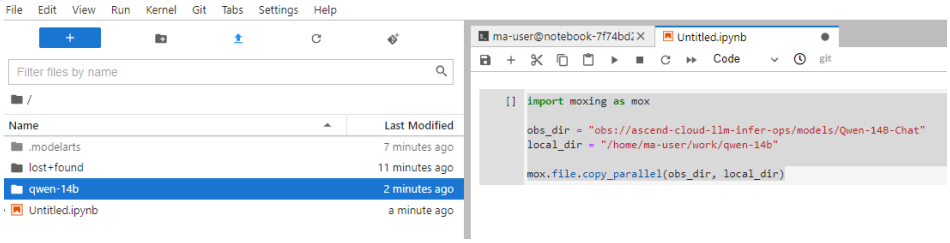
1. 将OBS中的模型权重和**表4-471**获取的AscendCloud-3rdLLM-6.3.905-xxx.zip代码包上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"
```

```
mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 4-892 上传 OBS 文件到 Notebook 的代码示例



2. 构建推理代码。

   解压AscendCloud-3rdLLM-6.3.905-xxx.zip代码包。

   ```
   unzip AscendCloud-3rdLLM-6.3.905-*.zip
   ```

   运行推理构建脚本build.sh文件，自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。

   ```
   cd llm_inference
   bash build.sh
   ```

   运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

Step3 配置 NPU 环境

在Notebook的terminal中执行如下命令进行环境配置。

配置需要的NPU卡。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3
```

0,1,2,3修改为需要使用的卡，如需使用全部8张卡，修改为0,1,2,3,4,5,6,7。

配置PYTHONPATH。

```
export PYTHONPATH=$PYTHONPATH:${vllm_path}
```

${vllm_path}：指定到ascend_vllm文件夹的绝对路径。

进入工作目录。

```
cd ascend_vllm
```

Step4 部署并启动推理服务

在Step3中的terminal部署并启动推理服务。有2种方式，使用vllm-api启动推理服务，或者使用openai-api启动推理服务。参考命令如下：

```
# 使用vllm-api
python vllm/entrypoints/api_server.py \
--model="${model_path}" \
--tensor-parallel-size 1 \
--gpu-memory-utilization 0.95 \
--max-model-len=4096 \
--trust-remote-code \
--dtype="float16" \
--host=0.0.0.0 \
--port=8080
```

```
# 使用openai-api
python vllm/entrypoints/openai/api_server.py \
--model="${model_path}" \
--tensor-parallel-size 1 \
--gpu-memory-utilization 0.95 \
--max-model-len=4096 \
--trust-remote-code \
--dtype="float16" \
--host=0.0.0.0 \
--port=8080
```

参数说明：

- --model：模型地址，模型格式是Huggingface的目录格式。
- --tensor-parallel-size：并行卡数。
- --gpu-memory-utilization：0~1之间的float，实际使用的显存是系统读取的最大显存*gpu-memory-utilization。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口。

服务启动后，会打印如下信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step5 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

  ```
  curl http://localhost:8080/generate  -d '{"prompt": "hello", "temperature":0, "max_tokens":20}'
  ```

  vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**

- 方式二：使用OpenAI接口请求服务，命令参考如下。

  ```
  curl http://localhost:8080/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
       "model": "/data/nfs/model/llama-2-7b",
       "temperature": 0,
       "max_tokens": 20,
       "messages": [
          {"role": "system", "content": "You are a helpful assistant."},
          {"role": "user", "content": "hello"}
       ]
    }'
  ```

表 4-473 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path} 参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

Step6 推理服务的高阶配置（可选）

如需开启以下高阶配置，请在**Step3 配置NPU环境**时增加需要开启的高阶配置参数。

- 词表切分

  在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量。

  ```
  export USE_VOCAB_PARALLEL=1
  ```

  关闭词表切分的命令：

  ```
  unset USE_VOCAB_PARALLEL
  ```

  配置后重启推理服务生效。

- Matmul_all_reduce融合算子

  使用Matmul_all_reduce融合算子能提升全量推理性能，该算子对驱动和固件版本要求较高，默认不开启。如需开启，配置以下环境变量。

  ```
  export USE_MM_ALL_REDUCE_OP=1
  ```

  关闭Matmul_all_reduce融合算子的命令：

```
unset  USE_MM_ALL_REDUCE_OP
```

配置后重启推理服务生效。

- 查看详细日志

  查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量。

  ```
  export DETAIL_TIME_LOG=1
  export RAY_DEDUP_LOGS=0
  ```

  关闭详细日志命令：

  ```
  unset  DETAIL_TIME_LOG
  ```

  配置后重启推理服务生效。

Step7 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.3.2）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9，qwen系列、qwen1.5系列、llama3系列模型还需打开词表切分配置export USE_VOCAB_PARALLEL=1。

序号	模型名称	4*64GB	8*32GB
1	qwen1.5-72b	24576	8192
2	qwen-72b	24576	8192
3	llama3-70b	32768	8192
4	llama2-70b	98304	32768
6	llama-65b	24576	8192

序号	模型名称	2*64GB	4*32GB
1	qwen1.5-32b	65536	24576

序号	模型名称	1*64GB	1*32GB
1	qwen1.5-7b	49152	16384
2	qwen-7b	49152	16384
3	llama3-8b	98304	32768
4	llama2-7b	126976	16384
5	chatglm3-6b	126976	65536

序号	模型名称	1*64GB	1*32GB
6	chatglm2-6b	126976	65536

序号	模型名称	1*64GB	2*32GB
1	qwen1.5-14b	24576	24576
2	qwen-14b	24576	24576
3	llama2-13b	24576	24576

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

4.51.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备ascend_vllm代码包、模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- ascend_vllm代码包在**Step9 构建推理代码**已生成。
- 模型权重文件获取地址请参见**表4-469**。
- 推理启动脚本run_vllm.sh制作请参见**•创建推理脚本文件run_vllm.sh**。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 4-893 准备模型文件和权重文件

- **创建推理脚本文件run_vllm.sh**

  run_vllm.sh脚本内容如下。

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
```

```
export PYTHONPATH=$PYTHONPATH:/home/mind/model/ascend_vllm

cd /home/mind/model/ascend_vllm/
python /home/mind/model/ascend_vllm/vllm/entrypoints/api_server.py --model="${model_path}" --ssl-keyfile="/home/mind/model/key.pem" --ssl-certfile="/home/mind/model/cert.pem" --tensor-parallel-size 1 --gpu-memory-utilization 0.95 --max-model-len=4096 --trust-remote-code --dtype="float16"  --host=0.0.0.0 --port=8080
```

参数说明：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：home/mind/model/chatglm3-6b。
- --tensor-parallel-size：并行卡数。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- 其他参数可以根据实际情况进行配置，也可使用openai接口启动服务。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“资产管理 > AI应用 > 创建”，开始创建AI应用。
2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 4-894 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

说明

若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台，单击“模型部署 > 在线服务 > 部署”，开始部署在线服务。
2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 4-895 部署在线服务

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 4-896 服务部署完成

Step4 调用在线服务

进入在线服务详情页面，选择“预测”，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”即可看到预测结果。在线服务的更多内容介绍请参见文档**查看服务详情**。

图 4-897 预测

Step5 推理服务高阶配置（可选）

如需开启以下高阶配置，请在**•创建推理脚本文件run_vllm.sh**章节创建的推理脚本run_vllm.sh中增加需要开启的高阶配置。

- 词表切分

  在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量。

  ```
  export USE_VOCAB_PARALLEL=1
  ```

  关闭词表切分的命令：

```
unset USE_VOCAB_PARALLEL
```

配置后重启推理服务生效。

- Matmul_all_reduce融合算子

  使用Matmul_all_reduce融合算子能提升全量推理性能，该算子对驱动和固件版本要求较高，默认不开启。如需开启，配置以下环境变量。

  ```
  export USE_MM_ALL_REDUCE_OP=1
  ```

  关闭Matmul_all_reduce融合算子的命令：

  ```
  unset  USE_MM_ALL_REDUCE_OP
  ```

  配置后重启推理服务生效。

- 查看详细日志

  查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量。

  ```
  export DETAIL_TIME_LOG=1
  export RAY_DEDUP_LOGS=0
  ```

  关闭详细日志命令：

  ```
  unset  DETAIL_TIME_LOG
  ```

  配置后重启推理服务生效。

Step6 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

4.51.5 推理精度测试

本章节介绍如何进行推理精度测试，建议在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。若需要在生产环境中进行推理精度测试，请通过调用接口的方式进行测试。

Step1 执行精度测试

精度测试需要数据集进行测试。推荐公共数据集mmlu和ceval。AscendCloud-3rdLLM-6.3.905-xxx.zip代码包已包含数据集。

精度测试使用的是openai接口，部署服务的时候请使用openai-api启动，暂不支持vllm-api接口。

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-3rdLLM的/llm_evaluation目录中，代码目录结构如下：

   ```
   benchmark_eval
   ├── config
   │   ├── config.json  # 服务的配置模板，已配置了ma-standard，tgi示例
   │   ├── mmlu_subject_mapping.json  # mmlu数据集学科信息
   │   ├── ceval_subject_mapping.json  # ceval数据集学科信息
   ├── evaluators
   │   ├── evaluator.py # 数据集数据预处理方法集
   │   ├── chatglm.py  # 处理请求相应模块, 一般和chatglm的官方评测数据集ceval搭配
   │   ├── llama.py  # 处理请求相应模块, 一般和llama的评测数据集mmlu搭配
   ├── mmlu-exam, mmlu数据集
   ├── ceval-exam, ceval数据集
   ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
   ├── service_predict.py  # 发送请求的服务
   ```

2. 执行精度测试启动脚本eval_test.py，具体操作命令如下，可以根据参数说明修改参数。

   ```
   python eval_test.py \
    --max_workers=1 \
   ```

```
--service_name=qwen-14b-test \
--eval_dataset=ceval \
--service_url=${API接口公网地址}/v1/completions \
--few_shot=3 \
--is_devserver=False \
--vllm_model=${model_path} \
--deploy_method=vllm
```

参数说明:

- max_workers：请求的最大线程数，默认为1。
- service_name：服务名称，保存评测结果时创建目录，示例为：qwen-14b-test。
- eval_dataset：评测使用的评测集（枚举值），目前仅支持mmlu、ceval。
- service_url：服务接口地址，若服务部署在notebook中，该地址为"http://127.0.0.1:${port}/v1/completions"；若服务部署在生产环境中，该地址由API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

  图 4-898 API 接口公网地址

- few_shot：开启少量样本测试后添加示例样本的个数。默认为3，取值范围为0~5整数。
- is_devserver： 是否DevServer部署方式，True表示DevServer模式。False表示ModelArts Standard模式。
- vllm_model：对应**Step4 部署并启动推理服务**中的模型地址参数model，模型格式是Huggingface的目录格式。
- deploy_method：部署方法，不同的部署方式api参数输入、输出解析方式不同，目前支持tgi、vllm等方式，本案例使用vllm部署方式。

说明

若要在生产环境中进行精度测试，还需修改benchmark_eval/config/config.json中app_code，app_code获取方式见**访问在线服务（APP认证）**。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{service_name}/{eval_dataset}-{timestamp} 的目录结果保存到对应的测试工程。执行多少次，则会在{service_name}下生成多少次结果。

单独的评测结果如下：

```
{eval_dataset}-{timestamp}  # 例如: mmlu-20240205093257
├── accuracy
│   ├── evaluation_accuracy.xlsx  # 测试的评分结果，包含各个学科数据集的评分和总和评分。
├── infer_info
│   ├── xxx1.csv # 单个数据集的评测结果
│   ├── ......
│   ├── xxxn.csv # 单个数据集的评测结果
├── summary_result
│   ├── answer_correct.xlsx  # 回答正确的结果
│   ├── answer_error.xlsx  # 保存回答了问题的选项，但是回答结果错误
│   ├── answer_result_unknow.xlsx  # 保存未推理出结果的问题，例如超时、系统错误
│   ├── system_error.xlsx  # 保存推理结果，但是可能答非所问，无法判断是否正确，需要人工判断进行纠偏。
```

4.51.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

benchmark 方法介绍

性能benchmark包括两部分。

- 静态性能测试：评估在固定输入、固定输出和固定并发下，模型的吞吐与首token延迟。该方式实现简单，能比较清楚的看出模型的性能和输入输出长度、以及并发的关系。
- 动态性能测试：评估在请求并发在一定范围内波动，且输入输出长度也在一定范围内变化时，模型的延迟和吞吐。该场景能模拟实际业务下动态的发送不同长度请求，能评估推理框架在实际业务中能支持的并发数。

性能benchmark验证使用到的脚本存放在代码包AscendCloud-3rdLLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本
```

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

notebook中进行测试：

```
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

图 4-899 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在notebook中，该参数为notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中**${model_path}**。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 4-900 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57729576
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

   动态benchmark需要使用数据集进行测试，可以使用公开数据集，例如Alpaca、ShareGPT。也可以根据业务实际情况，使用generate_datasets.py脚本生成和业务数据分布接近的数据集。

   公开数据集下载地址：

   - ShareGPT: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**
   - Alpaca: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

   使用generate_datasets.py脚本生成数据集方法：

generate_datasets.py脚本通过指定输入输出长度的均值和标准差，生成一定数量的正态分布的数据。具体操作命令如下，可以根据参数说明修改参数。

```
cd benchmark_tools
python generate_datasets.py --datasets custom_datasets.json --tokenizer /path/to/tokenizer \
--min-input 100 --max-input 3600 --avg-input 1800 --std-input 500 \
--min-output 40 --max-output 256 --avg-output 160 --std-output 30 --num-requests 1000
```

generate_datasets.py脚本执行参数说明如下：

- --datasets：数据集保存路径，如custom_datasets.json。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

2. 执行脚本benchmark_serving.py测试动态benchmark。具体操作命令如下，可以根据参数说明修改参数。

   notebook中进行测试：

   ```
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset
   custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8
   10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens
   3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json
   --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts
   10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv
   benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

     图 4-901 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --datasets：数据集路径。
   - --datasets-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。

- --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在notebook中，该参数为notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中**${model_path}**。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。

脚本运行完后，测试结果保存在benchmark_serving.csv中，示例如下图所示。

图 4-902 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.066428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

5 MLLM 多模态模型训练推理

5.1 Qwen-VL 基于 Standard+OBS+SFS 适配 PyTorch NPU 训练指导（6.3.912）

5.1.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，以基于DeepSpeed的Qwen-VL模型为例，为用户提供了多模态理解模型在ModelArts Standard上的全量微调和LoRA微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

- 6.3.912版本是第一次发布

支持的模型列表

本方案支持以下模型的训练，如表5-1所示。

表 5-1 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址	框架
1	Qwen-VL	7b	https://huggingface.co/Qwen/Qwen-VL-Chat	DeepSpeed

操作流程

图 5-1 操作流程图

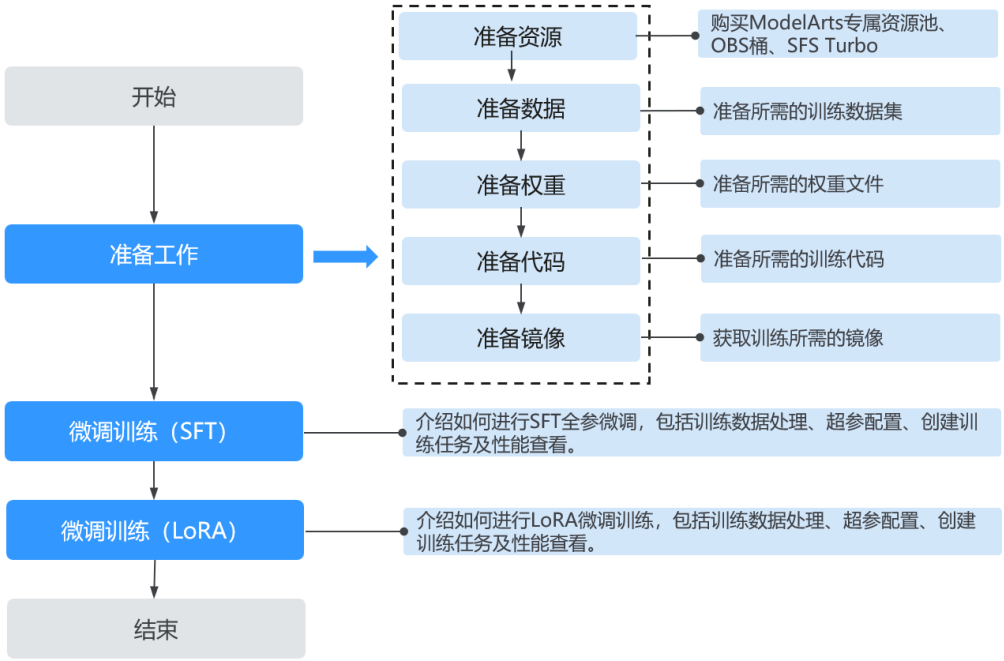


表 5-2 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。

阶段	任务	说明
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

5.1.2 准备工作

5.1.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：不同模型训练推荐的NPU卡数请参见**不同模型推荐的参数与NPU卡数设置**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名： standard-qwenvl-7b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与

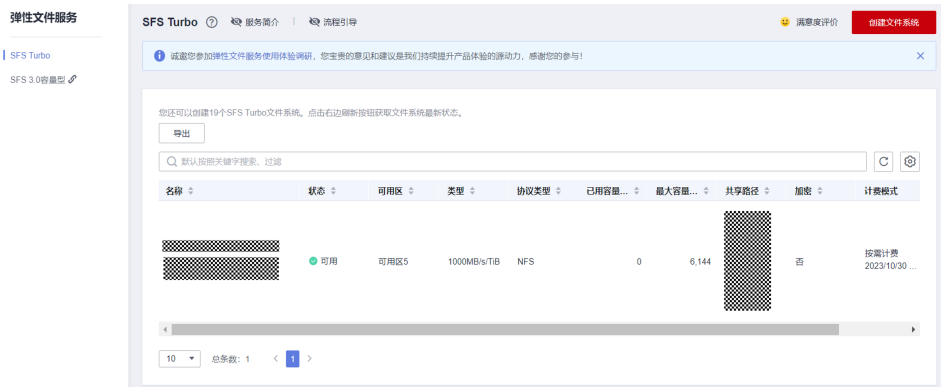
SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 5-2 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

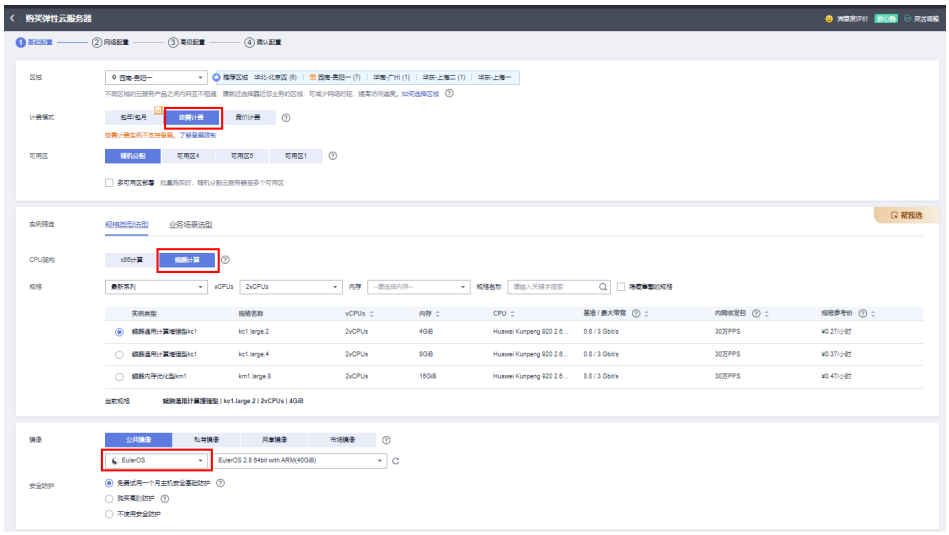
图 5-3 SFS 类型和容量选择

创建 ECS 服务器

弹性云服务器（Elastic Cloud Server，ECS）是由CPU、内存、操作系统、云硬盘组成的一种可随时获取、弹性可扩展的云服务器。具体过程请参考**ECS文档**购买一个Linux弹性云服务器。创建完成后，单击“远程登录”，可直接访问ECS服务器。

注意：CPU架构必须选择鲲鹏计算；镜像推荐选择EulerOS；ECS服务器确保可以访问公网，用于获取镜像和构建镜像。

图 5-4 购买 ECS

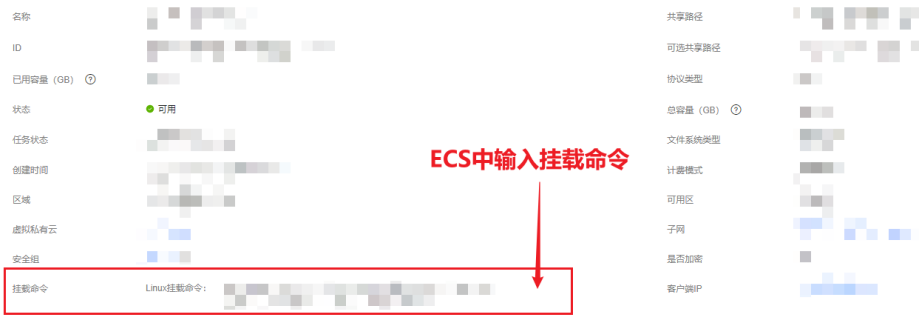


ECS 服务器挂载 SFS Turbo

ECS服务器中手动挂载SFS Turbo步骤如下：

1. 用户可通过CloudShell或SSH等方式登录并访问ECS服务器，进入ECS终端界面。创建/mnt/sfs_turbo目录作为挂载目录 ，命令为：mkdir /mnt/sfs_turbo。
2. 单击用户创建的SFS Turbo，查看SFS Turbo基本信息，找到并复制挂载命令。
3. 在ECS的终端中粘贴SFS Turbo挂载命令，完成挂载。
4. 挂载完成后，可通过后续的步骤获取到代码和数据，并上传至/mnt/sfs_turbo路径下。

图 5-5 SFS Turbo 基本信息



ModelArts 网络关联 SFS Turbo

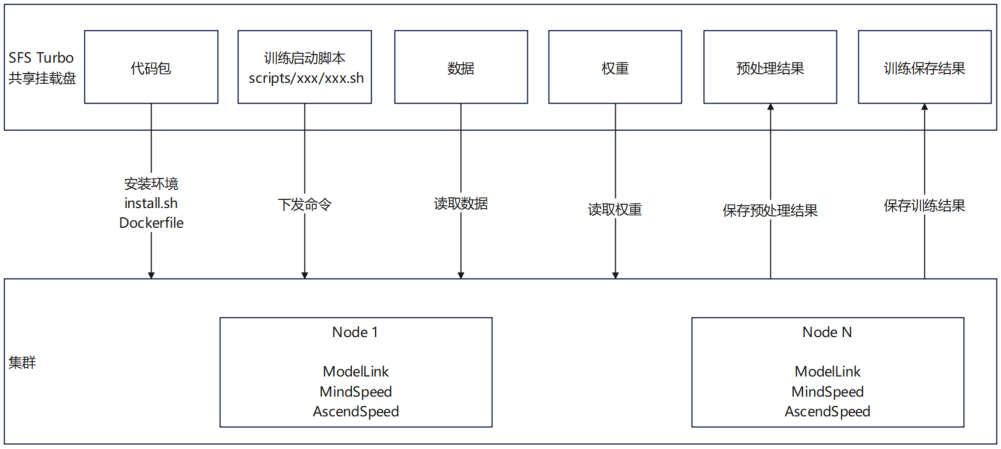
OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

如果ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 5-6 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程

图 5-7 SFS Turbo 和集群交互示意图



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的原始数据集、原始Hugging Face权重文件以及训练代码都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过CloudShell远程登录ECS并将代码包上传至SFS Turbo中。
2. 在**表5-4**获取基础镜像，随后通过**准备镜像**中的步骤执行代码包中AscendCloud-AIGC-6.3.912-xxx/multimodal_algorithm/QwenVL/train/<commit_id>/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 在构建镜像的过程中会下载完整的模型代码、执行环境，然后自动进行NPU适配，并将以上源码和环境打包至镜像中。
4. 在ModelArts中创建训练作业如：**SFT全参微调训练**，执行代码包中例如：finetune/finetune_ds.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**查看日志和性能**、**训练脚本说明** 了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

5.1.2.2 准备数据

本教程使用自定义数据集，数据集的介绍及下载链接参考**自定义数据**。

自定义数据

Qwen-VL指令微调数据：Qwen-VL-Chat微调的数据需要用户自行制作，需要准备一个JSON文件存放训练样本，每个样本需包含id和对话内容。对话内容按user和

assistant轮流发言记录。具体的格式需要参考Qwen-VL**官方指导资料**，示例如下所示：

```
[
 {
  "id": "identity_0",
  "conversations": [
   {
    "from": "user",
    "value": "你好"
   },
   {
    "from": "assistant",
    "value": "我是Qwen-VL,一个支持视觉输入的大模型。"
   }
  ]
 },
 {
  "id": "identity_1",
  "conversations": [
   {
    "from": "user",
    "value": "Picture 1: <img>qwenvl_dataset/new_single_bar/demo.jpeg</img>\n图中的狗是什么品种？"
   },
   {
    "from": "assistant",
    "value": "图中是一只拉布拉多犬。"
   },
   {
    "from": "user",
    "value": "框出图中的格子衬衫"
   },
   {
    "from": "assistant",
    "value": "<ref>格子衬衫</ref><box>(588,499),(725,789)</box>"
   }
  ]
 },
 {
  "id": "identity_2",
  "conversations": [
   {
    "from": "user",
    "value": "Picture 1: <img>qwenvl_dataset/new_single_bar/Chongqing.jpeg</img>\nPicture 2: <img>qwenvl_dataset/new_single_bar/Beijing.jpeg</img>\n图中都是哪"
   },
   {
    "from": "assistant",
    "value": "第一张图片是重庆的城市天际线，第二张图片是北京的天际线。"
   }
  ]
 }
]
```

为针对多样的VL任务，特殊tokens如下： <img> </img> <ref> </ref> <box> </box>.

- 对于带图像输入的内容可表示为 Picture id: <img>img_path</img>\n{your prompt}，其中id表示对话中的第几张图片。"img_path"可以是本地的图片或网络地址。
- 对话中的检测框可以表示为<box>(x1,y1),(x2,y2)</box>，其中 (x1, y1) 和(x2, y2)分别对应左上角和右下角的坐标，并且被归一化到[0, 1000)的范围内. 检测框对应的文本描述也可以通过<ref>text_caption</ref>表示。
- json文件中的图片路径为数据集相对路径，例如qwenvl_dataset/new_single_bar/Chongqing.jpeg。

上传数据集至 OBS

1. 准备数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-qwenvl-7b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://standard-qwenvl-7b
└── training_data
        └── qwenvl_dataset                  #数据集目录
            ├── chart_qa_train_ocr.json      # json文件
            └── new_single_bar              # 图片目录
                └── single_bar_1_1000.jpg
                └── single_bar_1_1001.jpg
                └── single_bar_1_1002.jpg
                └── single_bar_1_1003.jpg
                    ...
```

5.1.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表5-1**。

   权重文件下载有如下几种方式，但不仅限于以下方式：

   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Qwen/Qwen-VL-Chat为例：

     ```
     huggingface-cli download --resume-download Qwen/Qwen-VL-Chat --local-dir <模型下载路径>
     ```

     如果要下载指定版本的模型文件，则命令如下：

     ```
     huggingface-cli download --resume-download Qwen/Qwen-VL-Chat --revision <模型版本>  --local-dir <模型下载路径>
     ```

   - **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
   - **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-qwenvl-7b中创建文件夹models。
3. 参考文档利用**OBS-Browser-Plus**工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以Qwen/Qwen-VL-Chat为例：

```
obs://<bucket_name>/models/Qwen-VL-Chat/
├── config.json
├── configuration_qwen.py
├── generation_config.json
├── modeling_qwen.py   # 需要修改代码
├── pytorch_model-00001-of-00010.bin
├── pytorch_model-00002-of-00010.bin
├── pytorch_model-00003-of-00010.bin
├── pytorch_model-00004-of-00010.bin
├── pytorch_model-00005-of-00010.bin
├── pytorch_model-00006-of-00010.bin
├── pytorch_model-00007-of-00010.bin
├── pytorch_model-00008-of-00010.bin
├── pytorch_model-00009-of-00010.bin
├── pytorch_model-00010-of-00010.bin
├── pytorch_model.bin.index.json
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
```

```
├── tokenization_qwen.py  # 需要修改代码
├── SimSun.ttf            # 需要手动下载
├── tokenizer_config.json
└── visual.py
```

4. 对于Qwen-VL模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

5.1.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表5-3**所示。

表 5-3 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.912代码包中AscendCloud-AIGC代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
AscendCloud-AIGC-6.3.912-xxx
├─aigc_inference
├─aigc_train
├─AscendCloud-Pytorch-Plugin
└─multimodal_algorithm
  ├─ascendcloud_multimodal_plugin
  ...
  ├─QwenVL
  ├── train/<commit_id>     # 本教程中需要的模型代码
            └── Dockerfile                  # 构建镜像
            └── qwen-vl_install.sh          # 安装模型运行环境
            └── qwen-vl.patch               # 使用git apply修改模型相关代码
```

代码上传至 OBS

代码包解压后，在OBS中创建mllm_train目录，并将train/<commit_id>上传至该目录中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
obs://standard-qwenvl-7b
├── training_data
│   └── qwenvl_dataset                  #数据集目录
│       ├── chart_qa_train_ocr.json      # json文件
│       └── new_single_bar               # 图片目录
│           └── single_bar_1_1000.jpg
│           └── single_bar_1_1001.jpg
│           └── single_bar_1_1002.jpg
│           └── single_bar_1_1003.jpg
│               ...
├── mllm_train
│   └── train/<commit_id>                    # 本教程中需要的模型代码
│       └── Dockerfile                  # 构建镜像
│       └── qwen-vl_install.sh           # 安装模型运行环境
│       └── qwen-vl.patch               # 使用git apply修改模型相关代码
│
└── models
    └── Qwen-VL-Chat
```

5.1.2.5 将数据预热到 SFS Turbo

训练任务开始前可通过数据预热功能将文件元数据和数据内容全部从OBS导入到SFS Turbo高性能文件存储中，数据预热功能的具体操作请参考**创建SFS Turbo 和 OBS 之间的联动任务**。

在**ECS服务器挂载SFS Turbo**已经将SFS Turbo挂载到了/mnt/sfs_turbo目录，这里参考**obsutil文档**，直接使用obsutil命令将OBS桶中的所有数据拷贝到该目录即可。

```
obsutil cp obs://standard-qwenvl-7b  /mnt/sfs_turbo -r -f -flat

结束后可以看到如下目录结构：
/mnt/sfs_turbo
├── training_data
│   └── qwenvl_dataset                  #数据集目录
│       ├── chart_qa_train_ocr.json      # json文件
│       └── new_single_bar               # 图片目录
│           └── single_bar_1_1000.jpg
│           └── single_bar_1_1001.jpg
│           └── single_bar_1_1002.jpg
│           └── single_bar_1_1003.jpg
│               ...
├── mllm_train
│   └── train/<commit_id>                    # 本教程中需要的模型代码
│       └── Dockerfile                  # 构建镜像
│       └── qwen-vl_install.sh           # 安装模型运行环境
│       └── qwen-vl.patch               # 使用git apply修改模型相关代码
│
└── models
    └── Qwen-VL-Chat
```

5.1.2.6 准备镜像

5.1.2.6.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 5-4 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**ECS中构建新镜像**的方式来部署训练环境。可以在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。在构建镜像的过程中会下载完整的模型代码、执行环境，然后自动进行NPU适配，并将以上源码和环境打包至镜像中。

ECS需要连通公网，否则会导致安装下载源码、安装环境依赖、上传镜像到SWR等操作失败。ECS打通公网配置请参见**ECS绑定弹性公网IP**。

5.1.2.6.2 ECS 获取基础镜像

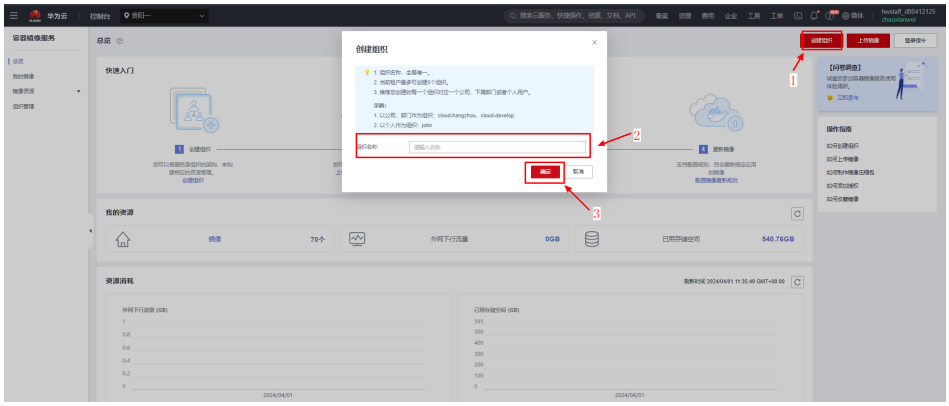
Step1 登录 ECS 服务器

根据**创建ECS服务器**创建完成ECS服务器后，单击“远程登录”，可使用华为CloudShell远程登录。后续安装Docker、获取镜像、构建镜像等操作均在该ECS上进行。

Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 5-8 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

```
docker -v  #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker
```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step4 获取训练镜像

请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表5-4**。

```
docker pull {image_url}
```

5.1.2.6.3 ECS 中构建新镜像

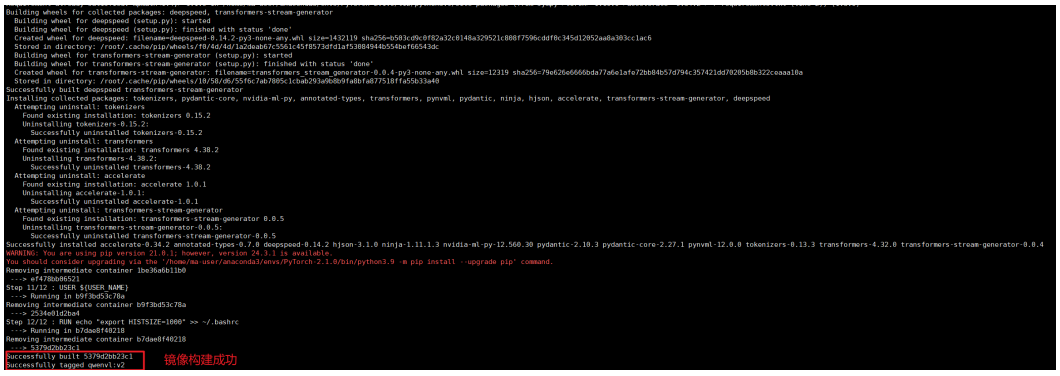
通过**ECS获取基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表5-3**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-AIGC-6.3.912-xxx.zip，并直接进入到QwenVL/train/<commit_id>文件夹下面
```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-AIGC-*.zip -d ./AscendCloud/AscendCloud-AIGC
cd ./AscendCloud/AscendCloud-AIGC/multimodal_algorithm/QwenVL/train/<commit_id>
```
2. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
```
docker build -t <镜像名称>:<版本名称>  .
```

图 5-9 docker 镜像构建过程



如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

```
docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
```

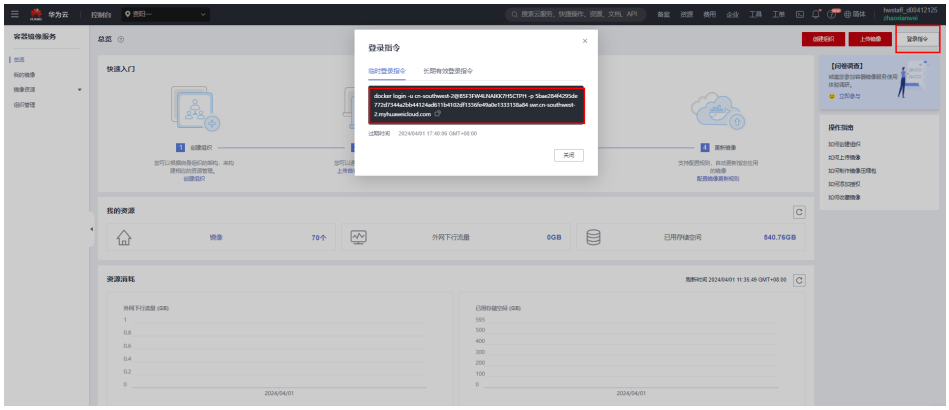
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
- 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

5.1.2.6.4 ECS 中上传新镜像

Step1 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 5-10 复制登录指令



Step2 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

   ```
   docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```

   参数说明：

   - ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
   - <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
   - <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

   ```
   docker tag  {image_url}  swr.cn-southwest-2.myhuaweicloud.com/ma-group/
   pytorch_2_1_ascend:20240606
   ```

2. 上传镜像至镜像仓库。

   ```
   docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```

   示例：

   ```
   docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606
   ```

5.1.3 SFT 全参微调训练

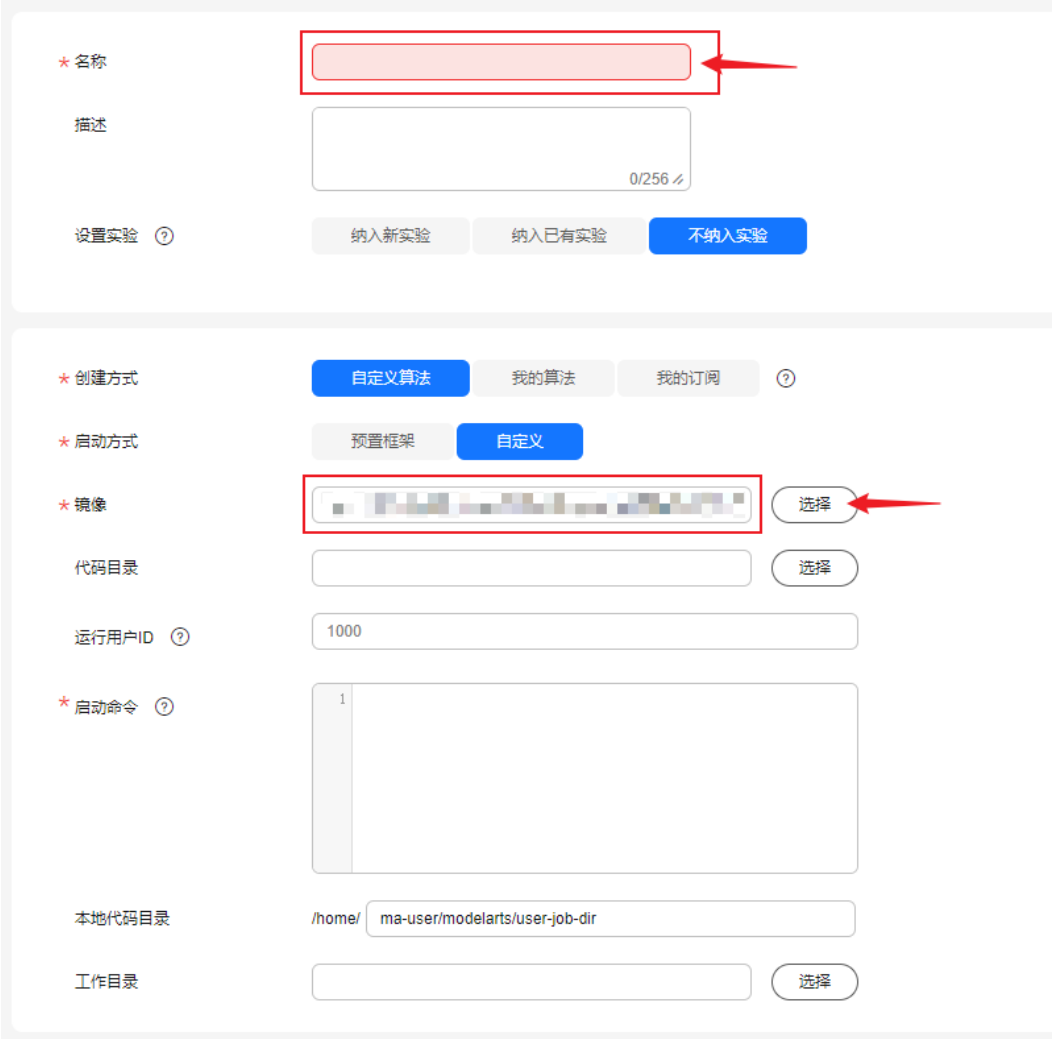
前提条件

已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

图 5-11 创建训练作业



训练作业启动命令中输入：

```
cd /home/ma-user/work/Qwen-VL;
ln -s ${DATA}/ qwenvl_dataset;
bash finetune/finetune_ds.sh;
```

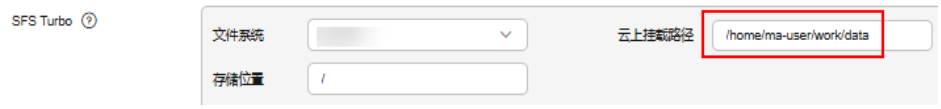
选择用户自己的专属资源池，以及规格与节点数。

图 5-12 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/data
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 5-13 选择 SFS Turbo



说明

SFS Turbo不能直接挂载到容器的工作路径 /home/ma-user/work/下，会覆盖镜像中的代码目录，导致训练失败。

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

Step2 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表5-5**表格中的配置进行填写。

表 5-5 需要填写的环境变量

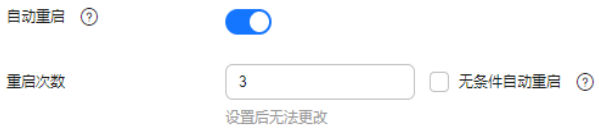
环境变量	示例值	参数说明
GPUS_PER_NODE	8	**必须填写。**根据资源规格每个节点上NPU的数量填写。
DATA	/home/ma-user/work/data/training_data/qwenvl_dataset	**必须修改**。训练时指定的输入数据路径。
MODEL	/home/ma-user/work/data/models/Qwen-VL-Chat	**必须修改**。训练时指定的模型权重路径。
OUTPUT	/home/ma-user/work/data/output	**必须修改**。训练完成后指定的输出模型的路径。

环境变量	示例值	参数说明
LOG	/home/ma-user/work/data/log	保存训练过程记录的日志LOG文件。

Step3 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。

图 5-14 开启故障重启



Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。本次qwenvl模型选用 8* ascend-snt9b。

图 5-15 选择资源池规格

在OBS中新建一个log目录，作业日志选择OBS中的该路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

5.1.4 LoRA 微调训练

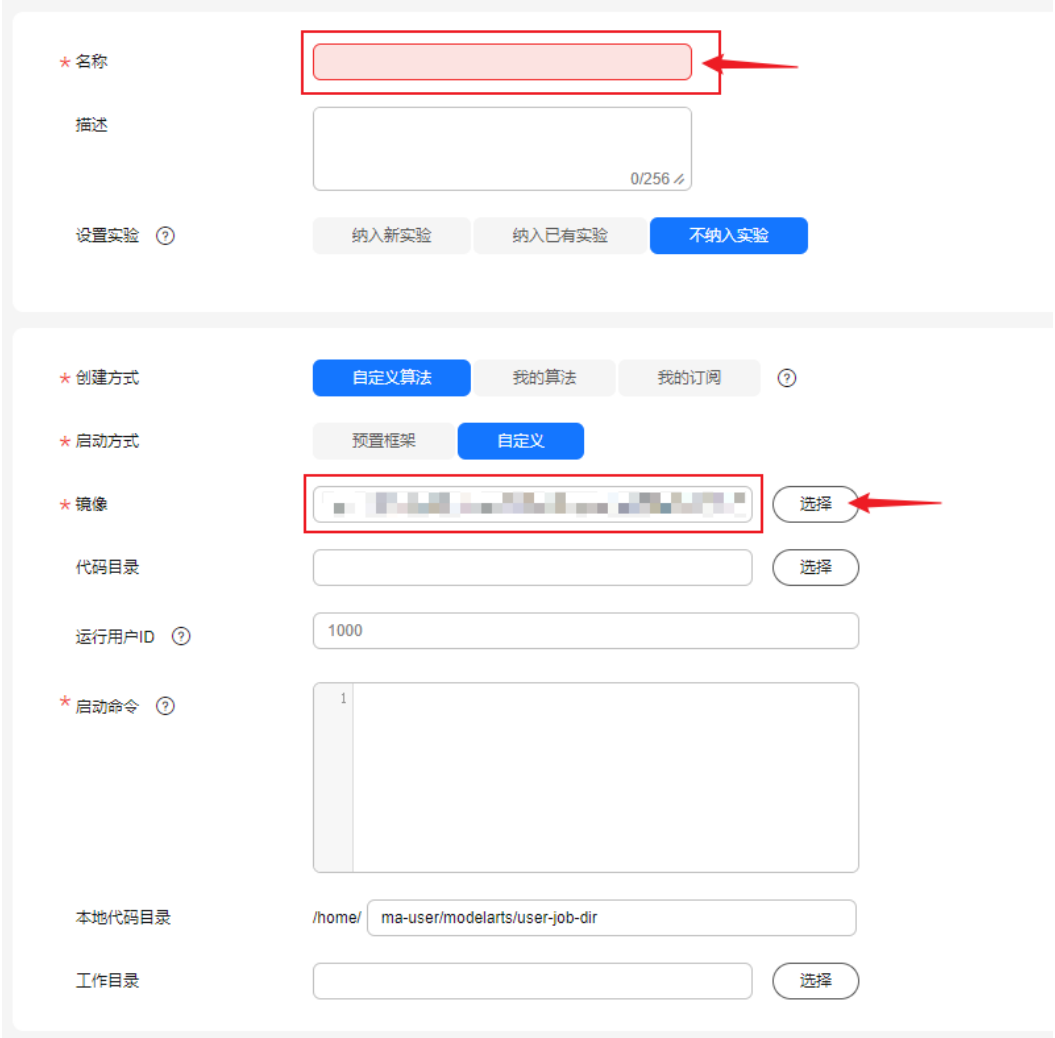
前提条件

已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

图 5-16 创建训练作业



训练作业启动命令中输入：

```
cd /home/ma-user/work/Qwen-VL;
ln -s ${DATA}/ qwenvl_dataset;
sh finetune/finetune_lora_ds.sh
```

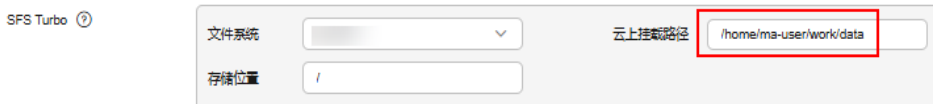
选择用户自己的专属资源池，以及规格与节点数。

图 5-17 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/data
- 存储位置：输入用户的“子目录挂载”路径。如果默认没有填写，则忽略。

图 5-18 选择 SFS Turbo



说明

SFS Turbo不能直接挂载到容器的工作路径 /home/ma-user/work/下，会覆盖镜像中的代码目录，导致训练失败。

作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

Step2 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照**表5-6**表格中的配置进行填写。

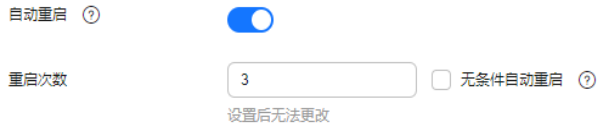
表 5-6 需要填写的环境变量

环境变量	示例值	参数说明
GPUS_PER_NODE	8	**必须填写。**根据资源规格每个节点上NPU的数量填写。
DATA	/home/ma-user/work/data/training_data/qwenvl_dataset	**必须修改**。训练时指定的输入数据路径。
MODEL	/home/ma-user/work/data/models/Qwen-VL-Chat	**必须修改**。训练时指定的模型权重路径。
OUTPUT	/home/ma-user/work/data/output	**必须修改**。训练完成后指定的输出模型的路径。
LOG	/home/ma-user/work/data/log	保存训练过程记录的日志 LOG 文件。

Step3 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。

图 5-19 开启故障重启



Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。本次qwenvl模型选用 8* ascend-snt9b。

图 5-20 选择资源池规格

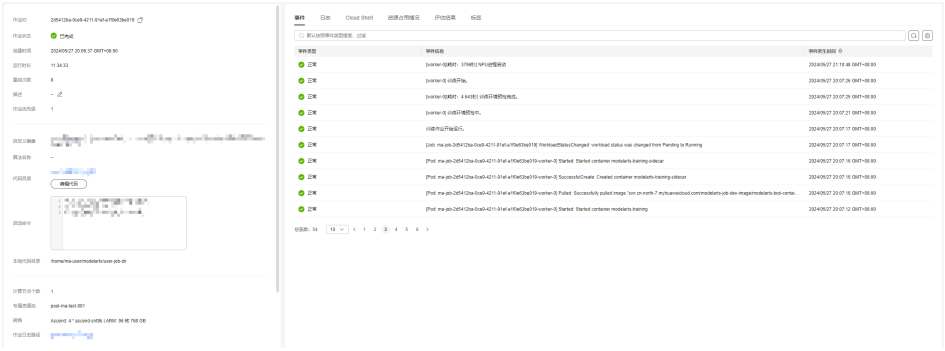
在OBS中新建一个log目录，作业日志选择OBS中的该路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

5.1.5 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

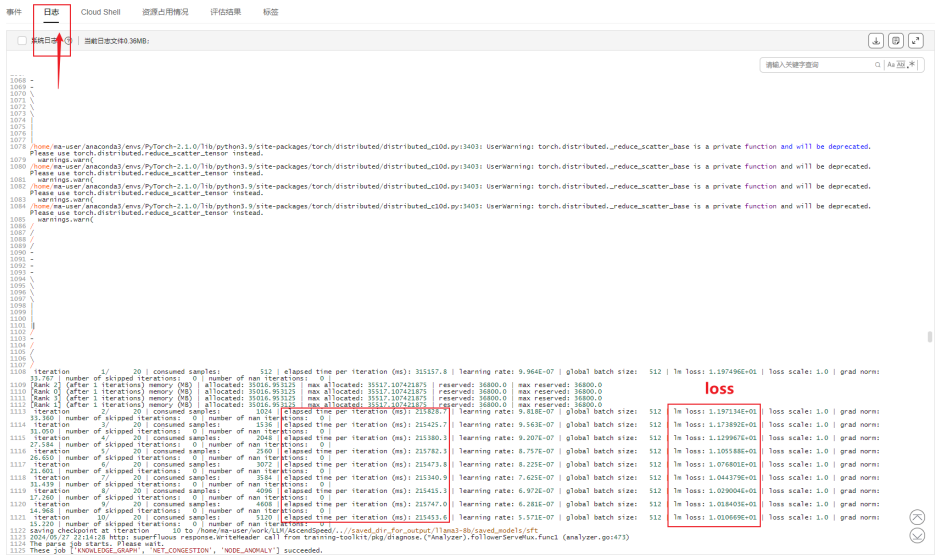
图 5-21 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 5-22 查看日志和性能



5.1.6 训练脚本说明

5.1.6.1 训练脚本参数说明

在AscendCloud-AIGC代码包的multimodal_algorithm目录下集成了多个多模态模型的适配脚本，用户可通过不同模型中的xxx_install.sh脚本一键适配。在用户通过Dockerfile构建模型的环境镜像时会执行该脚本，这会从github上拉取模型的官方源码，并通过git apply qwen-vl.patch的方式进行NPU适配，最后将以上源码和环境打包至镜像中。

```
AscendCloud-AIGC-6.3.912-xxx
├─aigc_inference
├─aigc_train
├─AscendCloud-Pytorch-Plugin
└─multimodal_algorithm
   ├─ascendcloud_multimodal_plugin
   │  ...
   └─QwenVL
   ├── train/<commit_id>     # 本教程中需要的模型代码
                └── Dockerfile                  # 构建镜像
                └── qwen-vl_install.sh          # 安装模型运行环境
                └── qwen-vl.patch               # 使用git apply修改模型相关代码
```

5.1.6.2 不同模型推荐的参数与 NPU 卡数设置

表 5-7 不同模型推荐的参数与 NPU 卡数设置

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
Qwen-VL	Qwen-VL	7B	full	2048	gradient_accumulation_steps: 16	ZeRO-3	1*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

5.1.6.3 训练 tokenizer 文件说明

在训练开始前，有些模型需要对模型的tokenizer文件，或者模型配置文件进行修改，具体的修改如下：

Qwen-VL

1. 修改文件modeling_qwen.py：

```
# 将36 37 两行注释部分
36 SUPPORT_BF16 = SUPPORT_CUDA #and torch.cuda.is_bf16_supported()
37 SUPPORT_FP16 = SUPPORT_CUDA #and torch.cuda.get_device_capability(0)[0] >= 7
```

2. 修改文件tokenization_qwen.py：

```
# tonkenization_qwen.py会在cache中读取SimSun.ttf 文件，如果没有，就会联网下载，ModelArts作业在执行过程中可能不能请求网络，会遇到报错。
# 直接手动下载 https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf ，放到模型权重目录Qwen-VL-Chat下。
# 然后将tokenization_qwen.py中30-35行注释，并重新定义变量FONT_PATH读取字体文件Simsun，修改如下：
30 # FONT_PATH = try_to_load_from_cache("Qwen/Qwen-VL-Chat", "SimSun.ttf")
31 # if FONT_PATH is None:
32 #     if not os.path.exists("SimSun.ttf"):
33 #         ttf = requests.get("https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf")
34 #         open("SimSun.ttf", "wb").write(ttf.content)
35 #     FONT_PATH = "SimSun.ttf"
FONT_PATH = os.path.join(os.getenv('DATA'), "SimSun.ttf"
```

5.1.7 常见错误原因和解决方法

5.1.7.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法：

通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。

5.1.7.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

5.1.7.3 联网下载 SimSun.ttf 时可能会遇到网络问题

联网下载SimSun.ttf时肯会遇到网络问题

tonkenization_qwen.py会在cache中读取SimSun.ttf 文件，如果没有，就会联网下载，可能会遇到：

1. SSL:CERTIFICATE_VERIFY_FAILED
2. ssl.1129错误
3. 407 Proxy Authentication Required

解决方案：

```
1. 直接手动下载 SimSun.ttf 传到 模型权重目录下 /home/ma-user/work/model-dir/Qwen-VL-Chat/
     https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf

2. 将 文件 /home/ma-user/work/model-dir/Qwen-VL-Chat/tokenization_qwen.py 中的 30-35 行注释
3. 然后增加一行直接读取本地的Simsun.ttf文件，写绝对路径
# FONT_PATH = try_to_load_from_cache("Qwen/Qwen-VL-Chat", "SimSun.ttf")# if FONT_PATH is None:
#     if not os.path.exists("SimSun.ttf"):
#         ttf = requests.get("https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf")
#         open("SimSun.ttf", "wb").write(ttf.content)
#     FONT_PATH = "SimSun.ttf"
FONT_PATH = "/home/ma-user/work/model-dir/Qwen-VL-Chat/SimSun.ttf"
```

5.1.7.4 在运行 finetune_ds.sh 时遇到报错

在运行finetune_ds.sh 时遇到报错

```
pydantic_core._pydantic_core.ValidationError: 1 validation error for DeepSpeedZeroConfig
stage3_prefetch_bucket_size
Input should be a valid integer, got a number with a fractional part [type=int_from_float,
input_value=15099494.4, input_type=float]
```

将deepspeed配置文件的 stage3_prefetch_bucket_size 参数值从 auto 改成 整数 15099494

5.2 Qwen-VL 模型基于 Standard+OBS 适配 PyTorch NPU 训练指导（6.3.912）

5.2.1 场景介绍

方案概览

本文档利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，以基于DeepSpeed的Qwen-VL模型为例，为用户提供了多模态理解模型在ModelArts Standard上的全量微调和LoRA微调方案。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 适配的CANN版本是cann_8.0.rc3，驱动版本是23.0.6。
- 本案例仅支持在专属资源池上运行，确保专属资源池可以访问公网。

文档更新内容

6.3.912版本是第一次发布。

支持的模型列表

本方案支持以下模型的训练，如表5-8所示。

表 5-8 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址	框架
1	Qwen-VL	7b	https://huggingface.co/Qwen/Qwen-VL-Chat	DeepSpeed

操作流程

图 5-23 操作流程图

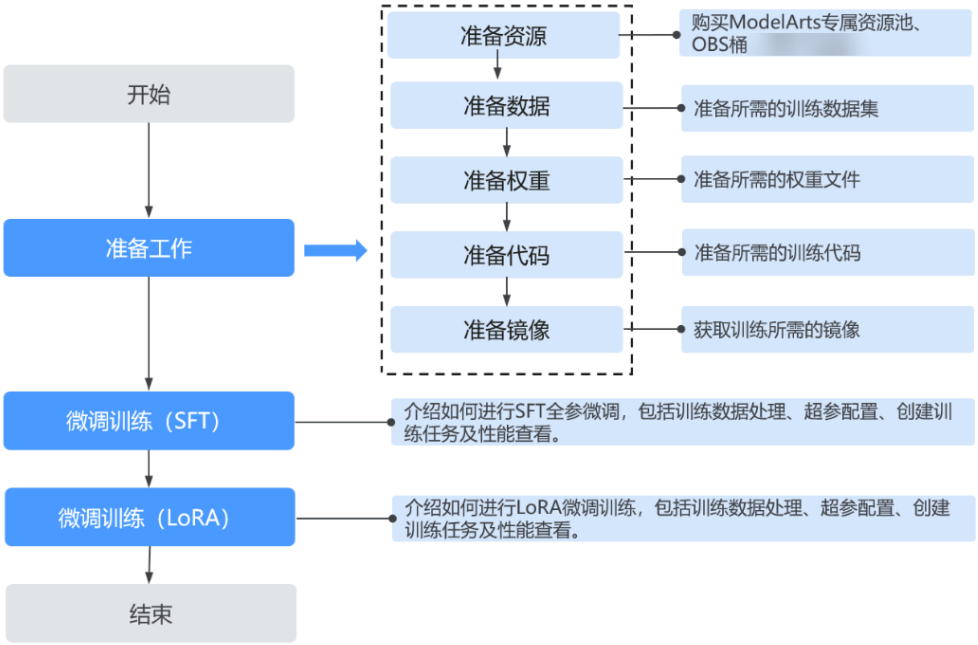


表 5-9 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

5.2.2 准备工作

5.2.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：不同模型训练推荐的NPU卡数请参见**不同模型推荐的参数与NPU卡数设置**。

硬盘空间：至少200GB。

昇腾资源规格：

- Ascend: 1*ascend-snt9b表示昇腾单卡。
- Ascend: 8*ascend-snt9b表示昇腾8卡。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名： standard-qwenvl-7b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

5.2.2.2 准备数据

本教程使用自定义数据集，数据集的介绍及下载链接参考**自定义数据**。

自定义数据

- **Qwen-VL指令微调数据**：Qwen-VL-Chat微调的数据需要用户自行制作，需要准备一个JSON文件存放训练样本，每个样本需包含id和对话内容。对话内容按user和assistant轮流发言记录。具体的格式需要参考Qwen-VL**官方指导资料**，示例如下所示：

```
[
 {
  "id": "identity_0",
  "conversations": [
   {
    "from": "user",
    "value": "你好"
   },
   {
    "from": "assistant",
    "value": "我是Qwen-VL,一个支持视觉输入的大模型。"
   }
  ]
 },
 {
  "id": "identity_1",
  "conversations": [
   {
```

```
      "from": "user",
      "value": "Picture 1: <img>qwenvl_dataset/new_single_bar/demo.jpeg</img>\n图中的狗是什么品种? "
    },
    {
      "from": "assistant",
      "value": "图中是一只拉布拉多犬。"
    },
    {
      "from": "user",
      "value": "框出图中的格子衬衫"
    },
    {
      "from": "assistant",
      "value": "<ref>格子衬衫</ref><box>(588,499),(725,789)</box>"
    }
  ]
 },
 {
  "id": "identity_2",
  "conversations": [
    {
      "from": "user",
      "value": "Picture 1: <img>qwenvl_dataset/new_single_bar/Chongqing.jpeg</img>\nPicture 2: <img>qwenvl_dataset/new_single_bar/Beijing.jpeg</img>\n图中都是哪"
    },
    {
      "from": "assistant",
      "value": "第一张图片是重庆的城市天际线，第二张图片是北京的天际线。"
    }
  ]
 }
]
```

为针对多样的VL任务，特殊tokens如下： <img> </img> <ref> </ref> <box> </box>.

- 对于带图像输入的内容可表示为 Picture id: <img>img_path</img>\n{your prompt}，其中id表示对话中的第几张图片。"img_path"可以是本地的图片或网络地址。
- 对话中的检测框可以表示为<box>(x1,y1),(x2,y2)</box>，其中 (x1, y1) 和 (x2, y2)分别对应左上角和右下角的坐标，并且被归一化到[0, 1000)的范围内. 检测框对应的文本描述也可以通过<ref>text_caption</ref>表示。
- json文件中的图片路径为数据集相对路径，例如qwenvl_dataset/new_single_bar/Chongqing.jpeg。

上传数据集至 OBS

1. 准备数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-qwenvl-7b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://standard-qwenvl-7b
└── training_data
        └── qwenvl_dataset                  #数据集目录
            ├── chart_qa_train_ocr.json      # json文件
            └── new_single_bar              # 图片目录
                └── single_bar_1_1000.jpg
                └── single_bar_1_1001.jpg
                └── single_bar_1_1002.jpg
                └── single_bar_1_1003.jpg
                    ...
```

5.2.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表5-8**。

   权重文件下载有如下几种方式，但不仅限于以下方式：

   - **方法一：网页下载**：通过单击表格中权重文件获取地址的访问链接，即可在模型主页的Files and Version中下载文件。
   - **方法二：huggingface-cli**：**huggingface-cli**是 Hugging Face 官方提供的命令行工具，自带完善的下载功能。具体步骤可参考：**HF-Mirror**中的使用教程。完成依赖安装和环境变量配置后，以Qwen/Qwen-VL-Chat为例：
     ```
     huggingface-cli download --resume-download Qwen/Qwen-VL-Chat --local-dir <模型下载路径>
     ```
     如果要下载指定版本的模型文件，则命令如下：
     ```
     huggingface-cli download --resume-download Qwen/Qwen-VL-Chat --revision <模型版本>  --local-dir <模型下载路径>
     ```
   - **方法三**：使用专用多线程下载器 hfd：**hfd** 是本站开发的 huggingface 专用下载工具，基于成熟工具 git+aria2，可以做到稳定下载不断线。
   - **方法四**：使用**Git clone**，官方提供了 git clone repo_url 的方式下载，但是不支持断点续传，并且clone 会下载历史版本占用磁盘空间。

2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-qwenvl-7b中创建文件夹models。
3. 参考文档利用**OBS-Browser-Plus**工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以Qwen/Qwen-VL-Chat为例：
   ```
   obs://<bucket_name>/models/Qwen-VL-Chat/
   ├── config.json
   ├── configuration_qwen.py
   ├── generation_config.json
   ├── modeling_qwen.py   # 需要修改代码
   ├── pytorch_model-00001-of-00010.bin
   ├── pytorch_model-00002-of-00010.bin
   ├── pytorch_model-00003-of-00010.bin
   ├── pytorch_model-00004-of-00010.bin
   ├── pytorch_model-00005-of-00010.bin
   ├── pytorch_model-00006-of-00010.bin
   ├── pytorch_model-00007-of-00010.bin
   ├── pytorch_model-00008-of-00010.bin
   ├── pytorch_model-00009-of-00010.bin
   ├── pytorch_model-00010-of-00010.bin
   ├── pytorch_model.bin.index.json
   ├── qwen_generation_utils.py
   ├── qwen.tiktoken
   ├── README.md
   ├── tokenization_qwen.py  # 需要修改代码
   ├── SimSun.ttf            # 需要手动下载
   ├── tokenizer_config.json
   └── visual.py
   ```
4. 对于Qwen-VL模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

5.2.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表5-10**所示。

表 5-10 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.912-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.912代码包中AscendCloud-AIGC代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
AscendCloud-AIGC-6.3.912-xxx
├─aigc_inference
├─aigc_train
├─AscendCloud-Pytorch-Plugin
└─multimodal_algorithm
   ├─ascendcloud_multimodal_plugin
   ...
   ├─QwenVL
   ├── train/<commit_id>     # 本教程中需要的模型代码
                └── Dockerfile                  # 构建镜像
                └── qwen-vl_install.sh          # 安装模型运行环境
                └── qwen-vl.patch               # 使用git apply修改模型相关代码
```

代码上传至 OBS

代码包解压后，在OBS中创建mllm_train目录，并将train/<commit_id>上传至该目录中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
obs://standard-qwenvl-7b
├── training_data
│      └── qwenvl_dataset                  #数据集目录
│            ├── chart_qa_train_ocr.json       # json文件
│            └── new_single_bar               # 图片目录
│                  └── single_bar_1_1000.jpg
│                └── single_bar_1_1001.jpg
│                └── single_bar_1_1002.jpg
│                └── single_bar_1_1003.jpg
│                  ...
├── mllm_train
│      └── train/<commit_id>                    # 本教程中需要的模型代码
│            └── Dockerfile                  # 构建镜像
│              └── qwen-vl_install.sh          # 安装模型运行环境
│            └── qwen-vl.patch               # 使用git apply修改模型相关代码
│
└── models
       └── Qwen-VL-Chat
```

5.2.2.5 准备镜像

5.2.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 5-11 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	CANN：cann_8.0.rc3 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**ECS中构建新镜像**的方式来部署训练环境。可以在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。在构建镜像的过程中会下载完整的模型代码、执行环境，然后自动进行NPU适配，并将以上源码和环境打包至镜像中。

ECS需要连通公网，否则会导致安装下载源码、安装环境依赖、上传镜像到SWR等操作失败。ECS打通公网配置请参见**ECS绑定弹性公网IP**。

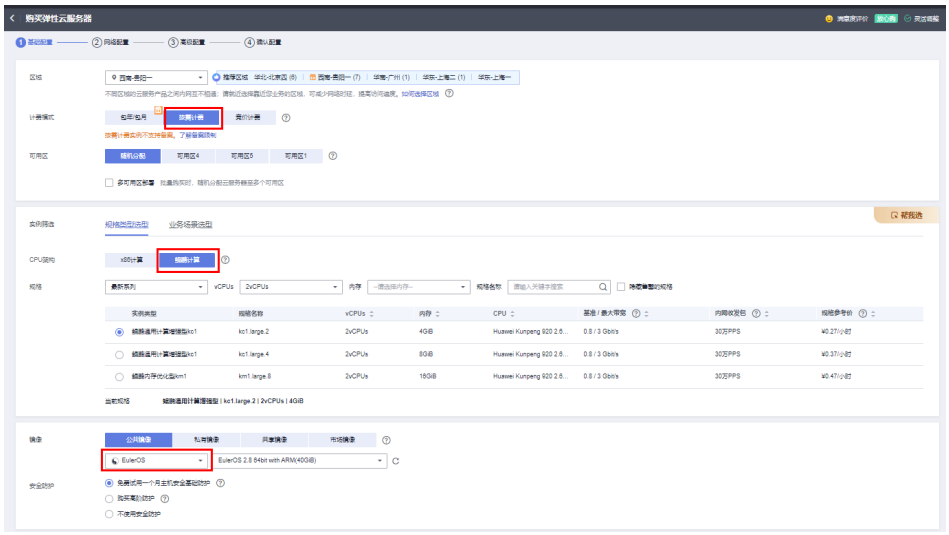
5.2.2.5.2 ECS 获取基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS，登录凭证使用密钥对。

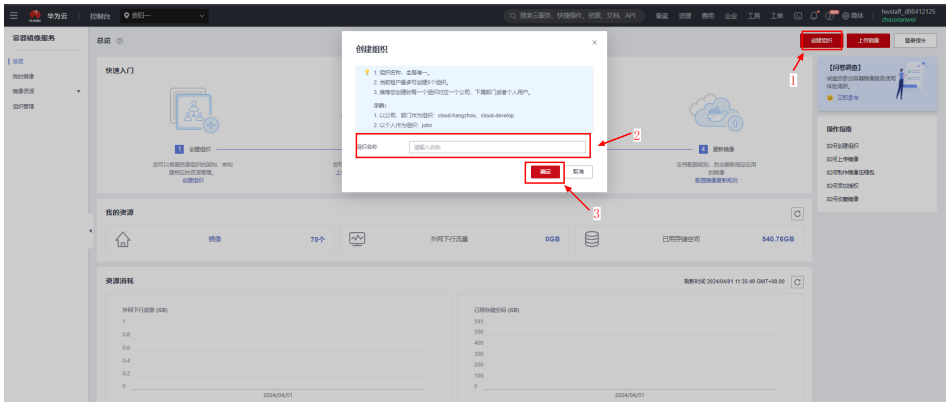
图 5-24 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 5-25 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step4 获取训练镜像

请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表5-11**。

```
docker pull {image_url}
```

5.2.2.5.3 ECS 中构建新镜像

通过**ECS获取基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表5-10**。

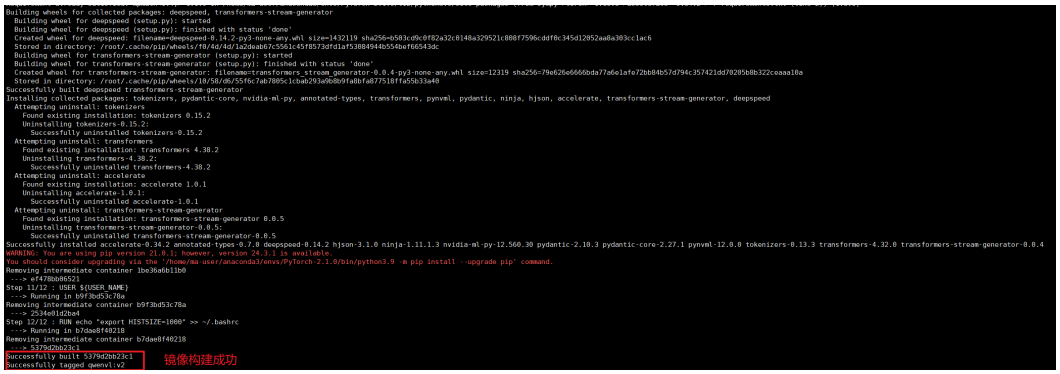
1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-AIGC-6.3.912-xxx.zip，并直接进入到QwenVL/train/<commit_id>文件夹下面

   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-AIGC-*.zip -d ./AscendCloud/AscendCloud-AIGC
   cd ./AscendCloud/AscendCloud-AIGC/multimodal_algorithm/QwenVL/train/<commit_id>
   ```

2. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网

   ```
   docker build -t <镜像名称>:<版本名称> .
   ```

   图 5-26 docker 镜像构建过程

   如果无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。

   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称> .
   ```

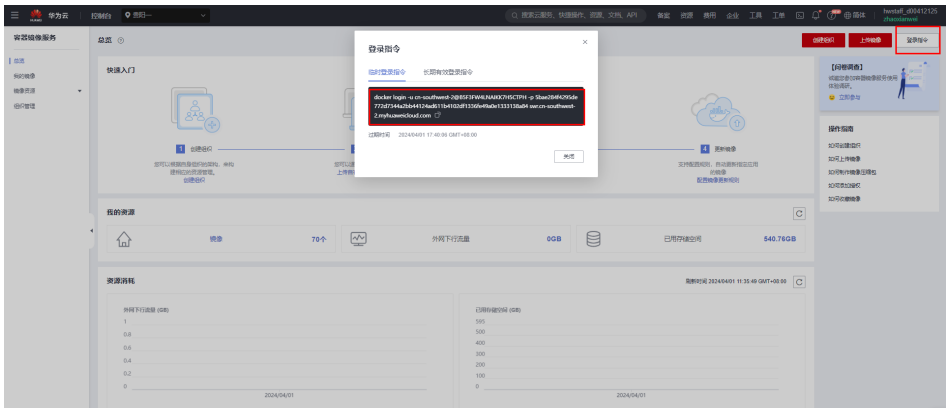
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

5.2.2.5.4 ECS 中上传新镜像

Step1 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 5-27 复制登录指令



Step2 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

   ```
   docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```

   参数说明：

   - ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
   - <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
   - <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

   ```
   docker tag  {image_url}  swr.cn-southwest-2.myhuaweicloud.com/ma-group/
   pytorch_2_1_ascend:20240606
   ```

2. 上传镜像至镜像仓库。

   ```
   docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```

   示例：

   ```
   docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606
   ```

5.2.3 SFT 全参微调训练

前提条件

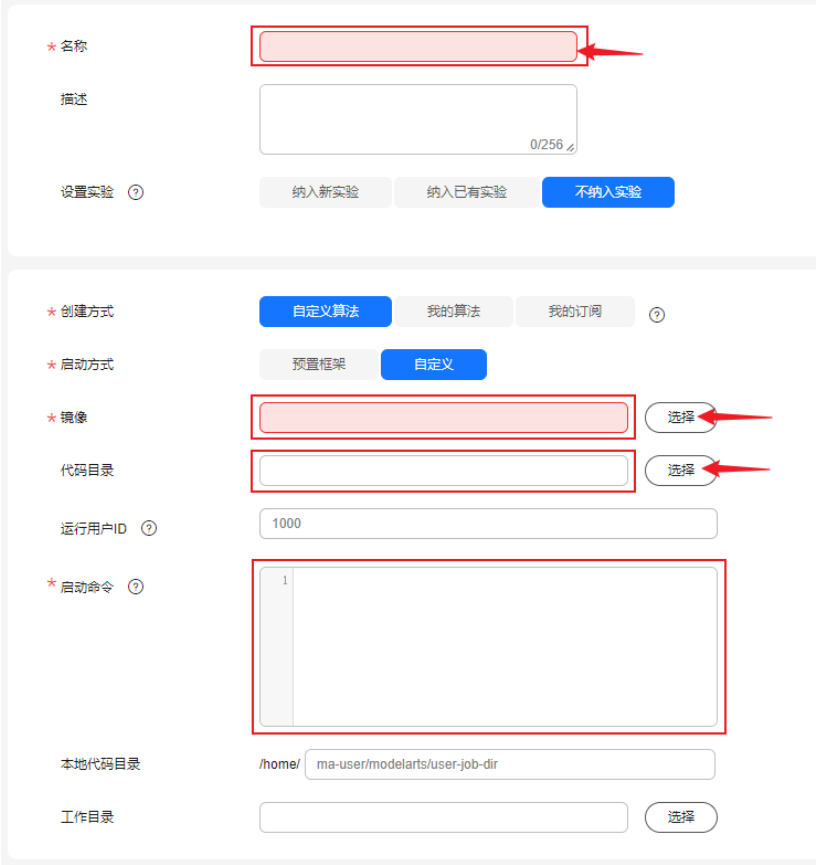
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的mllm_train/train/<commit_id>代码目录。

图 5-28 创建训练作业



训练作业启动命令中输入：

```
cd /home/ma-user/work/Qwen-VL;
ln -s ${DATA}/ qwenvl_dataset;
bash finetune/finetune_ds.sh;
```

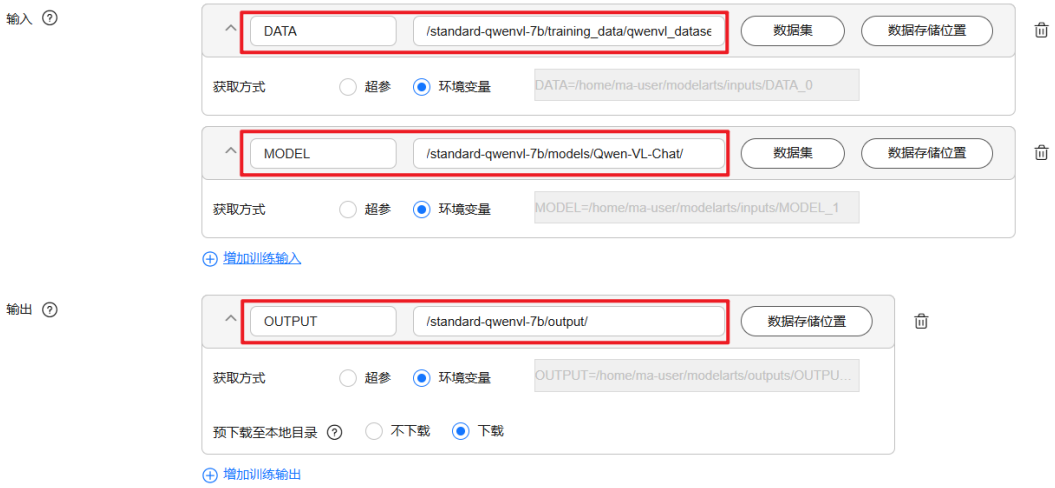
Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：DATA、MODEL。
   - **DATA**：训练数据集的路径/standard-qwenvl-7b/training_data/qwenvl_dataset/。
   - **MODEL**：模型权重的路径/standard-qwenvl-7b/models/Qwen-VL-Chat/。
2. 在“输出”的输入框内设置变量：OUTPUT。

   OUTPUT：训练完成后指定的输出模型的路径/standard-qwenvl-7b/output/。在OBS桶中新建一个output目录，用于训练的输出路径。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。

4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。



Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表5-12表格中的配置进行填写。

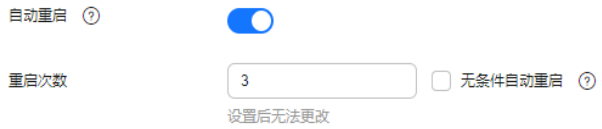
表 5-12 需要填写的环境变量

环境变量	示例值	参数说明
GPUS_PER_NODE	8	**默认必须填写。**根据资源规格每个节点上NPU的数量填写。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。

图 5-29 开启故障重启



Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。本次qwenvl模型选用 8* ascend-snt9b。

图 5-30 选择资源池规格

在OBS中新建一个log目录，作业日志选择OBS中的该路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

5.2.4 LoRA 微调训练

前提条件

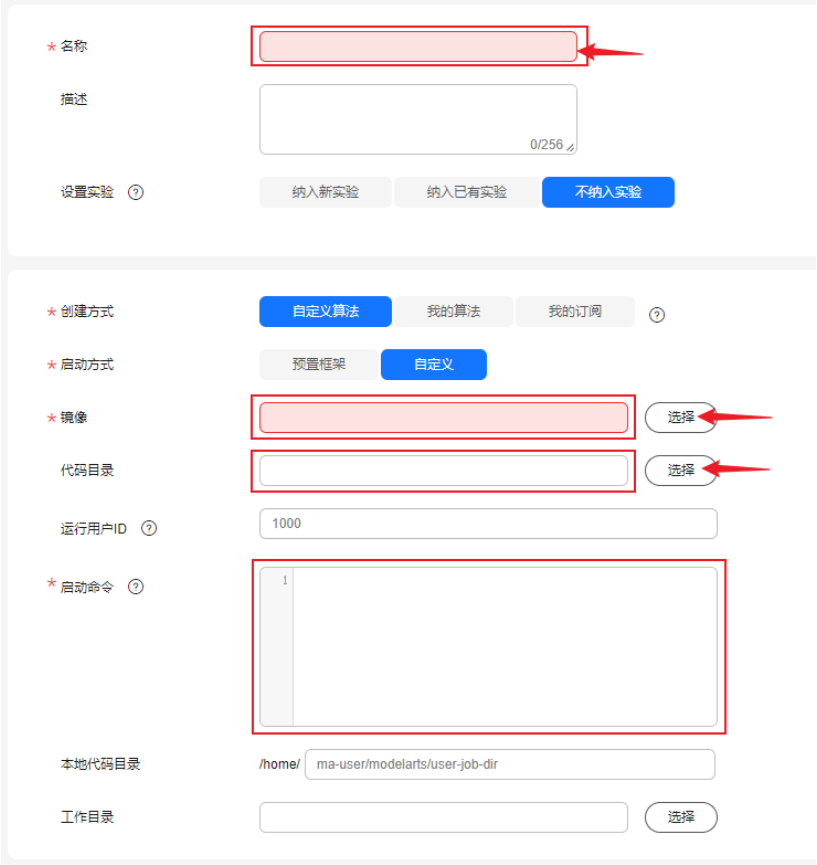
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的mllm_train/train/<commit_id>代码目录。

图 5-31 创建训练作业



训练作业启动命令中输入：

```
cd /home/ma-user/work/Qwen-VL;
ln -s ${DATA}/ qwenvl_dataset;
sh finetune/finetune_lora_ds.sh
```

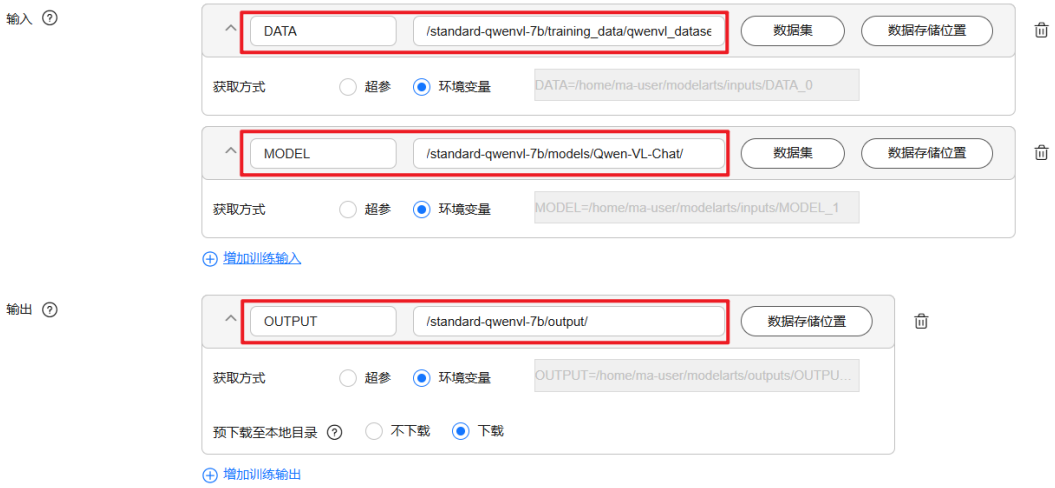
Step2 配置数据输入和输出

单击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：DATA、MODEL。
   - **DATA**：训练数据集的路径/standard-qwenvl-7b/training_data/qwenvl_dataset/。
   - **MODEL**：模型权重的路径/standard-qwenvl-7b/models/Qwen-VL-Chat/。
2. 在“输出”的输入框内设置变量：OUTPUT。

   OUTPUT：训练完成后指定的输出模型的路径/standard-qwenvl-7b/output/。在OBS桶中新建一个output目录，用于训练的输出路径。
3. 分别单击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。

4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。



Step3 配置环境变量

单击“增加环境变量”，在增加的环境变量填写框中，按照表5-13表格中的配置进行填写。

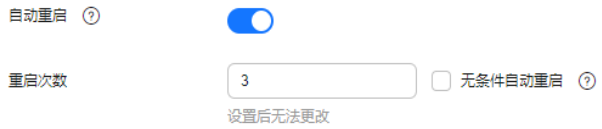
表 5-13 需要填写的环境变量

环境变量	示例值	参数说明
GPUS_PER_NODE	8	**默认必须填写。**根据资源规格每个节点上NPU的数量填写。

Step4 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。

图 5-32 开启故障重启



Step5 其他配置

选择用户自己的专属资源池，以及规格与节点数。本次qwenvl模型选用 8* ascend-snt9b。

图 5-33 选择资源池规格

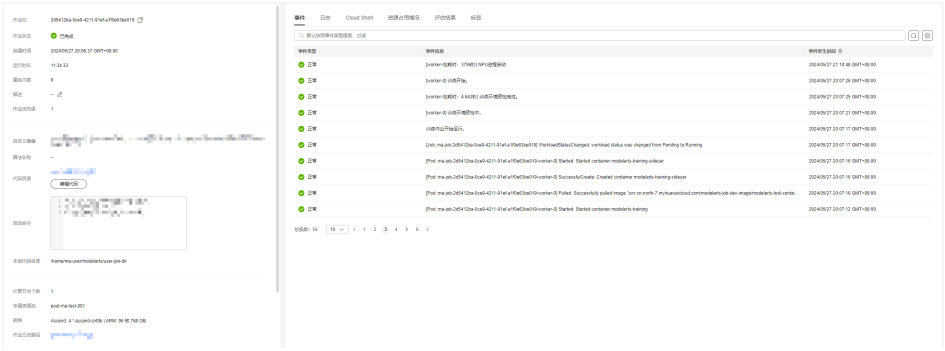
在OBS中新建一个log目录，作业日志选择OBS中的该路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

5.2.5 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

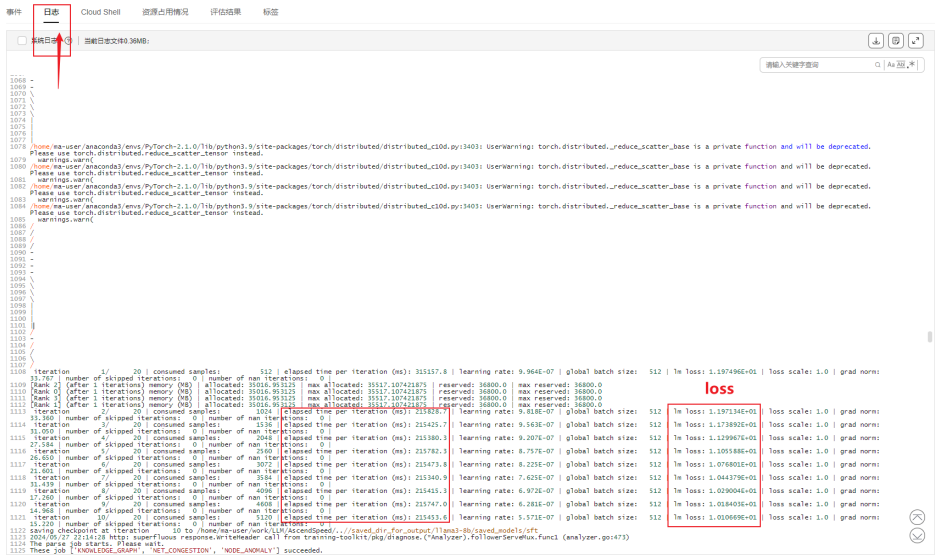
图 5-34 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 5-35 查看日志和性能



5.2.6 训练脚本说明

5.2.6.1 训练脚本存放目录说明

在AscendCloud-AIGC代码包的multimodal_algorithm目录下集成了多个多模态模型的适配脚本，用户可通过不同模型中的xxx_install.sh脚本一键适配。在用户通过Dockerfile构建模型的环境镜像时会执行该脚本，这会从github上拉取模型的官方源码，并通过git apply qwen-vl.patch的方式进行NPU适配，最后将以上源码和环境打包至镜像中。

```
AscendCloud-AIGC-6.3.912-xxx
├─aigc_inference
├─aigc_train
├─AscendCloud-Pytorch-Plugin
└─multimodal_algorithm
   ├─ascendcloud_multimodal_plugin
   │  ...
   └─QwenVL
   ├── train/<commit_id>     # 本教程中需要的模型代码
                └── Dockerfile                  # 构建镜像
                └── qwen-vl_install.sh          # 安装模型运行环境
                └── qwen-vl.patch               # 使用git apply修改模型相关代码
```

5.2.6.2 不同模型推荐的参数与 NPU 卡数设置

表 5-14 不同模型推荐的参数与 NPU 卡数设置

模型	Template	模型参数量	训练策略类型	序列长度 cutoff_len	梯度累积值	优化工具 (Deepspeed)	规格与节点数
Qwen-VL	Qwen-VL	7B	full	2048	gradient_accumulation_steps: 16	ZeRO-3	1*节点 & 8*Ascend
			lora		gradient_accumulation_steps: 8	ZeRO-2	1*节点 & 8*Ascend

5.2.6.3 训练 tokenizer 文件说明

在训练开始前，有些模型需要对模型的tokenizer文件，或者模型的配置文件进行修改，具体的修改如下：

Qwen-VL

1. 修改文件modeling_qwen.py：
```
# 将36 37 两行注释部分
36 SUPPORT_BF16 = SUPPORT_CUDA #and torch.cuda.is_bf16_supported()
37 SUPPORT_FP16 = SUPPORT_CUDA #and torch.cuda.get_device_capability(0)[0] >= 7
```
2. 修改文件tokenization_qwen.py：
```
# tonkenization_qwen.py会在cache中读取SimSun.ttf 文件，如果没有，就会联网下载，ModelArts作业在执行过程中可能不能请求网络，会遇到报错。
# 直接手动下载 https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf ，放到模型权重目录Qwen-VL-Chat下。
# 然后将tokenization_qwen.py中30-35行注释，并重新定义变量FONT_PATH读取字体文件Simsun，修改如下：
30 # FONT_PATH = try_to_load_from_cache("Qwen/Qwen-VL-Chat", "SimSun.ttf")
31 # if FONT_PATH is None:
32 #     if not os.path.exists("SimSun.ttf"):
33 #         ttf = requests.get("https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf")
34 #         open("SimSun.ttf", "wb").write(ttf.content)
35 #     FONT_PATH = "SimSun.ttf"
FONT_PATH = os.path.join(os.getenv('DATA'), "SimSun.ttf")
```

5.2.7 常见错误原因和解决方法

5.2.7.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

```
RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.
```

解决方法：

通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。

5.2.7.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

```
enp67s0f5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.170.22.142  netmask 255.255.255.0  broadcast 10.170.22.255
        inet6 fe80::4ab9:d990:5410:c2a3  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:41:ad:25  txqueuelen 1000  (Ethernet)
        RX packets 4117286148  bytes 5866173345386 (5.3 TiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 356479073  bytes 7356589926408 (6.6 TiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
export GLOO_SOCKET_IFNAME=enp67s0f5   # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5     # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5   # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

5.2.7.3 联网下载 SimSun.ttf 时可能会遇到网络问题

联网下载SimSun.ttf时肯会遇到网络问题

tonkenization_qwen.py会在cache中读取SimSun.ttf 文件，如果没有，就会联网下载，可能会遇到：

1. SSL:CERTIFICATE_VERIFY_FAILED
2. ssl.1129错误
3. 407 Proxy Authentication Required

解决方案：

```
1. 直接手动下载 SimSun.ttf 传到 模型权重目录下 /home/ma-user/work/model-dir/Qwen-VL-Chat/
     https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf

2. 将 文件 /home/ma-user/work/model-dir/Qwen-VL-Chat/tokenization_qwen.py 中的 30-35 行注释
3. 然后增加一行直接读取本地的Simsun.ttf文件，写绝对路径
# FONT_PATH = try_to_load_from_cache("Qwen/Qwen-VL-Chat", "SimSun.ttf")# if FONT_PATH is None:
#     if not os.path.exists("SimSun.ttf"):
#         ttf = requests.get("https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/SimSun.ttf")
#         open("SimSun.ttf", "wb").write(ttf.content)
#     FONT_PATH = "SimSun.ttf"
FONT_PATH = "/home/ma-user/work/model-dir/Qwen-VL-Chat/SimSun.ttf"
```

5.2.7.4 在运行 finetune_ds.sh 时遇到报错

在运行finetune_ds.sh 时遇到报错

```
pydantic_core._pydantic_core.ValidationError: 1 validation error for DeepSpeedZeroConfig
stage3_prefetch_bucket_size
Input should be a valid integer, got a number with a fractional part [type=int_from_float,
input_value=15099494.4, input_type=float]
```

将deepspeed配置文件的 stage3_prefetch_bucket_size 参数值从 auto 改成 整数 15099494

5.3 Qwen-VL 基于 Lite Server 适配 PyTorch NPU 的 Finetune 训练指导(6.3.912)

Qwen-VL是规模视觉语言模型，可以以图像、文本、检测框作为输入，并以文本和检测框作为输出。具有强大的性能、多语言对话、多图交错对话、支持中文开放域定位、细粒度识别和理解等特点。

本文档主要介绍如何利用训练框架PyTorch_npu + 华为自研Ascend Snt9B硬件，完成Qwen-VL Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 5-15 环境要求

名称	版本
CANN	cann_8.0.rc3
驱动	24.1.rc1
PyTorch	2.1.0

获取软件和镜像

表 5-16 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912-xxx.zip软件包中的AscendCloud-AIGC-6.3.912-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练至少需要单机8卡。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**DevServer资源开通**，购买DevServer资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取软件和镜像**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练默认使用单机8卡。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=64g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户。

```
docker exec -it ${container_name} bash
```

Step3 获取代码并上传

上传代码AscendCloud-AIGC-6.3.912-xxx.zip到容器的工作目录${container_work_dir}中，包获取路径请参见**表5-16**。

Step4 准备训练环境

1. 下载权重。从HuggingFace下载**Qwen-VL-Chat**，或将您已下载的权重文件上传到容器工作目录${container_work_dir}中。

```
# 模型结构如下：
Qwen-VL-Chat/
├── config.json
├── configuration_qwen.py
```

```
├── generation_config.jsons
├── modeling_qwen.py
├── pytorch_model-00001-of-00010.bin
├── pytorch_model-00002-of-00010.bin
├── pytorch_model-00003-of-00010.bin
├── pytorch_model-00004-of-00010.bin
├── pytorch_model-00005-of-00010.bin
├── pytorch_model-00006-of-00010.bin
├── pytorch_model-00007-of-00010.bin
├── pytorch_model-00008-of-00010.bin
├── pytorch_model-00009-of-00010.bin
├── pytorch_model-00010-of-00010.bin
├── pytorch_model.bin.index.json
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
├── SimSun.ttf
├── tokenization_qwen.py
├── tokenizer_config.json
└── visual.py
```

2. 赋予容器访问权重文件的权限。上传文件到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 在容器中解压代码包并执行Qwen-VL安装脚本。

```
# 解压代码包
unzip AscendCloud-AIGC-6.3.912-*.zip
rm -rf AscendCloud-AIGC-6.3.912-*
# 执行安装脚本
# model_path 配置为Qwen-VL的权重路径，例：/home/ma-user/Qwen-VL-Chat
git config --global http.sslVerify false
bash multimodal_algorithm/QwenVL/train/aa00ed04091eea5fcdd32985e7915f1c53e7d599/qwen-vl_install.sh
cp -f ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin/ascendcloud_multimodal/train/models/qwenvl/modeling_qwen.py ${container_work_dir}/Qwen-VL-Chat
```

Step5 准备训练数据集

用户需自行制作数据集，并将数据集上传到容器的工作目录中，再赋予容器读写数据集目录的权限。

数据集制作请参考Qwen-VL**官方指导资料**，将所有数据样本放到一个列表中并存入json文件中。每个样本对应一个字典，包含id和conversation，其中后者为一个列表。数据集的json文件示例如下所示。

```
[
 {
  "id": "identity_0",
  "conversations": [
   {
    "from": "user",
    "value": "你好"
   },
   {
    "from": "assistant",
    "value": "我是Qwen-VL,一个支持视觉输入的大模型。"
   }
  ]
 },
 {
  "id": "identity_1",
  "conversations": [
   {
```

```
      "from": "user",
      "value": "Picture 1: <img>https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/demo.jpeg</
img>\n图中的狗是什么品种？"
    },
    {
      "from": "assistant",
      "value": "图中是一只拉布拉多犬。"
    },
    {
      "from": "user",
      "value": "框出图中的格子衬衫"
    },
    {
      "from": "assistant",
      "value": "<ref>格子衬衫</ref><box>(588,499),(725,789)</box>"
    }
  ]
 },
 {
  "id": "identity_2",
  "conversations": [
    {
      "from": "user",
      "value": "Picture 1: <img>assets/mm_tutorial/Chongqing.jpeg</img>\nPicture 2: <img>assets/
mm_tutorial/Beijing.jpeg</img>\n图中都是哪"
    },
    {
      "from": "assistant",
      "value": "第一张图片是重庆的城市天际线，第二张图片是北京的天际线。"
    }
  ]
 }
]
```

- 为针对多样的VL任务，特殊tokens如下：<img> </img> <ref> </ref> <box> </box>。
- 对于带图像输入的内容可表示为Picture id: <img>img_path</img>\n{your prompt}，其中id表示对话中的第几张图片。"img_path"可以是本地的图片或网络地址。
- 对话中的检测框可以表示为<box>(x1,y1),(x2,y2)</box>，其中 (x1, y1) 和(x2, y2)分别对应左上角和右下角的坐标，并且被归一化到[0, 1000)的范围内。检测框对应的文本描述也可以通过<ref>text_caption</ref>表示。

Step6 开始训练

进入代码根目录。

```
cd ${container_work_dir}/Qwen-VL
```

执行训练脚本

```
#配置训练参数
vim finetune/finetune_ds.sh
MODEL: 修改为权重文件实际路径
DATA: 修改为数据集路径
--output_dir： 训练后的权重所在目录名称，默认为output_qwen
--num_train_epochs： 训练轮数，默认为5
#配置修改完成后保存退出，执行训练脚本
bash finetune/finetune_ds.sh  > log_dir/xx.log   #保存训练日志
```

训练后的产物路径说明如下。

```
# 日志路径：
指定${log_dir}
# 训练输出权重路径：
{container_work_dir}/Qwen-VL/output_qwen
```


训练过程中，训练日志会在最后的Rank节点打印。

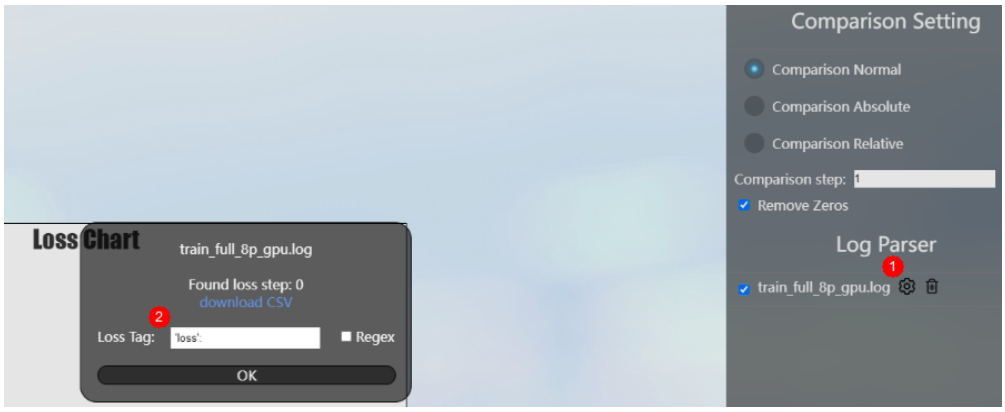
日志里存在loss参数，loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

FAQ

问题：使用**TrainingLogParser**工具解析训练日志中loss数据，坐标栏空白，未显示数据走势曲线。

解决方法：在解析工具页面右侧，单击日志文件名右边的设置图标，在弹出的窗口中修改Loss Tag。将字符串loss加上单引号，改为'loss': ，如**图5-36**所示。

图 5-36 修改 Loss Tag



5.4 Qwen-VL 基于 Lite Server 适配 PyTorch NPU 的推理指导（6.3.909）

Qwen-VL是规模视觉语言模型，可以以图像、文本、检测框作为输入，并以文本和检测框作为输出。具有强大的性能、多语言对话、多图交错对话、支持中文开放域定位、细粒度识别和理解等特点。

本文档主要介绍如何利用训练框架PyTorch_npu + 华为自研Ascend Snt9B硬件，完成Qwen-VL推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

表 5-17 环境要求

名称	版本
PyTorch	pytorch_2.1.0
驱动	23.0.6

获取镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.909-xxx.zip软件包中的AscendCloud-AIGC-6.3.909-xxx.zip，AscendCloud-OPP-6.3.909-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_2_ascend:pytorch_2.2.0-cann_8.0.rc3-py_3.10-hce_2.0.2406-aarch64-snt9b-20240910150953-6faa0ed	从SWR拉取。

约束限制

- 推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取镜像**。
```
docker pull {image_url}
```
2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。推理默认使用单机单卡。
```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```
参数说明：
   - device=/dev/davinci0：挂载NPU设备，示例中挂载了1张卡davinci0。
   - ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
   - shm-size：共享内存大小。
   - ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
   - ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

   说明
   - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
   - driver及npu-smi需同时挂载至容器。
   - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。
```
docker exec -it ${container_name} bash
```

Step3 准备推理环境

1. 下载权重。从HuggingFace下载**Qwen-VL-Chat**，或将您已下载的权重文件上传到容器工作目录${container_work_dir}中。
```
# 模型结构如下：
Qwen-VL-Chat/
├── config.json
├── configuration_qwen.py
├── generation_config.jsons
├── modeling_qwen.py
├── pytorch_model-00001-of-00010.bin
├── pytorch_model-00002-of-00010.bin
├── pytorch_model-00003-of-00010.bin
├── pytorch_model-00004-of-00010.bin
├── pytorch_model-00005-of-00010.bin
├── pytorch_model-00006-of-00010.bin
├── pytorch_model-00007-of-00010.bin
├── pytorch_model-00008-of-00010.bin
```

```
├── pytorch_model-00009-of-00010.bin
├── pytorch_model-00010-of-00010.bin
├── pytorch_model.bin.index.json
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
├── SimSun.ttf
├── tokenization_qwen.py
├── tokenizer_config.json
└── visual.py
```

2. 赋予容器访问权重文件的权限。上传文件到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 在容器中解压代码包并执行Qwen-VL安装脚本。

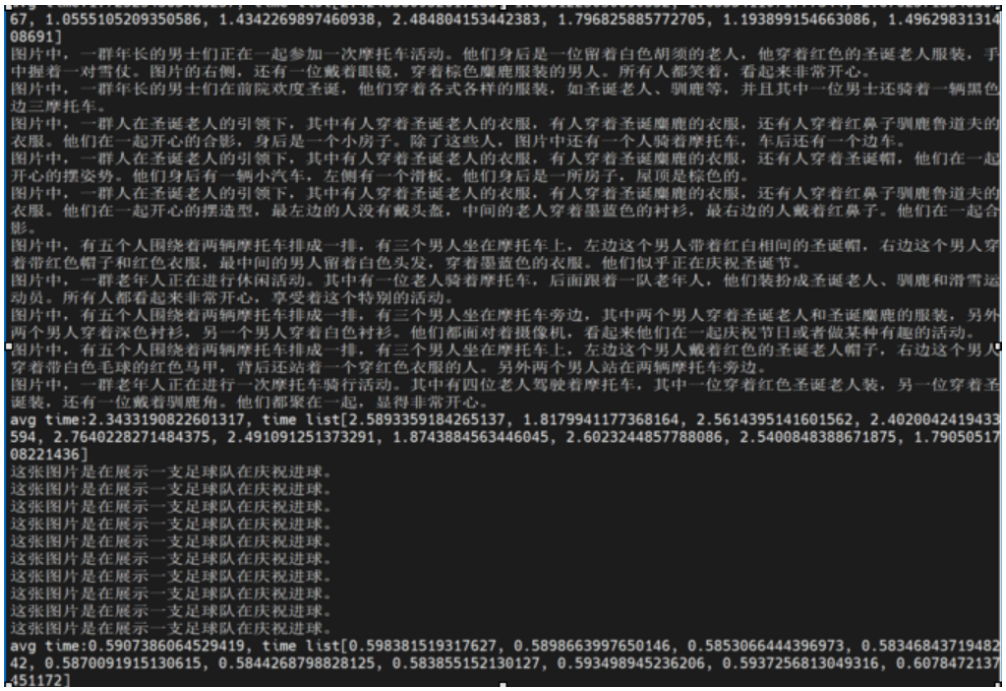
```
# 解压代码包
unzip AscendCloud-AIGC-6.3.909-*.zip
rm -rf AscendCloud-AIGC-6.3.909-*
# 执行安装脚本
# model_path 配置为Qwen-VL的权重路径，例：/home/ma-user/Qwen-VL-Chat
git config --global http.sslVerify false
bash multimodal_algorithm/QwenVL/6d0ab0efd0a/qwen_vl_install.sh {model_path}
# 执行完成后，代码路径为ModelZoo-PyTorch/PyTorch/built-in/mlm/Qwen-VL
# 安装bc命令
sudo yum install -y bc

#安装cann ops算子
unzip AscendCloud-OPP-6.3.909-xxx.zip
pip install ascend_cloud_ops_cann-xx.whl
pip install ascend_cloud_ops_atb-xx.whl
```

Step4 开始推理

在容器工作目录下进到Qwen-VL/infer_test，将要测试的图片放到Qwen-VL/infer_test/images文件夹中，执行如下命令，运行推理脚本。

```
bash infer_demo.sh
```

推理结果如下所示：



5.5 MiniCPM-V2.6 基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.912）

本文档主要介绍如何在ModelArts Lite的Server环境中，使用NPU卡对MiniCPM-V2.6进行LoRA微调及SFT微调。本文档中提供的训练脚本，是基于原生MiniCPM-V的代码基础适配修改，可以用于NPU芯片训练。

方案概览

本方案介绍了在ModelArts的Server上使用昇腾计算资源开展MiniCPM-V 2.6 LoRA训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B单机。

表 5-18 环境要求

名称	版本
CANN	cann_8.0.rc3
驱动	24.1.rc1
PyTorch	2.1.0

获取软件和镜像

表 5-19 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912-xxx.zip软件包中的AscendCloud-AIGC-6.3.912-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表5-19**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-19**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。
- --device=/dev/davinci0 ： 挂载对应卡到容器，当需要挂载多卡，请依次添加多项该配置

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it -u ma-user ${container_name} bash
```

Step4 安装依赖和软件包

1. 从github拉取MiniCPM-V代码。

   ```
   cd /home/ma-user
   git clone https://github.com/OpenBMB/MiniCPM-V.git
   cd /home/ma-user/MiniCPM-V
   git checkout c541f1044e7c0bb2ba48e3eb21daf070e90cd6a2
   ```

2. 获取openbmb/MiniCPM-V-2_6模型。

   https://huggingface.co/openbmb/MiniCPM-V-2_6

   ```
   #手动下载模型权重放置在指定路径
   sudo chown -R ma-user:ma-group ${container_work_dir}
   mkdir -p ${container_work_dir}/minicpm/MiniCPM-V-2_6/
   cp -r MiniCPM-V-2_6 ${container_work_dir}/minicpm/MiniCPM-V-2_6/
   ```

3. 准备coco数据集。

   ```
   cd MiniCPM-V/finetune/
   # Download COCO images
   wget http://images.cocodataset.org/zips/train2014.zip && unzip train2014.zip
   wget http://images.cocodataset.org/zips/val2014.zip && unzip val2014.zip
   ```

4. 制作数据集，参考官网下面链接data preparation章节。

   MiniCPM-V/finetune/readme.md at main · OpenBMB/MiniCPM-V (github.com)

   制成coco2014_train.json文件和coco2014_val.json放在MiniCPM-V/finetune/目录中。json文件示例如下。

   图 5-37 json 文件示例

   coco2014_val_mini...

   ```
   [
     {
       "id": "0",
       "image": "val2014/COCO_val2014_000000522418.jpg",
       "conversations": [
         {
           "role": "user",
           "content": "<image>\nProvide a one-sentence caption for the provided image."
         },
         {
           "role": "assistant",
           "content": "A woman wearing a net on her head cutting a cake."
         }
       ]
     },
     {
       "id": "1",
       "image": "val2014/COCO_val2014_000000522418.jpg",
       "conversations": [
         {
           "role": "user",
           "content": "<image>\nProvide a one-sentence caption for the provided image."
         },
         {
           "role": "assistant",
           "content": "A woman cutting a large white sheet cake."
         }
       ]
     },
     {
       "id": "2",
       "image": "val2014/COCO_val2014_000000522418.jpg",
       "conversations": [
         {
           "role": "user",
           "content": "<image>\nProvide a one-sentence caption for the provided image."
         },
   ```

5. 执行微调脚本前需要补充安装依赖包。

   ```
   pip install accelerate
   pip install tensorboard
   pip install deepspeed==0.15.1
   pip install peft
   pip install numpy==1.24.4
   pip install transformers==4.40.0
   pip install einops
   ```

Step5 MiniCPM-V2.6 微调前修改脚本

使用/home/ma-user/MiniCPM-V/finetune/finetune_lora.sh官方脚本对MiniCPM-V 2.6进行lora微调。使用/home/ma-user/MiniCPM-V/finetune/finetune_ds.sh官方脚本对MiniCPM-V 2.6进行sft微调。微调脚本默认使用 transformers Trainer 和 DeepSpeed。

在 ds_config_zero2.json 修改overlap_comm为false。

```
},

"scheduler": {
    "type": "WarmupLR",
    "params": {
        "warmup_min_lr": "auto",
        "warmup_max_lr": "auto",
        "warmup_num_steps": "auto"
    }
},

"zero_optimization": {
    "stage": 2,
    "offload_optimizer": {
        "device": "none",
        "pin_memory": true
    },
    "allgather_partitions": true,
    "allgather_bucket_size": 2e8,
    "overlap_comm": false,
    "reduce_scatter": true,
    "reduce_bucket_size": 2e8,
    "contiguous_gradients": true
},

"gradient_accumulation_steps": "auto",
"gradient_clipping": "auto",
"steps_per_print": 100,
"train_batch_size": "auto",
"train_micro_batch_size_per_gpu": "auto",
"wall_clock_breakdown": false
}
```

loss固定

```
pip install mindstudio-probe
```

在finetune.py脚本前添加

```
from msprobe.pytorch import seed_all
seed_all(1234)
```

npu

在finetune.py脚本前添加

```
import torch_npu
from torch_npu.contrib import transfer_to_npu
```

下载插件包AscendCloud-AIGC-6.3.912-xxx.zip到${container_work_dir}并解压后得到multimodal_algorithm。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
unzip AscendCloud-AIGC-6.3.909-xxx.zip
cd ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin
pip install -e .

# 在MiniCPM-V/finetune/finetune.py引入优化代码包
from ascendcloud_multimodal.train.models.minicpmv.minicpmv2_6 import ascend_modeling_minicpmv2_6
```

Step6 监督微调

```
bash finetune_ds.sh
```

修改模型权重路径${model_path}，保持其余参数一致。脚本参数设置如下：

```
#!/bin/bash
GPUS_PER_NODE=8
NNODES=1
NODE_RANK=0
MASTER_ADDR=localhost
MASTER_PORT=6001
MODEL=${mdoel_path}
# or openbmb/MiniCPM-V-2, openbmb/MiniCPM-Llama3-V-2_5
# ATTENTION: specify the path to your training data, which should be a json file consisting of a list of conversations.# See the section for finetuning in README for more information.
DATA="coco2014_train.json"
EVAL_DATA="coco2014_val.json"
LLM_TYPE="qwen2" # if use openbmb/MiniCPM-V-2, please set LLM_TYPE=minicpm, if use openbmb/MiniCPM-Llama3-V-2_5, please set LLM_TYPE="llama3"
MODEL_MAX_Length=2048 # if conduct multi-images sft, please set MODEL_MAX_Length=4096
DISTRIBUTED_ARGS="
    --nproc_per_node $GPUS_PER_NODE \
    --nnodes $NNODES \
    --node_rank $NODE_RANK \
    --master_addr $MASTER_ADDR \
    --master_port $MASTER_PORT
"
torchrun $DISTRIBUTED_ARGS finetune.py  \
    --model_name_or_path $MODEL \
    --llm_type $LLM_TYPE \
    --data_path $DATA \
    --eval_data_path $EVAL_DATA \
    --remove_unused_columns false \
    --label_names "labels" \
    --prediction_loss_only false \
    --bf16 true \
    --bf16_full_eval true \
    --fp16 false \
    --fp16_full_eval false \
    --do_train \
    --do_eval \
    --tune_vision true \
    --tune_llm true \
    --model_max_length $MODEL_MAX_Length \
    --max_slice_nums 9 \
    --max_steps 1000 \
    --eval_steps 5000 \
    --output_dir output/output_minicpmv26 \
    --logging_dir output/output_minicpmv26 \
    --logging_strategy "steps" \
    --per_device_train_batch_size 1 \
    --per_device_eval_batch_size 1 \
    --gradient_accumulation_steps 1 \
    --evaluation_strategy "steps" \
    --save_strategy "steps" \
    --save_steps 2000 \
    --save_total_limit 10 \
    --learning_rate 1e-6 \
    --weight_decay 0.1 \
    --adam_beta2 0.95 \
    --warmup_ratio 0.01 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --gradient_checkpointing true \
    --deepspeed ds_config_zero2.json \
    --report_to "tensorboard"
```

Step7 lora 微调

```
bash finetune_lora.sh
```

修改模型权重路径${model_path}，保持其余参数一致。脚本参数设置如下：

```
#!/bin/bash
GPUS_PER_NODE=8
```

```
NNODES=1
NODE_RANK=0
MASTER_ADDR=localhost
MASTER_PORT=6001
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

MODEL=${mdoel_path} # or openbmb/MiniCPM-V-2, openbmb/MiniCPM-Llama3-V-2_5
# ATTENTION: specify the path to your training data, which should be a json file consisting of a list of conversations.
# See the section for finetuning in README for more information.
DATA="coco2014_train.json"
EVAL_DATA="coco2014_val.json"
LLM_TYPE="qwen2"
# if use openbmb/MiniCPM-V-2, please set LLM_TYPE=minicpm#if use openbmb/MiniCPM-Llama3-V-2_5, please set LLM_TYPE=llama3 MODEL_MAX_Length=2048
# if conduct multi-images sft, please set MODEL_MAX_Length=4096
MODEL_MAX_Length=2048

DISTRIBUTED_ARGS="
    --nproc_per_node $GPUS_PER_NODE \
    --nnodes $NNODES \
    --node_rank $NODE_RANK \
    --master_addr $MASTER_ADDR \
    --master_port $MASTER_PORT
"
torchrun $DISTRIBUTED_ARGS finetune.py  \
   --model_name_or_path $MODEL \
   --llm_type $LLM_TYPE \
   --data_path $DATA \
   --eval_data_path $EVAL_DATA \
   --remove_unused_columns false \
   --label_names "labels" \
   --prediction_loss_only false \
   --bf16 true \
   --bf16_full_eval true \
   --fp16 false \
   --fp16_full_eval false \
   --do_train \
   --do_eval \
   --num_train_epochs 1 \
   --tune_vision true \
   --tune_llm false \
   --use_lora true \
   --lora_target_modules "llm\..*layers\.\d+\.self_attn\.(q_proj|k_proj|v_proj|o_proj)" \
   --model_max_length $MODEL_MAX_Length \
   --max_slice_nums 9 \
   --max_steps 1000 \
   --eval_steps 10000 \
   --output_dir output/output__lora \
   --logging_dir output/output_lora \
   --logging_strategy "steps" \
   --per_device_train_batch_size 1 \
   --per_device_eval_batch_size 1 \
   --gradient_accumulation_steps 1 \
   --evaluation_strategy "steps" \
   --save_strategy "steps" \
   --save_steps 10000 \
   --save_total_limit 10 \
   --learning_rate 1e-6 \
   --weight_decay 0.1 \
   --adam_beta2 0.95 \
   --warmup_ratio 0.01 \
   --lr_scheduler_type "cosine" \
   --logging_steps 1 \
   --gradient_checkpointing true \
   --deepspeed ds_config_zero2.json \
   --report_to "tensorboard"
```

5.6 MiniCPM-V2.0 推理及 LoRA 微调基于 Lite Server 适配 PyTorch NPU 指导（6.3.910）

本文档主要介绍如何在ModelArts Lite的Server环境中，使用NPU卡对MiniCPM-V2.0进行LoRA微调及推理。本文档中提供的训练脚本，是基于原生MiniCPM-V的代码基础适配修改，可以用于NPU芯片训练。

MiniCPM系列的最新多模态版本MiniCPM-V2.0。该模型基于**MiniCPM 2.4B**和SigLip-400M构建，共拥有2.8B参数。MiniCPM-V2.0具有领先的光学字符识别（OCR）和多模态理解能力。该模型在综合性OCR能力评测基准OCRBench上达到开源社区的最佳水平，甚至在场景文字理解方面实现接近 Gemini Pro 的性能。

MiniCPM-V2.0值得关注的特性包括:

1. **领先的 OCR 和多模态理解能力。**MiniCPM-V2.0显著提升了OCR和多模态理解能力，场景文字理解能力接近Gemini Pro，在多个主流评测基准上性能超过了更大参数规模（例如 17-34B）的主流模型。
2. **可信行为。**MiniCPM-V2.0是第一个通过多模态RLHF对齐的端侧多模态大模型（借助**RLHF-V** [CVPR'24] 系列技术）。该模型在Object HalBench 达到和GPT-4V相仿的性能。
3. **任意长宽比高清图像高效编码。** MiniCPM-V2.0可以接受180万像素的任意长宽比图像输入（基于最新的 **LLaVA-UHD** 技术），这使得模型可以感知到小物体、密集文字等更加细粒度的视觉信息。
4. **高效部署。**MiniCPM-V2.0可以高效部署在大多数消费级显卡、个人电脑以及移动手机等终端设备。
5. **双语支持。**MiniCPM-V2.0提供领先的中英双语多模态能力支持。 该能力通过**VisCPM** [ICLR'24]论文中提出的多模态能力的跨语言泛化技术实现。

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展MiniCPM-V2.0 LoRA训练的详细过程，及一份推理示例代码。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B单机。

表 5-20 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 5-21 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.910-xxx.zip软件包中的AscendCloud-AIGC-6.3.910-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.910版本，请参考**表5-21**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-21**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下可存放项目所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。
- --device=/dev/davinci0：挂载对应卡到容器，当需要挂载多卡，请依次添加多项该配置

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step4 安装依赖和软件包

1. git clone和git lfs下载大模型可以参考如下操作。
   a. 由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到容器的/home/ma-user目录下
      ```
      https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
      ```
      或直接下载到容器，这样在容器中可以直接使用。
      ```
      cd /home/ma-user
      wget https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
      ```
   b. 进入容器，执行安装git lfs命令。
      ```
      cd /home/ma-user
      tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
      cd git-lfs-3.2.0
      sudo sh install.sh
      ```
   c. 设置git配置去掉ssl校验。
      ```
      git config --global http.sslVerify false
      ```
2. 从github拉取MiniCPM-V代码。
   ```
   cd /home/ma-user
   git clone https://github.com/OpenBMB/MiniCPM-V.git
   cd /home/ma-user/MiniCPM-V
   git checkout ba27a162aa236e17036bb903
   ```
3. 获取openbmb/MiniCPM-V-2模型。
   ```
   cd /home/ma-user
   git clone https://huggingface.co/openbmb/MiniCPM-V-2
   cd /home/ma-user/MiniCPM-V-2
   git checkout ee00ff7ce36667e7df81cb2a0
   ```
4. 安装MiniCPM-V2_0 Ascend软件包。
   a. 将获取到的MiniCPM-V Ascend软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径参见表5-21。
   b. 解压AscendCloud-AIGC-*.zip文件，解压后将里面指定文件与对应MiniCPM-V文件进行替换。
      ```
      cd  /home/ma-user
      unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
      cd AscendCloud/multimodal_algorithm/MiniCPM-V2_0/
      dos2unix install.sh
      bash install.sh
      ```

      说明

      AscendCloud-AIGC-*.zip后面的*表示时间戳，请按照实际替换。

      MiniCPM-V2_0 Ascend软件包内容如下：
      ```
      |---MiniCPM-V2_0/
          --- infer.py      推理示例代码
          --- install.sh    安装torch-npu适配修改及优化脚本
          --- model_modify.patch    优化模型融合算子git patch文件
          --- modify.patch          适配优化MiniCPM-V2.0代码git patch文件
          --- README.md             适配文档基于官方代码commit id说明
      ```

Step5 MiniCPM-V2.0 微调

使用/home/ma-user/MiniCPM-V/finetune/finetune_lora.sh官方脚本对MiniCPM-V2.0进行微调。微调脚本默认使用 transformers Trainer 和 DeepSpeed。

1. **数据准备**

   要准备微调数据，您应该将每个样本制定为一个字典，其中包含一个 ID、一个图像路径（或图像列表）和一个对话列表。然后，将数据样本保存在 JSON 文件中。

对于视觉语言任务，您必须提供占位符（例如<image>或<image_XX>）来定义在对话中插入图像嵌入的位置。如果没有提供占位符，则图像将默认放置在对话的前面。

单幅图像示例

如果您的输入仅包含一张图片，则可以使用单个占位符<image>来指示应在对话中插入图像的位置。

包含 1 个样本的单个图像示例：

```
[
  {
    "id": "0",
    "image": 'path/to/image_0.jpg',
    "conversations": [
          {
            'role': 'user',
            'content': '<image>\nHow many desserts are on the white plate?'
          },
          {
             'role': 'assistant',
             'content': 'There are three desserts on the white plate.'
          },
          {
             'role': 'user',
             'content': 'What type of desserts are they?'
          },
          {
             'role': 'assistant',
             'content': 'The desserts are cakes with bananas and pecans on top. They share similarities with donuts, but the presence of bananas and pecans differentiates them.'
          },
          {
             'role': 'user',
             'content': 'What is the setting of the image?'},
          {
             'role': 'assistant',
             'content': 'The image is set on a table top with a plate containing the three desserts.'
          },
      ]
  },
]
```

2. **LoRA微调**

   需要在整个训练过程中更新LLM的所有参数，请在shell脚本中指定正确的MODEL路径、DATA路径和LLM_TYPE。

   ```
   MODEL="/path/to/openbmb/MiniCPM-V-2" # or openbmb/MiniCPM-V-2
   # ATTENTION: specify the path to your training data, which should be a json file consisting of a list of conversations.
   # See the section for finetuning in README for more information.
   DATA="path/to/trainging_data"
   EVAL_DATA="path/to/test_data"
   LLM_TYPE="minicpm" # if use openbmb/MiniCPM-V-2, please set LLM_TYPE=minicpm
   ```

   LoRA 允许轻量级模型调整，只需更新一小部分参数。要启动训练，请运行以下脚本：

   ```
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
   bash finetune_lora.sh
   ```

注：以上微调文档提示来自官方文档，有关可用LoRA微调脚本参数及其功能的全面文档，您可以参考**官方MiniCPM-V文档**。

Step6 MiniCPM-V2.0 推理

多轮对话

请参考以下代码进行推理infer.py，此示例文件附于MiniCPM-V2_0 Ascend软件包内。

```
cd /home/ma-user/MiniCPM-V
python infer.py
# 执行python infer.py即可得到示例参考结果：
=======First round answer======
The aircraft is an Airbus A380-Boeing 747.
=======Second round answer======
The Airbus A380 is a large passenger jet designed for long-haul flights. It has four engines, making it one of the most powerful commercial aircraft ever built with an impressive capacity to carry up to 524 passengers in its standard configuration or even more than that depending on seating arrangements and cabin layouts.
```

infer.py示例代码如下：

```
import json
from chat import MiniCPMVChat, img2base64
import torch_npu
from torch_npu.contrib import transfer_to_npu
model_type = "/home/ma-user/MiniCPM-V-2"  # or openbmb/MiniCPM-V-2

chat_model = MiniCPMVChat(model_type)
im_64 = img2base64('./assets/airplane.jpeg')
# First round chat
msgs = [{"role": "user", "content": "Tell me the model of this aircraft."}]
inputs = {"image": im_64, "question": json.dumps(msgs)}
answer = chat_model.chat(inputs)
print("=======First round answer======")
print(answer)
# Second round chat
# pass history context of multi-turn conversation
msgs.append({"role": "assistant", "content": answer})
msgs.append({"role": "user", "content": "Introduce something about Airbus A380."})
inputs = {"image": im_64, "question": json.dumps(msgs)}
answer = chat_model.chat(inputs)
print("=======Second round answer======")
print(answer)
```

5.7 InternVL2 基于 LIte Server 适配 PyTorch NPU 训练指导（6.3.912）

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展InternVL2-8B, InternVL2-26B和InternVL2-40B模型的训练过程，包括Finetune训练和LoRA训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.912版本，请参考获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。

获取软件和镜像

表 5-22 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.910软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

表 5-23 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表5-22**。

```
docker pull {image_url}
```

步骤三：启动容器镜像

启动容器镜像，启动前可以根据实际需要增加修改参数。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \${image_id}  \
/bin/bash
```

- --device=/dev/davinciX 挂载NPU设备，示例中挂载了1张卡
- work_dir：工作目录，目录下存放着训练所需代码、数据等文件
- container_work_dir: 容器工作目录，一般同work_dir
- container_name：自定义容器名
- image_id：镜像ID，通过docker images来查看拉取的镜像ID。

步骤四：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it  ${container_name} bash
```

修改权限。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
```

此步骤可能需要密码或root权限。

步骤五：下载代码安装环境

下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，获取路径参见**表5-22**。

```
unzip AscendCloud-AIGC-6.3.912-*.zip
bash multimodal_algorithm/InternVL2/train/5d8f485ad09b3eb9b2a7d9a24cca727fa58bb775/InternVL2_install.sh
cp multimodal_algorithm/InternVL2/train/5d8f485ad09b3eb9b2a7d9a24cca727fa58bb775/shells/* InternVL/internvl_chat/shell/internvl2.0/2nd_finetune/
```

步骤六：增加适配代码

表 5-24 添加优化代码

模型	使用方法

internVL2-40B	internVL2-40B模型需要执行下列步骤。
	`cd ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin`
	`pip install -e .`
	`cd ${container_work_dir}`
	修改InternVL/internvl_chat/internvl/train/internvl_chat_finetune.py 文件，加入如下命令，用于引入优化代码包。
	`from ascendcloud_multimodal.train.models.internvl2 import ascend_modeling_internvl`
	执行如下命令添加优化代码。
	`cp -rf multimodal_algorithm/ascendcloud_multimodal_plugin/ascendcloud_multimodal/train/models/internvl2/modeling_intern_vit.py ${container_work_dir}/InternVL/internvl_chat/internvl/model/internvl_chat/modeling_intern_vit.py`

internVL2-8B或
internVL2-26B
	internVL2-8B或internVL2-26B模型需要执行如下命令添加优化代码。
	`cd ${container_work_dir}`
	`cp -rf multimodal_algorithm/ascendcloud_multimodal_plugin/ascendcloud_multimodal/train/models/internvl2/modeling_intern_vit.py ${container_work_dir}/InternVL/internvl_chat/internvl/model/internvl_chat/modeling_intern_vit.py`
	`cp -rf multimodal_algorithm/ascendcloud_multimodal_plugin/ascendcloud_multimodal/train/models/internvl2/modeling_internlm2.py ${container_work_dir}/InternVL/internvl_chat/internvl/model/internlm2/modeling_internlm2.py`

步骤七：下载数据集

先创建文件夹用来存放数据集，再下载数据集。

```
cd ${container_work_dir}/InternVL/internvl_chat
mkdir -p data/coco && cd data/coco
# Download COCO images
wget http://images.cocodataset.org/zips/train2014.zip && unzip train2014.zip
wget http://images.cocodataset.org/zips/val2014.zip && unzip val2014.zip
wget http://images.cocodataset.org/zips/test2015.zip && unzip test2015.zip
mkdir -p annotations && cd annotations/ # Download converted annotation files
wget https://github.com/OpenGVLab/InternVL/releases/download/data/coco_karpathy_test.json
wget https://github.com/OpenGVLab/InternVL/releases/download/data/coco_karpathy_test_gt.json
wget https://github.com/OpenGVLab/InternVL/releases/download/data/coco_karpathy_train_567k.zip
unzip coco_karpathy_train_567k.zip
cd ../../
```

步骤八：下载模型权重

模型权重(可选择)InternVL2-8B， InternVL2-26B， InternVL2-40B

手动下载放置在${container_work_dir}/InternVL/internvl_chat/pretrained路径下，pretrained目录需手动创建。

https://huggingface.co/OpenGVLab/InternVL2-8B/tree/main

https://huggingface.co/OpenGVLab/InternVL2-26B/tree/main

https://huggingface.co/OpenGVLab/InternVL2-40B/tree/main

步骤九：开始训练

单机训练

```
cd ${container_work_dir}/InternVL/internvl_chat
# 8B全参微调
GPUS=8 PER_DEVICE_BATCH_SIZE=2 sh shell/internvl2.0/2nd_finetune/
internvl2_8b_internlm2_7b_dynamic_res_2nd_finetune_full.sh
# 8Blora微调
GPUS=8 PER_DEVICE_BATCH_SIZE=2 sh shell/internvl2.0/2nd_finetune/
internvl2_8b_internlm2_7b_dynamic_res_2nd_finetune_lora.sh
# 26Blora微调
GPUS=8 PER_DEVICE_BATCH_SIZE=2 sh shell/internvl2.0/2nd_finetune/
internvl2_26b_internlm2_20b_dynamic_res_2nd_finetune_lora.sh
# 40Blora微调
GPUS=8 PER_DEVICE_BATCH_SIZE=2 sh shell/internvl2.0/2nd_finetune/
internvl2_40b_hermes2_yi_34b_dynamic_res_2nd_finetune_lora.sh
```

多机训练

```
cd ${container_work_dir}/InternVL/internvl_chat
# 8B lora
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_8b_internlm2_7b_dynamic_res_2nd_finetune_lora_multi.sh
# 8B full
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_8b_internlm2_7b_dynamic_res_2nd_finetune_full_multi.sh
# 26B lora
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_26b_internlm2_20b_dynamic_res_2nd_finetune_lora_multi.sh
# 26B full
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_26b_internlm2_20b_dynamic_res_2nd_finetune_full_multi.sh
# 40B lora
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_40b_hermes2_yi_34b_dynamic_res_2nd_finetune_lora_multi.sh
# 40B full
GPUS=8 PER_DEVICE_BATCH_SIZE=2 NNODES=${NODE_NUM} NODE_RANK=${NODE_RANK}
MASTER_ADDR="${master_node_ip}" sh shell/internvl2.0/2nd_finetune/
internvl2_40b_hermes2_yi_34b_dynamic_res_2nd_finetune_full_multi.sh
```

参数说

- NODE_NUM：机器数量。
- NODE_RANK：机器rank num，主机为0，其余递增。
- MASTER_ADDR：主机IP地址。
- GPUS: 每台机器npu卡数
- PER_DEVICE_BATCH_SIZE：每张卡batch size

训练成功如下图所示。

图 5-38 训练成功

5.8 LLaVA-NeXT 基于 Lite Server 适配 PyTorch NPU 训练微调指导（6.3.912）

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展LLaVA-NeXT模型的训练过程，包括pretrain_clip训练和Finetune_onevision训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.912版本，请参考获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。

获取软件和镜像

表 5-25 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

表 5-26 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	24.1.rc1
PyTorch	2.1.0

步骤一 检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表1 获取软件和镜像**。

```
docker pull {image_url}
```

步骤三 启动容器镜像

启动容器镜像，启动前可以根据实际需要增加修改参数。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
```

```
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

- --device=/dev/davinciX 挂载NPU设备，示例中挂载了8张卡
- work_dir：工作目录，目录下存放着训练所需代码、数据等文件
- container_work_dir: 容器工作目录，一般同work_dir
- container_name：自定义容器名
- image_id：镜像ID，通过docker images来查看拉取的镜像ID。

步骤四 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it  ${container_name} bash
```

修改权限。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
```

此步骤可能需要密码或root权限。

步骤五 下载代码安装环境

下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，获取路径参见**表1 获取软件和镜像**。

```
mv AscendCloud-AIGC-6.3.912-xxx.zip ${container_work_dir}
cd ${container_work_dir}
unzip AscendCloud-AIGC-6.3.912-*.zip
cd multimodal_algorithm/LLAVA-NEXT/train/c7cc95c0ed68ee553cf0870b6684695df609bb38
bash llava_next_install.sh
cp pretrain_clip_ascend.sh finetune_onevision_ascend.sh ./LLaVA-NeXT/scripts/train
```

步骤六 增加适配代码

```
# 安装优化加速包
cd ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin
pip install -e .

# 使能优化加速包step1(此步默认在环境安装阶段已完成)
cd ${container_work_dir}/multimodal_algorithm/LLAVA-NEXT/train/
c7cc95c0ed68ee553cf0870b6684695df609bb38/LLaVA-NeXT/
在./llava/train/train_mem.py 引入优化代码包 from
ascendcloud_multimodal.train.models.llava_next.ascend_modeling_llava_next import *

# 使能优化加速包step2
cp -rf ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin/
ascendcloud_multimodal/train/models/llava_next/siglip_encoder.py ${container_work_dir}/
```

```
multimodal_algorithm/LLAVA-NEXT/train/c7cc95c0ed68ee553cf0870b6684695df609bb38/LLaVA-NeXT/llava/model/multimodal_encoder/
```

步骤七 下载数据集

数据集需从huggingface下载**LLaVA-Pretrain**，**VideoGPT-plus_Training_Dataset**(其中的**vcg-plus_112K.json**和**activitynet_videos.tgz**)。

方式1：手动下载以上所列数据集，并将其放置在${container_work_dir}/data路径下， data目录需手动创建。

方式2：利用git下载，须确保git lfs已成功安装：

```
mkdir -p ${container_work_dir}/data && cd ${container_work_dir}/data
# 下载pretrain_clip场景的数据集
git clone https://huggingface.co/datasets/liuhaotian/LLaVA-Pretrain
cd LLaVA-Pretrain
git lfs pull
# 待下载成功后，解压文件
unzip images.zip

# 下载finetune_onevision场景的数据集
cd ${container_work_dir}/data
git clone https://huggingface.co/datasets/MBZUAI/VideoGPT-plus_Training_Dataset
cd VideoGPT-plus_Training_Dataset
git lfs pull --include="annotations/vcg-plus_112K.json"
git lfs pull --include="instruction_tuning/activitynet_videos.tgz"
# 待下载成功后，解压文件
cd ${container_work_dir}/data/VideoGPT-plus_Training_Dataset/instruction_tuning
tar -xzvf activitynet_videos.tgz
```

步骤八 下载模型权重

模型权重需从huggingface准备**Qwen2-7B-Instruct**，**clip-vit-large-patch14-336**，**siglip-so400m-patch14-384**。

方式1：手动下载以上所列权重，并将其放置在${container_work_dir}/pretrained路径下， pretrained目录需手动创建。

方式2：利用git下载，须确保git lfs已成功安装：

```
mkdir -p ${container_work_dir}/pretrained
# 下载 Qwen2-7B-Instruct
cd ${container_work_dir}/pretrained
git clone https://huggingface.co/Qwen/Qwen2-7B-Instruct
cd Qwen2-7B-Instruct
git lfs pull

# 下载 clip-vit-large-patch14-336(pretrain_clip场景)
cd ${container_work_dir}/pretrained
git clone https://huggingface.co/openai/clip-vit-large-patch14-336
cd clip-vit-large-patch14-336
git lfs pull

# 下载 siglip-so400m-patch14-384(finetune_onevision场景)
cd ${container_work_dir}/pretrained
git clone https://huggingface.co/google/siglip-so400m-patch14-384
cd siglip-so400m-patch14-384
git lfs pull
```

步骤九 开始训练

单机训练

```
cd ${container_work_dir}/multimodal_algorithm/LLAVA-NEXT/train/c7cc95c0ed68ee553cf0870b6684695df609bb38/LLaVA-NeXT
```

```
# pretrain_clip场景
NUM_GPUS=8 NNODES=1 RANK=0 ADDR=localhost PORT=23245 bash scripts/train/pretrain_clip_ascend.sh
# 需修改pretrain_clip_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径

# finetune_onevision场景
NUM_GPUS=8 NNODES=1 RANK=0 ADDR=localhost PORT=23245 bash scripts/train/
finetune_onevision_ascend.sh # 需修改finetune_onevision_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径
```

多机训练

```
cd ${container_work_dir}/multimodal_algorithm/LLAVA-NEXT/train/
c7cc95c0ed68ee553cf0870b6684695df609bb38/LLaVA-NeXT
# pretrain_clip场景
NUM_GPUS=8 NNODES=${NODE_NUM} RANK=${NODE_RANK} ADDR=${MASTER_NODE_IP} PORT=23245
bash scripts/train/pretrain_clip_ascend.sh # 需修改pretrain_clip_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径

# finetune_onevision场景
NUM_GPUS=8 NNODES=${NODE_NUM} RANK=${NODE_RANK} ADDR=${MASTER_NODE_IP} PORT=23245
bash scripts/train/finetune_onevision_ascend.sh # 需修改finetune_onevision_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径
```

路径修改说明：

- 执行训练脚本前，需修改pretrain_clip_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径，如图5-39所示；
- 执行训练脚本前，修改finetune_onevision_ascend.sh中的数据集和模型路径为步骤七和步骤八的下载完成后的路径，如图5-40所示。

图 5-39 pretrain_clip 场景模型路径和数据集路径指引

```
export OMP_NUM_THREADS=8
export NCCL_IB_DISABLE=0
export NCCL_IB_GID_INDEX=3
export NCCL_SOCKET_IFNAME=eth0
export NCCL_DEBUG=INFO

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export CPU_AFFINITY_CONF=1
export HOST_CACHE_CAPACITY=10

LLM_VERSION="path/to/Qwen2-7B-Instruct"
LLM_VERSION_CLEAN="${LLM_VERSION//\//_}"
VISION_MODEL_VERSION="path/to/clip-vit-large-patch14-336"
VISION_MODEL_VERSION_CLEAN="${VISION_MODEL_VERSION//\//_}"

############### Pretrain ################

PROMPT_VERSION=plain

BASE_RUN_NAME="llavanext-${VISION_MODEL_VERSION_CLEAN}-${LLM_VERSION_CLEAN}-mlp2x_gelu-pretrain_blip558k_plain"
echo "BASE_RUN_NAME: ${BASE_RUN_NAME}"

ACCELERATE_CPU_AFFINITY=1 torchrun --nproc_per_node="${NUM_GPUS}" --nnodes="${NNODES}" --node_rank="${RANK}" --master_addr="${ADDR}" --master_port="${PORT}" \
    llava/train/train_mem.py \
    --deepspeed scripts/zero3.json \
    --model_name_or_path ${LLM_VERSION} \
    --version ${PROMPT_VERSION} \
    --data_path path/to/LLaVA-Pretrain/blip_laion_cc_sbu_558k.json \
    --image_folder path/to/LLaVA-Pretrain/images \
    --vision_tower ${VISION_MODEL_VERSION} \
    --mm_tunable_parts="mm_mlp_adapter" \
    --mm_vision_select_layer -2 \
    --mm_projector_type mlp2x_gelu \
    --mm_use_im_start_end False \
    --mm_use_im_patch_token False \
    --bf16 True \
    --output_dir ./checkpoints/projectors/${BASE_RUN_NAME} \
    --num_train_epochs 1 \
    --per_device_train_batch_size 16 \
    --per_device_eval_batch_size 4 \
    --gradient_accumulation_steps 1 \
    --evaluation_strategy "no" \
    --save_strategy "no" \
    --save_steps 50000 \
    --learning_rate 1e-3 \
    --weight_decay 0. \
    --warmup_ratio 0.03 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --tf32 False \
    --model_max_length 8192 \
    --gradient_checkpointing True \
    --dataloader_num_workers 16 \
    --lazy_preprocess True \
    --run_name $BASE_RUN_NAME \
    --attn_implementation eager
```

图 5-40 finetune_onevision 场景模型路径和数据集路径指引

```
export OMP_NUM_THREADS=24
export NCCL_IB_DISABLE=0
export NCCL_IB_GID_INDEX=3
export NCCL_SOCKET_IFNAME=eth0
export NCCL_DEBUG=INFO

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export CPU_AFFINITY_CONF=1
export HOST_CACHE_CAPACITY=10

LLM_VERSION="path/to/Qwen2-7B-Instruct"

# for 7b model we recommend bs=1, accum=2, 16 nodes, 128 gpus, lr=1e-5, warmup=0.03
# for 72b model we recommend bs=1, accum=1, 32 nodes, 256 gpus, lr=1e-5, warmup=0.03
LLM_VERSION_CLEAN="${LLM_VERSION//\//_}"

VISION_MODEL_VERSION="/path/to/siglip-so400m-patch14-384"
VISION_MODEL_VERSION_CLEAN="${VISION_MODEL_VERSION//\//_}"

NUM_GPUS=8
NNODES=1
RANK=0
ADDR=localhost
PORT=23245

############### Pretrain ################

PROMPT_VERSION="qwen_1_5"

BASE_RUN_NAME="llavanext-${VISION_MODEL_VERSION_CLEAN}-${LLM_VERSION_CLEAN}-mlp2x_gelu-pretrain_blip558k_plain"
echo "BASE_RUN_NAME: ${BASE_RUN_NAME}"

MID_RUN_NAME="llavanext-${VISION_MODEL_VERSION_CLEAN}-${LLM_VERSION_CLEAN}-mlp2x_gelu-pretrain_blip558k_plain"
CKPT_PATH=$LLM_VERSION # this could also be the previous stage
OUTPUT_DIR="./checkpoints/${MID_RUN_NAME}"
if [ ! -d "$OUTPUT_DIR" ]; then
  mkdir -p "$OUTPUT_DIR"
fi

ACCELERATE_CPU_AFFINITY=1 torchrun --nproc_per_node="${NUM_GPUS}" --nnodes="${NNODES}" --node_rank="${RANK}" --master_addr="${ADDR}" --master_port="${PORT}" \
    llava/train/train_mem.py \
    --deepspeed scripts/zero2.json \
    --model_name_or_path ${CKPT_PATH} \
    --version ${PROMPT_VERSION} \
    --data_path path/to/vcg-plus_112K.json \
    --video_folder path/to/Activity_Videos \
    --mm_tunable_parts="mm_vision_tower,mm_mlp_adapter,mm_language_model" \
    --vision_tower ${VISION_MODEL_VERSION} \
    --mm_vision_tower_lr=2e-6 \
    --mm_projector_type mlp2x_gelu \
    --mm_vision_select_layer -2 \
    --mm_use_im_start_end False \
    --mm_use_im_patch_token False \
    --group_by_modality_length True \
    --image_aspect_ratio anyres_max_9 \
    --image_grid_pinpoints  "(1x1),...,(6x6)" \
    --mm_patch_merge_type spatial_unpad \
    --bf16 True \
    --run_name $MID_RUN_NAME \
    --output_dir "./checkpoints/${MID_RUN_NAME}" \
    --num_train_epochs 1 \
    --per_device_train_batch_size 1 \
    --per_device_eval_batch_size 4 \
    --gradient_accumulation_steps 2 \
    --evaluation_strategy "no" \
    --save_strategy "no" \
    --learning_rate 1e-5 \
    --weight_decay 0. \
    --warmup_ratio 0.03 \
    --lr_scheduler_type "cosine" \
    --logging_steps 1 \
    --tf32 False \
    --model_max_length 32768 \
    --gradient_checkpointing True \
    --dataloader_num_workers 4 \
    --lazy_preprocess True \
    --torch_compile True \
    --torch_compile_backend "inductor" \
    --dataloader_drop_last True \
    --frames_upbound 32 \
    --attn_implementation eager
```

参数说明：

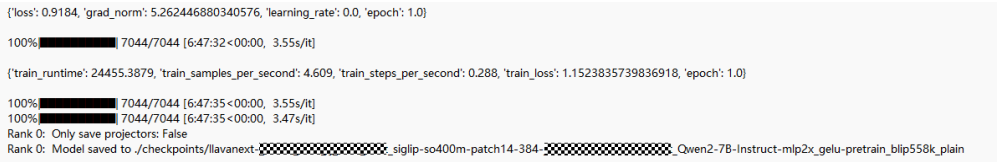
- NODE_NUM：机器数量。
- NODE_RANK：机器rank num，主机为0，其余递增。
- MASTER_ADDR：主机IP地址。

训练成功如下图所示。

图 5-41 pretrain_clip 训练成功

```
{'loss': 2.3558, 'grad_norm': 0.2941415476593059, 'learning_rate': 0.0, 'epoch': 1.0}

100%|[redacted]| 4361/4361 [3:12:57<00:00,  2.82s/it]

{'train_runtime': 11577.7033, 'train_samples_per_second': 48.207, 'train_steps_per_second': 0.377, 'train_loss': 2.511206555765612, 'epoch': 1.0}

100%|[redacted]| 4361/4361 [3:12:57<00:00,  2.82s/it]
100%|[redacted]| 4361/4361 [3:12:57<00:00,  2.65s/it]
Rank 0:  Only save projectors: True
Rank 0:  Model saved to ./checkpoints/projectors/llavanext-[redacted]_clip-vit-large-patch14-336-[redacted]_Qwen2-7B-Instruct-mlp2x_gelu-pretrain_blip558k_plain
```

图 5-42 finetune_onevision 训练成功



5.9 LLaVA 模型基于 Lite Server 适配 PyTorch NPU 预训练指导（6.3.912）

LLaVA是一种新颖的端到端训练的大型多模态模型，它结合了视觉编码器和Vicuna，用于通用的视觉和语言理解，实现了令人印象深刻的聊天能力，在科学问答（Science QA）上达到了新的高度。

本文档主要介绍如何利用ModelArts Lite Server，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成LLaVA模型训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。训练至少需要单机8卡，推理需要单机单卡。

表 5-27 环境要求

名称	版本
CANN	cann_8.0.rc3
驱动	24.1.rc1
PyTorch	2.1.0

获取软件和镜像

表 5-28 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表5-28**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练至少需要单机8卡。
- 确保容器可以访问公网。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**表5-28**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练默认使用单机8卡。

   ```
   docker run -itd --net=host \
   --device=/dev/davinci0 \
   ```

```
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

步骤三：获取代码并上传

上传代码AscendCloud-AIGC-6.3.912-xxx.zip到容器的工作目录中，包获取路径请参见**表5-28**。

上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

Step4 准备训练环境

1. 获取LLaVA模型代码。

```
cd ${container_work_dir}
unzip AscendCloud-6.3.912-xxx.zip
```



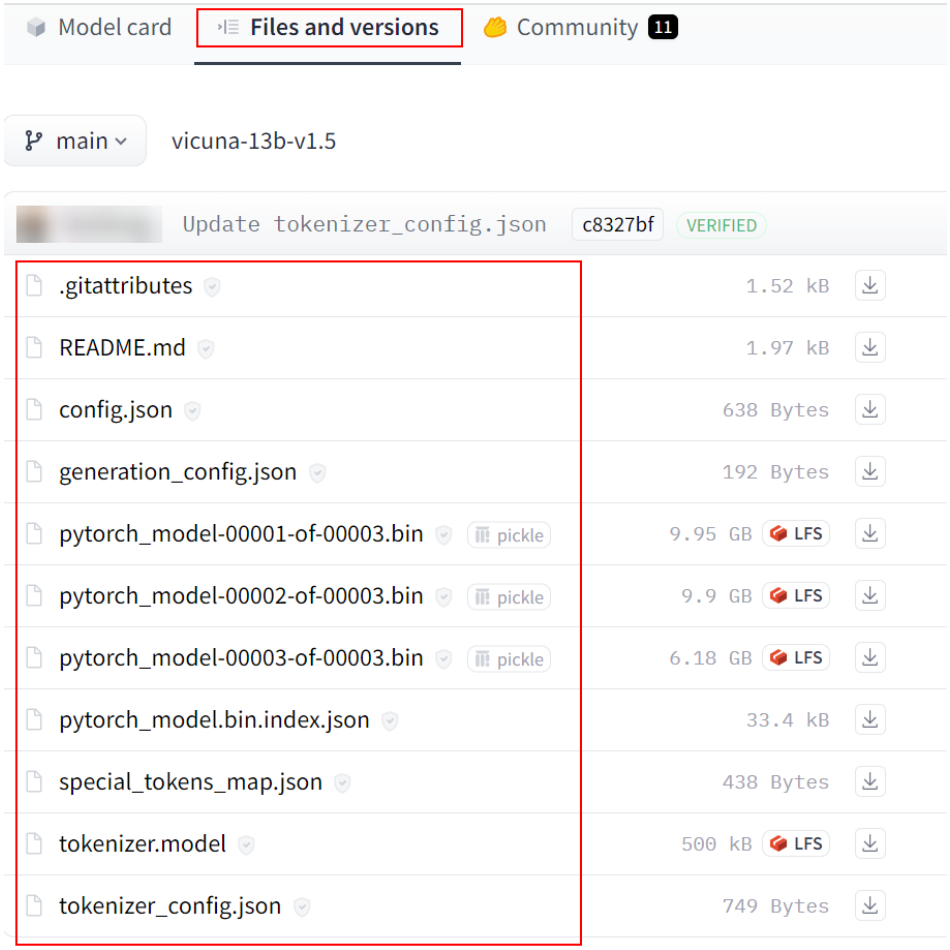
```
unzip AscendCloud-AIGC-6.3.912-xxx.zip
bash multimodal_algorithm/LLAVA/llava-train/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/build.sh
```

2. 安装优化插件

```
cd multimodal_algorithm/ascendcloud_multimodal_plugin
pip install -e .
```

3. 下载vicuna-13b-v1.5模型。下载地址：**lmsys/vicuna-13b-v1.5 · Hugging Face**

图 5-43 下载 vicuna-13b-v1.5 模型



步骤四：下载数据集

请用户自行下载GQA数据集，下载地址：**images**。

将GQA数据集放于${container_work_dir}/LLaVA/playground/data/LLaVA-Pretrain目录下。

下载**blip_laion_cc_sbu_558k.json**文件，并放于${container_work_dir}/LLaVA/playground/data/LLaVA-Pretrain目录下。

步骤五：开始训练

进入解压后的源码包根目录。

```
cd ${container_work_dir}/LLaVA
```

修改训练脚本模型路径(--model_name_or_path 模型路径)。

```
vim ./scripts/v1_5/pretrain_new.sh
```

运行训练脚本，默认是单机8卡。

```
bash ./scripts/v1_5/pretrain_new.sh
```

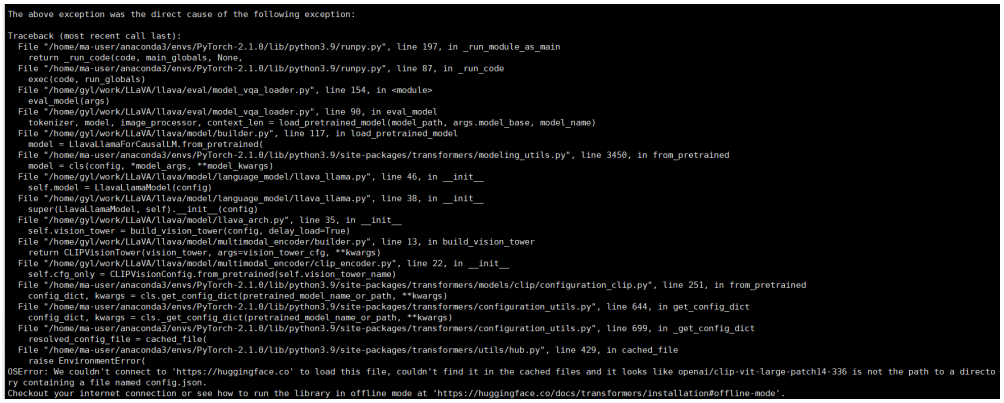
训练完成后，权重文件保存checkpoints/llava-v1.5-13b-pretrain路径下，并输出模型训练精度和性能信息。

FAQ

如果clip-vit-large-patch14-336模型不能自动下载。

请手动下载（**openai/clip-vit-large-patch14-336 at main (huggingface.co)**），并在pretrain_new.sh脚本中修改--vision_tower参数。

图 5-44 提示 clip-vit-large-patch14-336 模型不能自动下载



5.10 LLaVA 模型基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.906）

LLaVA是一种新颖的端到端训练的大型多模态模型，它结合了视觉编码器和Vicuna，用于通用的视觉和语言理解，实现了令人印象深刻的聊天能力，在科学问答（Science QA）上达到了新的高度。

本文档主要介绍如何利用ModelArts Lite Server，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成LLaVA模型推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。推理需要单机单卡。

表 5-29 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

名称	版本
驱动	23.0.5

获取软件和镜像

表 5-30 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取软件和镜像**。

```
docker pull {image_url}
```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
docker run -it --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step3 获取代码并上传

上传代码AscendCloud-AIGC-6.3.906-xxx.zip到容器的工作目录中，包获取路径请参见**获取软件和镜像**。

上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

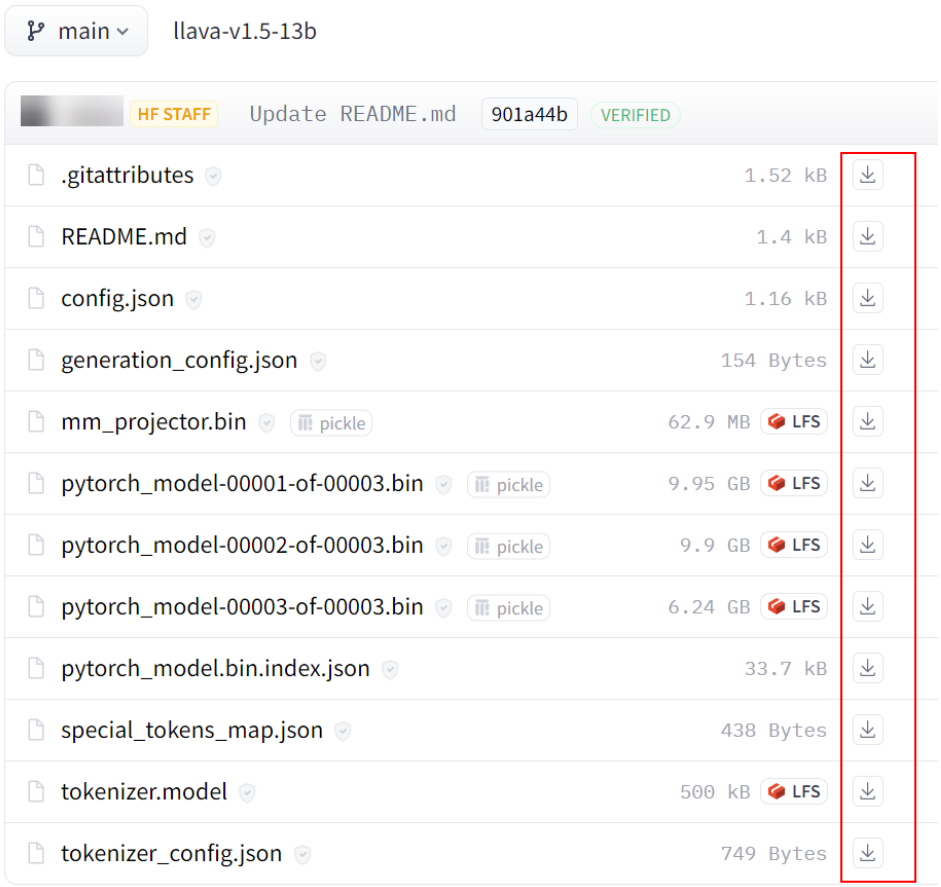
Step4 准备推理环境

1. 获取LLaVA模型代码。

```
cd ${container_work_dir}
unzip AscendCloud-6.3.906-xxx.zip
unzip AscendCloud-AIGC-6.3.906-xxx.zip
cd multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd
bash build.sh
cd LLaVA
mkdir ./playground/data/eval
```

2. 下载llava-v1.5-13b模型。下载地址：**liuhaotian/llava-v1.5-13b at main (huggingface.co)**

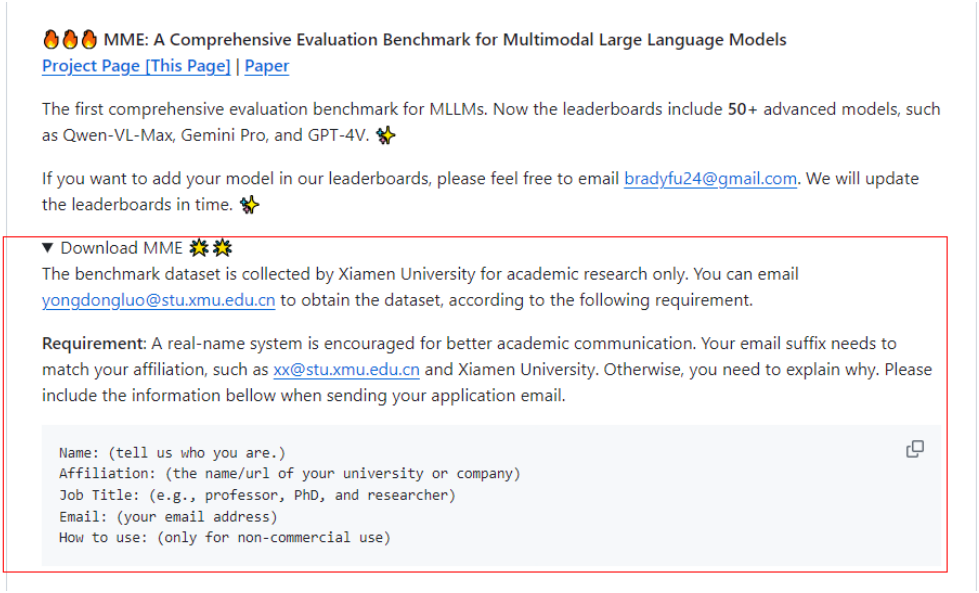
图 5-45 下载 llava-v1.5-13b 模型



Step5 下载数据集

请用户自行获取**MME评估集**，将MME评估集放于${container_work_dir}/multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA/playground/data/eval目录下。

图 5-46 MME 评估集



Step6 开始推理

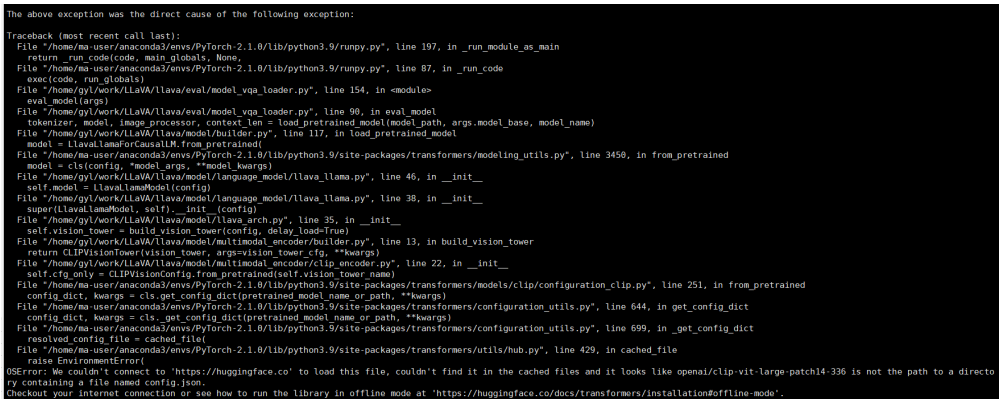
1. 进入解压后的源码包根目录。
   ```
   cd ${container_work_dir}/multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA
   ```
2. 修改mme_8p.sh。需要将脚本里模型的路径更改为实际存放模型的路径(--model-path 模型路径)，同时检查数据集路径与实际保持一致(--question-file --image-folder --answers-file)。
   ```
   vim ./scripts/v1_5/eval/mme_8p.sh
   ```
3. 运行评估脚本。启动单卡。
   ```
   ASCEND_RT_VISIBLE_DEVICES=0 bash ./scripts/v1_5/eval/mme_8p.sh
   ```
4. 启动8卡。可支持单机八卡推理，可以减短耗时。
   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 bash ./scripts/v1_5/eval/mme_8p.sh
   ```

FAQ

如果clip-vit-large-patch14-336模型不能自动下载。

请手动下载（**openai/clip-vit-large-patch14-336 at main (huggingface.co)**），并在llava-v1.5-13b模型下的config.json文件中修改mm_vision_tower参数中的模型路径。

图 5-47 提示 clip-vit-large-patch14-336 模型不能自动下载



5.11 Llama 3.2-Vision 基于 Lite Server 适配 Pytorch NPU 训练微调指导（6.3.912）

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展Llama 3.2-Vision-11B模型的训练过程，包括finetune full训练和LoRA训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.912版本，请参考获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite Server。
- 镜像适配的Cann版本是cann_8.0.rc3。
- 驱动版本：23.0.6。
- PyTorch版本：2.1.0。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。

获取软件和镜像

表 5-31 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表5-31**。

```
docker pull {image_url}
```

步骤三：启动容器镜像

启动容器镜像，启动前可以根据实际需要增加修改参数。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

- --device=/dev/davinciX：挂载NPU设备，示例中挂载了8张卡
- work_dir：工作目录，目录下存放着训练所需代码、数据等文件
- container_work_dir：容器工作目录，一般同work_dir
- container_name：自定义容器名
- image_id：镜像ID，通过docker images来查看拉取的镜像ID。

步骤四：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it  ${container_name} bash
```

修改权限。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
```

此步骤可能需要密码或root权限。

步骤五：下载代码安装环境

下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，获取路径参见**表1 获取软件和镜像**。

```
mv AscendCloud-AIGC-6.3.912-*.zip ${container_work_dir}
cd ${container_work_dir}
unzip AscendCloud-AIGC-6.3.912-*.zip
cp -r multimodal_algorithm/Llama32_Vision/train/6b33108084eb4a8efd5d09090a3e1a51f6920129/* ${container_work_dir}
# 下载复制后${container_work_dir}目录下存在如下5个文件
# 1.llama32_vision_install.sh
```

```
# 2.custom_dataset_info_demo.json
# 3.llama32_vision.patch
# 4.llama32_vision_11b_finetune_sft.sh
# 5.llama32_vision_11b_finetune_lora.sh
```

安装代码&环境

```
bash llama32_vision_install.sh
```

步骤六：增加适配代码

```
# 安装优化加速包
cd ${container_work_dir}/multimodal_algorithm/ascendcloud_multimodal_plugin
pip install -e .

# 使能优化加速包(此步默认在环境安装阶段已完成，可忽略)
# 在${container_work_dir}/ms-swift/swift/cli/sft.py中引入优化代码包 from
ascendcloud_multimodal.train.models.llama32_vision import ascend_modeling_mllama
```

步骤七：数据集下载与制作

下载COCO2014数据集：**train2014.zip**，**coco_karpathy_train_567k.zip**。本节展示了基于COCO2014数据集制作一个演示的demo数据集，若用户有自定义数据集需求，可按以下叙述的数据集格式构建用户自定义数据集。

在${container_work_dir}目录下载以上数据集，并将其放置在${container_work_dir}路径下；

```
cd ${container_work_dir}
wget http://images.cocodataset.org/zips/train2014.zip && unzip train2014.zip
wget https://github.com/OpenGVLab/InternVL/releases/download/data/coco_karpathy_train_567k.zip &&
unzip coco_karpathy_train_567k.zip
```

下载COCO2014数据集后目录结构如下：

```
${container_work_dir}
├── train2014
│   ├── COCO_train2014_000000270070.jpg
│   ├── COCO_train2014_000000101772.jpg
│   ├── COCO_train2014_000000359320.jpg
│   ...
│   ├── COCO_train2014_000000177349.jpg
│   ├── COCO_train2014_000000283267.jpg
│   └── COCO_train2014_000000334463.jpg
└── coco_karpathy_train_567k.jsonl
```

对于coco_karpathy_train_567k.jsonl文件进行过滤，过滤出train2014目录下图片对应的数据，并按如下格式重新构建json文件（coco2014_train.json）：

```
[
 {
  "id": "0",
  "image": "train2014/COCO_train2014_000000057870.jpg",
  "conversations": [
   {
    "role": "user",
    "content": "<image>\nProvide a one-sentence caption for the provided image."
   },
   {
    "role": "assistant",
    "content": "A restaurant has modern wooden tables and chairs."
   }
  ]
 },
 {
  "id": "1",
  "image": "train2014/COCO_train2014_000000057870.jpg",
```

```
  "conversations": [
    {
      "role": "user",
      "content": "<image>\nProvide a one-sentence caption for the provided image."
    },
    {
      "role": "assistant",
      "content": "A long restaurant table with rattan rounded back chairs."
    }
  ]
 },
...
...
...
 {
  "id": "413961",
  "image": "train2014/COCO_train2014_000000475546.jpg",
  "conversations": [
    {
      "role": "user",
      "content": "<image>\nProvide a one-sentence caption for the provided image."
    },
    {
      "role": "assistant",
      "content": "Group of people drinking wine at a public location."
    }
  ]
 },
 {
  "id": "413962",
  "image": "train2014/COCO_train2014_000000475546.jpg",
  "conversations": [
    {
      "role": "user",
      "content": "<image>\nProvide a one-sentence caption for the provided image."
    },
    {
      "role": "assistant",
      "content": "Three women and a man are sitting at a bar"
    }
  ]
 }
]
```

- 由于coco2014_train.json数据量过大，为方便适配，本指导中使用的是：取coco2014_train.json文件前4万条数据作为后续使用的demo数据集（命名为：coco2014_train_filter_40k.json）；
- 启动训练脚本前，需检查并保证下载的数据集train2014文件夹与训练脚本均在工作目录${container_work_dir}下。

demo数据集配置指导如下：

修改custom_dataset_info_demo.json文件中dataset_path(${container_work_dir}/ms-swift/swift/llm/data目录下)参数为以上步骤制作的json文件（coco2014_train_filter_40k.json）的路径；custom_dataset_info_demo.json文件如下。

```
{
  "coco2014_train_40k_demo": {
     "dataset_path": "path/to/coco2014_train_filter_40k.json",
     "image": "image",
     "conversations": {
       "user_role": "user",
       "assistant_role": "assistant",
       "from_key": "role",
       "value_key": "content",
       "error_strategy": "delete",
```

```
        "media_type": "image",
        "media_key": "image"
      }
    }
}
```

步骤八：下载模型权重

从modelscope下载**Llama-3.2-11B-Vision-Instruct**或将您已下载的权重文件上传到容器能正常访问的目录。

方式1：手动下载，并将其放置在${container_work_dir}路径下。

方式2：利用git下载，须确保git lfs已成功安装：

```
cd ${container_work_dir}
git clone https://www.modelscope.cn/LLM-Research/Llama-3.2-11B-Vision-Instruct.git
cd Llama-3.2-11B-Vision-Instruct
git lfs pull
```

步骤九：开始训练

单机训练

```
cd ${container_work_dir}
# 指定model_path为步骤八：下载模型权重下载的Llama-3.2-11B-Vision-Instruct权重路径
# 指定dataset参数为步骤七：数据集下载与制作中所述custom_dataset_info_demo.json中文件设置的数据集名称：coco2014_train_40k_demo
# 修改custom_dataset_info参数路径为${container_work_dir}/ms-swift/swift/llm/data/custom_dataset_info_demo.json
# finetune sft全参微调场景
model_path=path/to/Llama-3.2-11B-Vision-Instruct \
dataset=coco2014_train_40k_demo \
custom_dataset_info=path/to/ms-swift/swift/llm/data/custom_dataset_info_demo.json \
work_dir=${container_work_dir} \
globol_batch_size=32 \
per_device_batch_size=2 \
bash llama32_vision_11b_finetune_sft.sh
# finetune lora微调场景
model_path=path/to/Llama-3.2-11B-Vision-Instruct \
dataset=coco2014_train_40k_demo \
custom_dataset_info=path/to/ms-swift/swift/llm/data/custom_dataset_info_demo.json \
work_dir=${container_work_dir} \
globol_batch_size=32 \
per_device_batch_size=4 \
bash llama32_vision_11b_finetune_lora.sh
```

多机训练

```
cd ${container_work_dir}
# 指定model_path为步骤八：下载模型权重下载的Llama-3.2-11B-Vision-Instruct权重路径
# 指定dataset参数为步骤七：数据集下载与制作中所述custom_dataset_info_demo.json中文件设置的数据集名称：coco2014_train_40k_demo
# 指定custom_dataset_info参数路径为${container_work_dir}/ms-swift/swift/llm/data/custom_dataset_info_demo.json
# finetune sft场景
model_path=path/to/Llama-3.2-11B-Vision-Instruct \
dataset=coco2014_train_40k_demo \
custom_dataset_info=path/to/ms-swift/swift/llm/data/custom_dataset_info_demo.json \
work_dir=${container_work_dir} \
node_num=${NODE_NUM} \
node_rank=${NODE_RANK} \
master_addr=${MASTER_ADDR} \
globol_batch_size=32*${NODE_NUM} \
per_device_batch_size=2 \
bash llama32_vision_11b_finetune_sft.sh
# finetune lora场景
```

```
model_path=path/to/Llama-3.2-11B-Vision-Instruct \
dataset=coco2014_train_40k_demo \
custom_dataset_info=path/to/ms-swift/swift/llm/data/custom_dataset_info_demo.json \
work_dir=${container_work_dir} \
node_num=${NODE_NUM} \
node_rank=${NODE_RANK} \
master_addr=${MASTER_ADDR} \
global_batch_size=32*${NODE_NUM} \
per_device_batch_size=4 \
bash llama32_vision_11b_finetune_lora.sh
```

参数说明：

- node_num/NODE_NUM：机器数量，修改${NODE_NUM}为具体数字。
- node_rank/NODE_RANK：机器rank num，主机为0，其余递增，修改${NODE_RANK}为具体数字。
- master_addr/MASTER_ADDR：主机IP地址，修改${MASTER_ADDR}为主机IP。
- globol_batch_size：全局批次大小。
- per_device_batch_size：每张卡上的批次大小。

以单机结果为例，训练成功如下图所示。

图 5-48 sft 全参微调训练成功

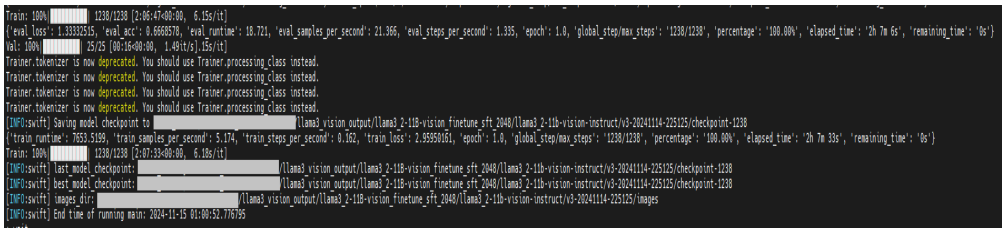
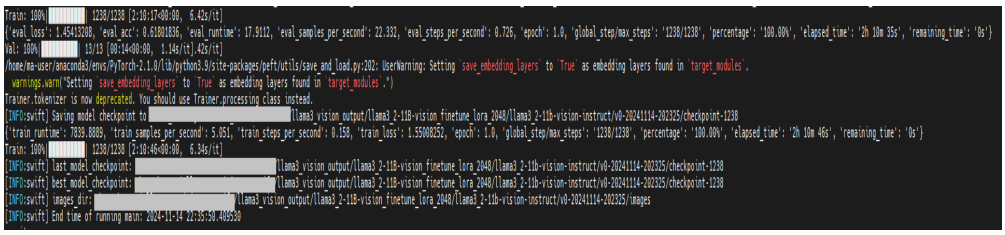


图 5-49 lora 微调训练成功



附：loss 曲线

loss结果

- sft全参微调NPU训练结果loss收敛且趋势与GPU训练loss一致

图 5-50 sft 全参微调单机 loss 曲线对比结果

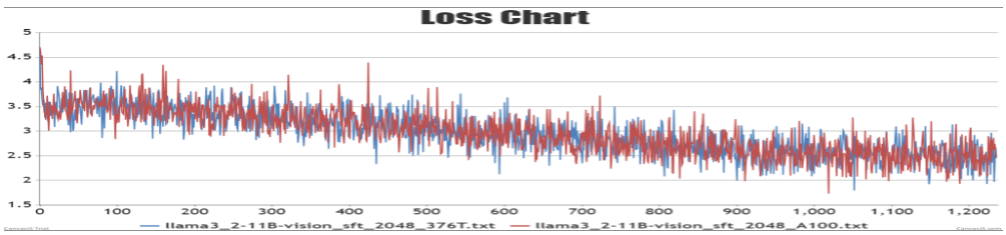
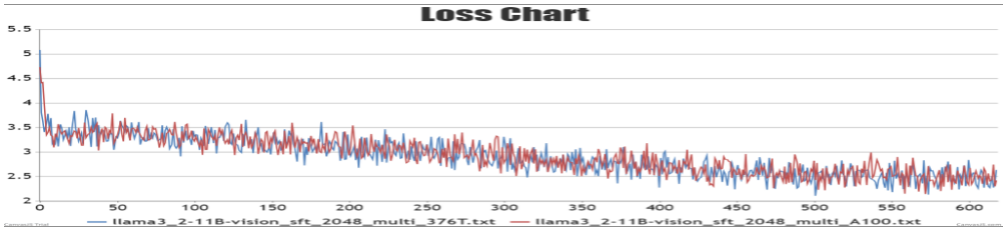


图 5-51 sft 全参微调双机 loss 曲线对比结果



- lora微调NPU训练结果loss收敛且趋势与GPU训练loss一致

图 5-52 lora 微调双机 loss 曲线对比结果

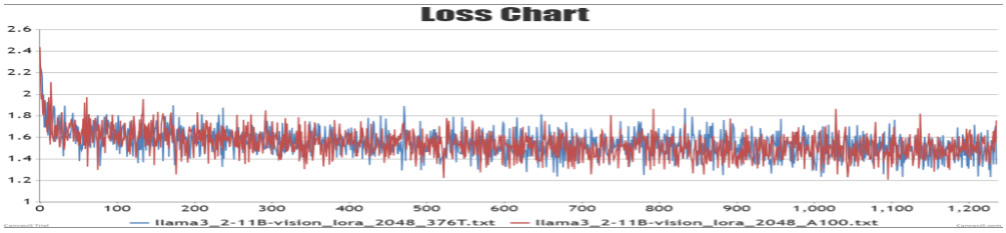
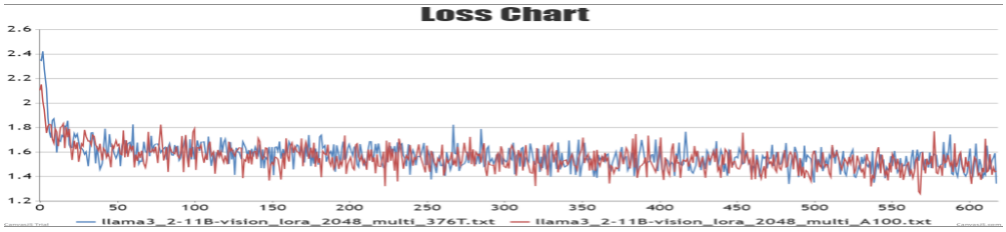


图 5-53 lora 微调双机 loss 曲线对比结果



5.12 LLaMA-VID 基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.910）

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展LLaMA-VID的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.910版本，请参考获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

获取软件和镜像

表 5-32 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.910软件包中的AscendCloud-AIGC-6.3.910-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	SWR上拉取。

表 5-33 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

步骤一 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表5-32**。

```
docker pull {image_url}
```

步骤三 启动容器镜像

启动容器镜像，启动前可以根据实际需要增加修改参数。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \${image_id}  \
/bin/bash
```

- --device=/dev/davinciX 挂载NPU设备，示例中挂载了1张卡
- work_dir：工作目录，目录下存放着训练所需代码、数据等文件
- container_work_dir: 容器工作目录，一般同work_dir
- container_name：自定义容器名
- image_id：镜像ID，通过docker images来查看拉取的镜像ID。

步骤四 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it  ${container_name} bash
```

修改权限。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
```

此步骤可能需要密码或root权限。

步骤五 下载代码及安装环境

下载华为侧插件代码包AscendCloud-AIGC-6.3.910-xxx.zip文件，获取路径参见**表5-32**。

```
unzip AscendCloud-AIGC-6.3.910-xxx.zip
解压后，进到指定目录：
cd multimodal_algorithm/LLaMA-VID/
执行安装脚本：
bash llama_vid_install.sh
```

步骤六 下载模型参数

```
从链接https://huggingface.co/YanweiLi/llama-vid-7b-full-224-video-fps-1中，下载模型参数，并根据以下目录结构存放
LLaMA-VID
├── llamavid
├── scripts
├── work_dirs
│   ├── llama-vid
│   │   ├── llama-vid-7b-full-224-video-fps-1
│   │   ├── ...
```

步骤七 下载 model_zoo 相关数据

从以下5个链接下载model_zoo数据

```
https://huggingface.co/lmsys/vicuna-7b-v1.5
https://huggingface.co/lmsys/vicuna-13b-v1.5
https://storage.googleapis.com/sfr-vision-language-research/LAVIS/models/BLIP2/eva_vit_g.pth
https://storage.googleapis.com/sfr-vision-language-research/LAVIS/models/InstructBLIP/
instruct_blip_vicuna7b_trimmed.pth
https://storage.googleapis.com/sfr-vision-language-research/LAVIS/models/InstructBLIP/
instruct_blip_vicuna13b_trimmed.pth
```

在根目录LLaMA-VID下创建model_zoo路径，下载的文件根据以下目录结构进行存放

```
LLaMA-VID
├── llamavid
├── scripts
├── work_dirs
├── model_zoo
│   ├── LLM
│   │   ├── vicuna
│   │   │   ├── 7B-V1.5
│   │   │   ├── 13B-V1.5
│   ├── LAVIS
│   │   ├── eva_vit_g.pth
│   │   ├── instruct_blip_vicuna7b_trimmed.pth
│   │   ├── instruct_blip_vicuna13b_trimmed.pth
```

步骤 8 下载 MSVD_QA 数据集

MSVD_QA数据集下载路径：https://mycuhk-my.sharepoint.com/:u:/g/personal/1155186668_link_cuhk_edu_hk/EUNEXqg8pctPq3WZPHb4Fd8BYIxHO5qPCnU6aWsrV1O4JQ?e=guynwu

解压后，存放的目录结构如下：

```
LLaMA-VID
├── data
│   ├── LLaMA-VID-Eval
│   │   ├── MSVD-QA
```

步骤 9 启动一级流水优化

```
export TASK_QUEUE_ENABLE=2
```

步骤 10 修改 msvd_eval.sh 参数

修改scripts/video/eval/msvd_eval.sh中的参数

1. 模型存放的地方，如果根据第2步的方式保存的模型，设置如下：
   ```
   CKPT="llama-vid/llama-vid-7b-full-224-video-fps-1"
   ```
2. 调用openai的key，评估精度时需要调用openai，需要填写正确的key，这个可能需要进行付费调用，评估1000条大概需要0.15美元
   ```
   OPENAIKEY=""
   ```
   注：openai不支持中国大陆和香港，不能使用中国的代理。
3. 推理结果保存的文件名，可不用修改，使用默认的文件名
   ```
   OUTPUTNAME=pred
   ```

步骤 11 执行推理脚本

```
bash scripts/video/eval/msvd_eval.sh
```

步骤 12 查看推理结果

在./work_dirs/eval_msvd/$CKPT下，存在两个文件，可以查看推理的精度和性能结果

1. 精度结果文件：accuracy_${time_stamp}.json
2. 性能结果文件：performance_${time_stamp}.json

图 5-54 精度结果

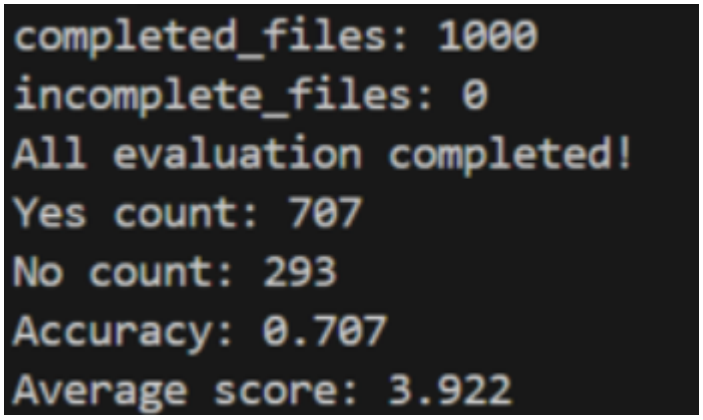
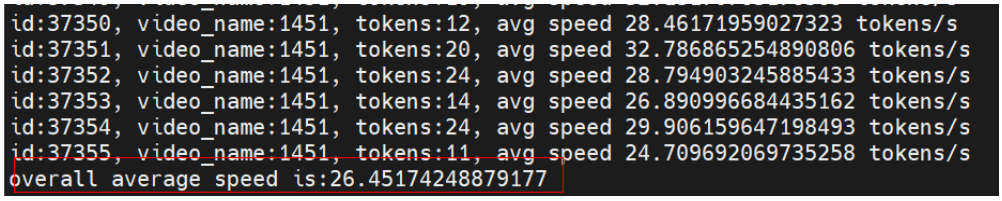


图 5-55 性能结果



5.13 moondream2 基于 Lite Server 适配 PyTorch NPU 推理指导

方案概览

本文档从模型部署的环境配置、模型转换、模型推理等方面进行介绍moondream2模型在ModelArts Lite Server上部署，支持NPU推理场景。

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

资源规格要求

推理部署推荐使用Lite Server资源和Ascend Snt9B单机单卡。

表 5-34 环境要求

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取镜像

表 5-35 获取镜像

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	从SWR拉取。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. 检查环境。

   a. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   b. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   c. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-35**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -itd \
      --device=/dev/davinci1 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 32g \
      --net=bridge \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。
```
docker exec -it ${container_name} bash
```

Step4 下载原始模型包

从HuggingFace官网下载moondream2模型包到本地，下载地址：**https://huggingface.co/vikhyatk/moondream2/tree/2024-03-06**。

在宿主机上创建一个空目录/home/temp，将下载的模型包存放在宿主机/home/temp/moondream2目录下，修改目录权限后，复制到容器中。

```
mkdir /home/temp       #创建一个空目录，将下载的模型包存放在宿主机/home/temp/moondream2目录下
chmod -R 777 moondream2     #修改moondream2目录权限
docker cp moondream2 moondream2:/home/ma-user/     #复制moondream2目录到容器中
```

Step5 准备测试数据

需要用户自己准备测试图片。

将测试图片存放在宿主机/home/temp/data目录下，修改目录权限后，复制到容器中。

```
chmod -R 777 data    #修改data目录权限
docker cp data moondream2:/home/ma-user/   #复制data目录到容器中
```

Step6 安装依赖

执行如下命令安装推理依赖。
```
pip install transformers timm einops torch==2.1.0 &&
pip install --upgrade sympy
```

Step7 启动推理

在容器/home/ma-user下运行启动推理脚本infer.py，NPU推理脚本内容参见**附录1：在NPU上运行infer.py脚本内容**。

```
python infer.py
```

运行结束后，会打印所有图片预测的平均时延。

NPU上运行后，结果会保存在/home/ma-user/result.txt下。

如果在GPU上运行，推荐直接在GPU宿主机上执行，因此不需要启动容器，直接将模型和数据复制到相应目录，然后安装PIP依赖后就可以运行。GPU推理脚本内容参见**附录2：在GPU上运行infer.py脚本内容**。

附录 1：在 NPU 上运行 infer.py 脚本内容

NPU上运行推理的infer.py脚本内容如下：

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from PIL import Image
import torch
import os
import time

import torch_npu
#from torch_npu.contrib import transfer_to_npu

import torchair as tng
from torchair.configs.compiler_config import CompilerConfig
```

```
#import logging
#from torchair.core.utils import logger
# 是否开启DEBUG日志
# logger.setLevel(logging.DEBUG)

model_id = "./moondream2"
revision = "2024-03-13"
model = AutoModelForCausalLM.from_pretrained(
    model_id, trust_remote_code=True, revision=revision
)

device = 'npu:0'
model = model.to(device)
tokenizer = AutoTokenizer.from_pretrained(model_id, revision=revision)


config = CompilerConfig()
npu_backend = tng.get_npu_backend(compiler_config=config)
model.text_model.transformer = torch.compile(model.text_model.transformer, backend=npu_backend,
dynamic=True, fullgraph=True)

filenames = os.listdir(r'./data')
filenames = sorted(filenames)
count = 0
total_time = 0.0
not_num = 1
with open("./result.txt", 'w+') as f:
  for file in filenames:
    t1 = time.time()
    image = Image.open('./data/'+file)
    enc_image = model.encode_image(image)
    enc_image = enc_image.to(device)
    result = model.answer_question(enc_image, "Describe in detail what is in the video frame. The rule is:
first describe the main body of the character in the video frame, including action, state, characteristics, etc.,
do not make associations or summarize. Then describe the environment, such as the background; then
describe how the video was shot, such as close-ups. Do not appear 'seems', 'may' and other words, need to
be sure of the description, do not need to be ambiguous description.", tokenizer)
    cost = time.time()-t1
    if not_num <=0:
      count = count+1
      total_time += cost
      print("infer time:"+str(cost))
      print("average infer time:"+str(total_time/count), " total count:"+str(count))
    else:
      not_num = not_num -1
    f.write(file + ":" + "\n")
    f.write(result + "\n\n")
```

附录 2：在 GPU 上运行 infer.py 脚本内容

GPU上运行推理的infer.py脚本内容如下：

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from PIL import Image
import torch
import os
import time

model_id = "./moondream2"
revision = "2024-03-13"
model = AutoModelForCausalLM.from_pretrained(
    model_id, trust_remote_code=True, revision=revision
)

device = 'cuda:0'
model = model.to(device)
tokenizer = AutoTokenizer.from_pretrained(model_id, revision=revision)

filenames = os.listdir(r'./data')
```

```
filenames = sorted(filenames)
count = 0
total_time = 0.0
not_num = 1
with open("./result.txt", 'w+') as f:
  for file in filenames:
    t1 = time.time()
    image = Image.open('./data/'+file)
    enc_image = model.encode_image(image)
    enc_image = enc_image.to(device)
    result = model.answer_question(enc_image, "Describe in detail what is in the video frame. The rule is: first describe the main body of the character in the video frame, including action, state, characteristics, etc., do not make associations or summarize. Then describe the environment, such as the background; then describe how the video was shot, such as close-ups. Do not appear 'seems', 'may' and other words, need to be sure of the description, do not need to be ambiguous description.", tokenizer)
    cost = time.time()-t1
    if not_num <=0:
      count = count+1
      total_time += cost
      print("infer time:"+str(cost))
      print("average infer time:"+str(total_time/count), " total count:"+str(count))
    else:
      not_num = not_num -1
    f.write(file + ":" + "\n")
    f.write(result + "\n\n")
```

6 文生图模型训练推理

6.1 FlUX.1 基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.912）

Flux是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。官方提供了三个版本：FLUX.1-pro、FLUX.1-dev和FLUX.1-schnell。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展Flux模型的FLUX.1-dev版本分别使用ComfyUI 0.2.2和Diffusers 0.30.2框架的推理过程。另外，FLUX.1-schnell模型的使用方法和FLUX.1-dev一致，只需替换权重文件即可，本文以FLUX.1-schn为例。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表6-1**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。

软件配套版本

表 6-1 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-2 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.912版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=60g \
   -p 8585:8585 \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。
- -p 8585:8585：映射端口号，用户可自定义未被占用的端口号。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

步骤三：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```


步骤四：下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，获取路径参见**表6-1**。将该目录上传到宿主机上的工作目录下，例如：${container_work_dir}/，然后解压到工作目录下。

步骤五：下载 ComfyUI 代码并安装依赖

1. **下载ComfyUI源码**

   从github下载ComfyUI代码并切换到0.2.2分支。

   ```
   cd ${container_work_dir}
   git clone -c http.sslVerify=false https://github.com/comfyanonymous/ComfyUI.git
   cd ComfyUI
   # 切换到comfyui 0.2.2分支
   git reset --hard 0c7c98a
   ```

2. **下载flux模型权重**

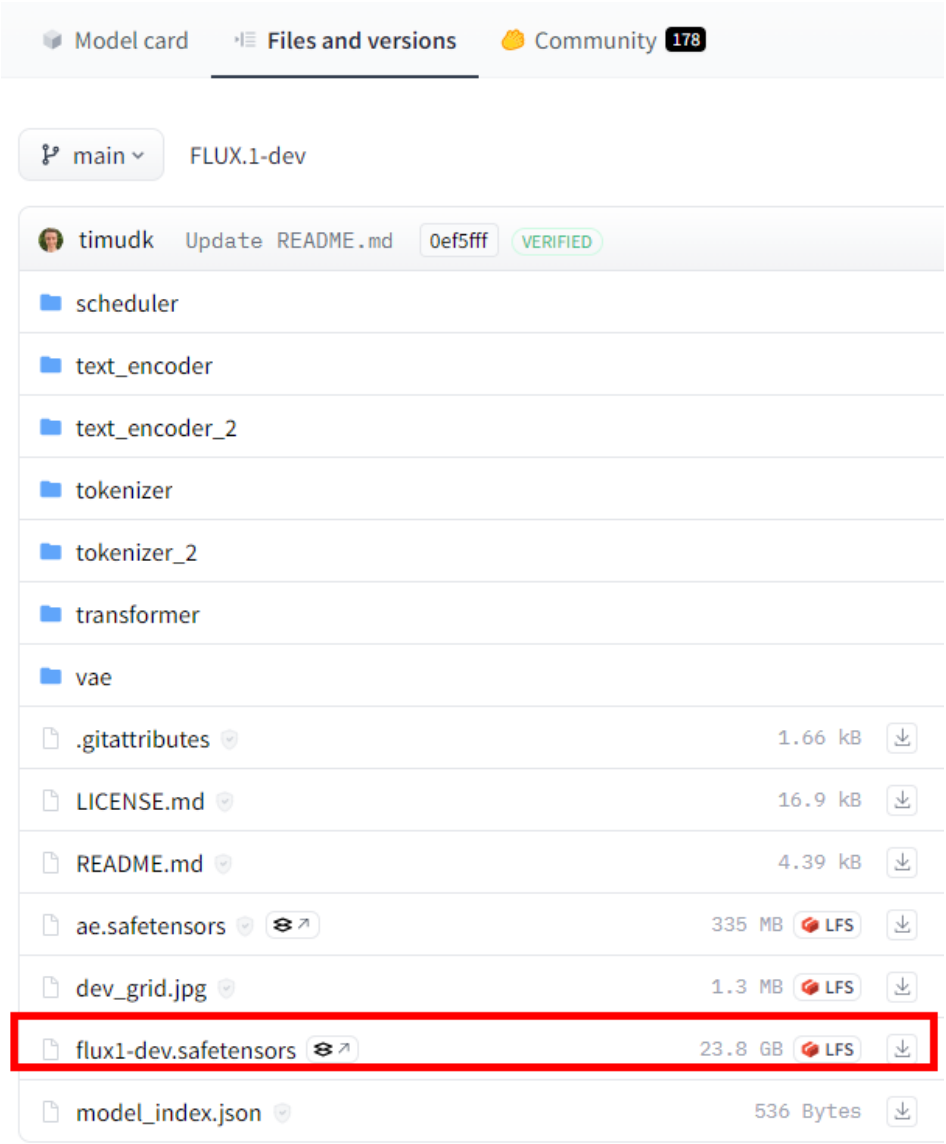
   下载模型权重文件，并将模型放到容器内自定义挂载的工作目录。

   - 下载Diffusion模型权重文件flux1-dev.safetensors，放到${container_work_dir}/ComfyUI/models/unet 目录下。其中，

     FLUX.1-dev下载链接：**https://huggingface.co/black-forest-labs/FLUX.1-dev/tree/main**

     如下图所示：

图 6-1 flux1-dev.safetensors



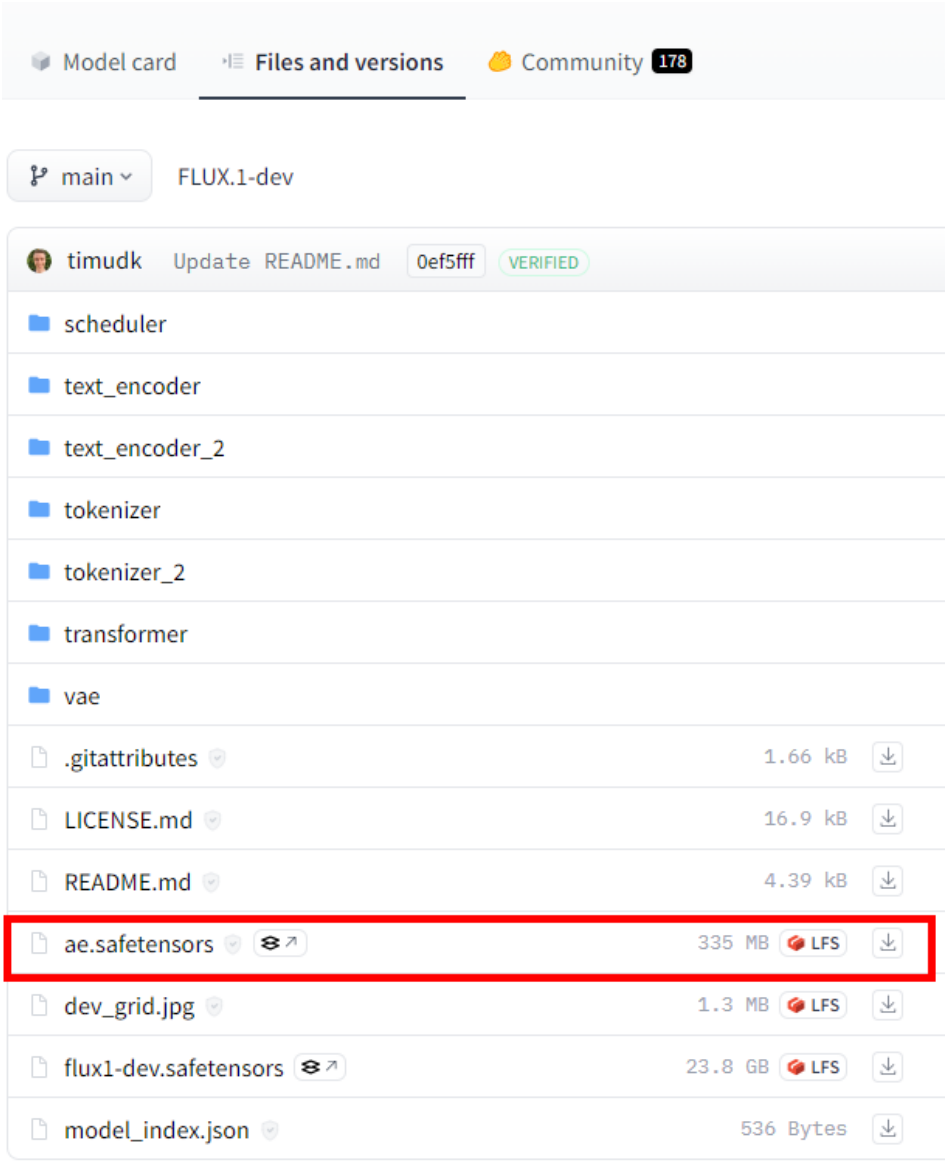
- 下载vae权重，放到 ${container_work_dir}/ComfyUI/models/vae 目录下，FLUX.1-dev和FLUX.1-schnell使用相同的vae权重。

  下载链接：

  https://huggingface.co/black-forest-labs/FLUX.1-dev/tree/main

  如下图所示：

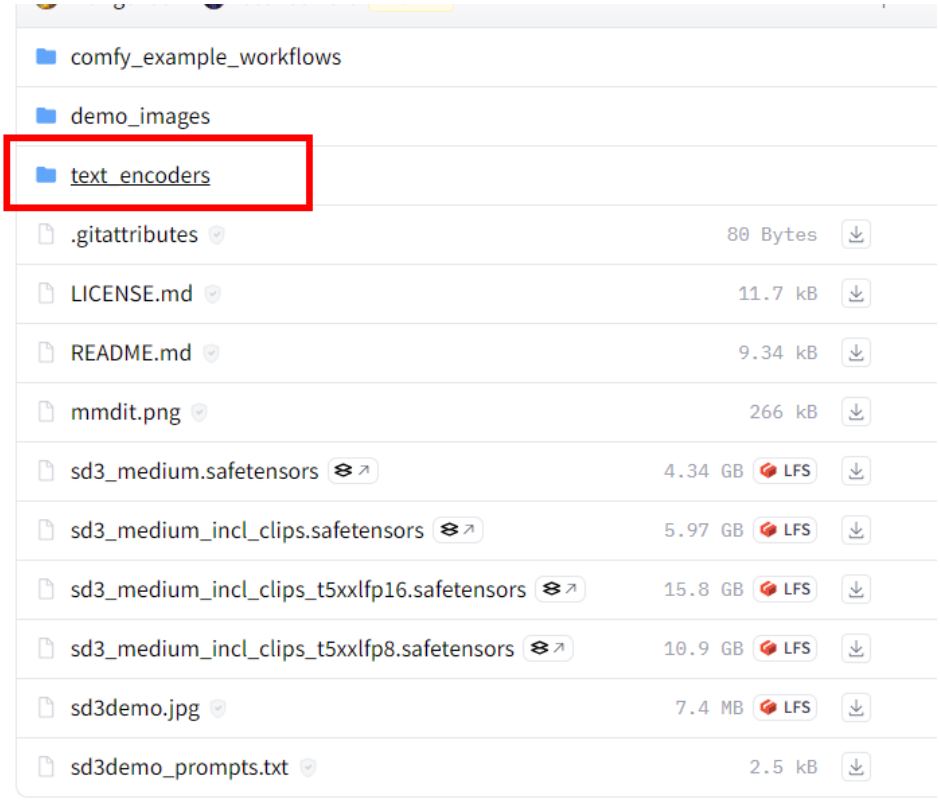
图 6-2 vae 权重



- 下载text_encoder权重文件夹，放到${container_work_dir}/ComfyUI/models/clip 目录下。

  下载链接：**https://huggingface.co/stabilityai/stable-diffusion-3-medium/tree/main**

图 6-3 text_encoder 权重文件



3. **替换Ascend_node**

   将${container_work_dir}/aigc_inference/torch_npu/comfyui/a82fae2/comfyui_ascend_node文件夹复制到${container_work_dir}/ComfyUI/custom_nodes/目录下。

4. **安装ascend_diffusers插件**

   执行以下命令安装华为侧插件ascend_diffusers。

   ```
   pip install -e ${container_work_dir}/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers
   ```

5. **安装依赖**

   运行以下命令进入工作目录，安装所需依赖包。

   ```
   cd ${container_work_dir}

   pip install transformers==4.44.2  accelerate==0.34.2  sentencepiece==0.2.0  einops==0.8.0
   torchsde==0.2.6  aiohttp==3.10.5  omegaconf==2.3.0  fastapi==0.115.0  uvicorn==0.30.6
   spandrel==0.4.0  kornia==0.7.3
   ```

6. **修改comfyui 源码**

   修改 ${container_work_dir}/ComfyUI/comfy/ldm/flux/math.py 文件中rope()方法，把linespace的dtype改成torch.float32：

```
def rope(pos: Tensor, dim: int, theta: int) -> Tensor:
    assert dim % 2 == 0
    if comfy.model_management.is_device_mps(pos.device) or comfy.model_management.is_intel_xpu():
        device = torch.device("cpu")
    else:
        device = pos.device

    ############## new code ##############
    scale = torch.linspace(0, (dim - 2) / dim, steps=dim//2, dtype=torch.float32, device=device)
    ######################################

    ############# old code ###############
    # scale = torch.linspace(0, (dim - 2) / dim, steps=dim//2, dtype=torch.float64, device=device)
    ######################################

    omega = 1.0 / (theta**scale)
    out = torch.einsum("...n,d->...nd", pos.to(dtype=torch.float32, device=device), omega)
    out = torch.stack([torch.cos(out), -torch.sin(out), torch.sin(out), torch.cos(out)], dim=-1)
    out = rearrange(out, "b n d (i j) -> b n d i j", i=2, j=2)
    return out.to(dtype=torch.float32, device=pos.device)
```

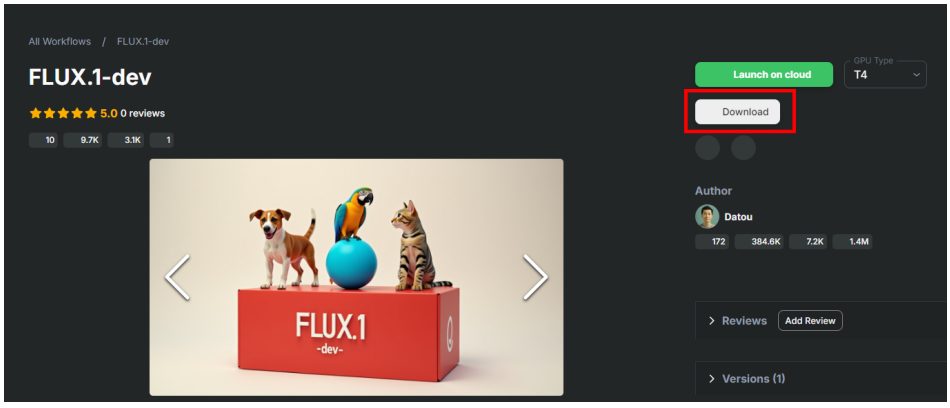
7. **下载workflow文件**

   以workflow-flux1-dev-KnSeTKHjvuTd0RiUDSmW-datou-openart.ai.json为例：

   下载链接： **https://openart.ai/workflows/datou/flux1-dev/KnSeTKHjvuTd0RiUDSmW**

   如下图所示，单击“Download”进行下载，下载的json文件放到windows机器上任意位置即可，后续在windows上启动服务后需要加载使用。

图 6-4 下载 workflow 文件



步骤六：ComfyUI 0.2.2 服务调用

1. **获取容器IP地址**

   在已启动的容器内，使用ifconfig命令获取容器IP，记为${container_ip_address}，本例中为172.17.0.7。若无效可使用ip addr，或者自行寻找其他方式获取到容器IP。

图 6-5 使用 ifconfig 命令获取容器 IP

```
(PyTorch-2.1.0) [ma-user@57a5b008b312 custom_nodes]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.7  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:07  txqueuelen 0  (Ethernet)
        RX packets 6581  bytes 5071762 (4.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2651  bytes 148084 (144.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. **使用容器IP启动服务**

```
cd ${container_work_dir}/ComfyUI

python main.py --port ${port} --force-fp16 --listen ${container_ip_address}
```

参数说明：

- port：为启动镜像时映射port
- container_ip_address：为容器IP，如上图的172.17.0.7
- 默认不使用图模式
- 若要使用图模式，需要配置环境变量 export GRAPH_MODE=1。如果使用了图模式，则首次推理时间较长，请耐心等待。

3. **浏览器启动**

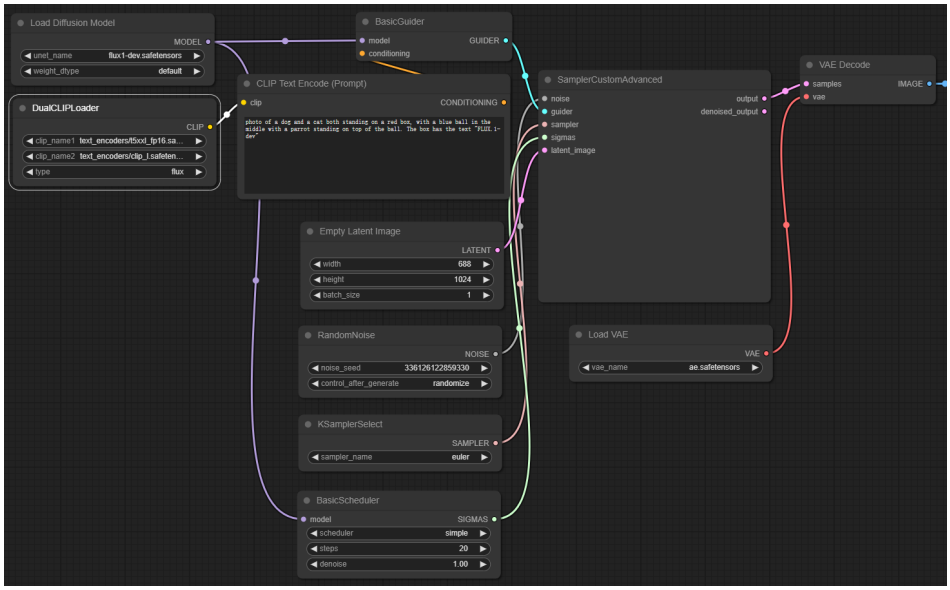
浏览器启动时，需要使用宿主机IP，在浏览器中输入 http://${host_ip_address}:${port} ,例如：**http://7.216.55.96:8585/**

参数说明：

- host_ip_address：为宿主机IP地址
- port：为启动镜像时映射port

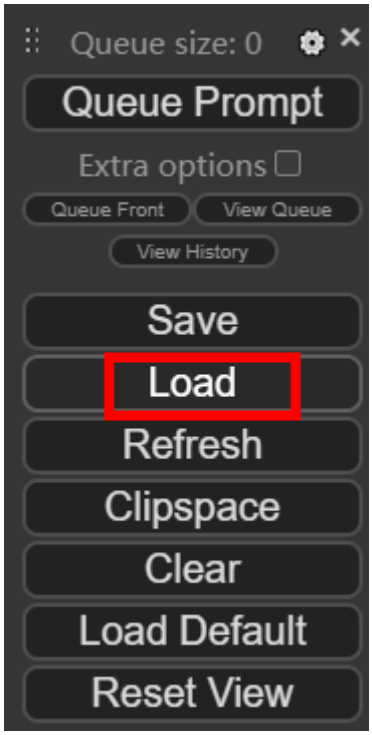
访问界面如下图。

图 6-6 访问界面



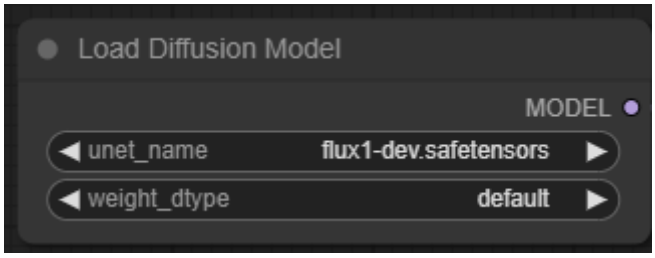
4. 加载workflow文件，选择workflow-flux1-dev-KnSeTKHjvuTd0RiUDSmW-datou-openart.ai.json。

图 6-7 加载 workflow 文件



5. 选择Diffusion model，单击选择flux1-dev.safetensors，如下图。

图 6-8 选择 flux1-dev.safetensors



6. 选择clip模型，clip_name1选择text_encoders/t5xxl_fp16.safetensors，clip_name2选择text_encoders/clip_l.safetensors, 如下图。

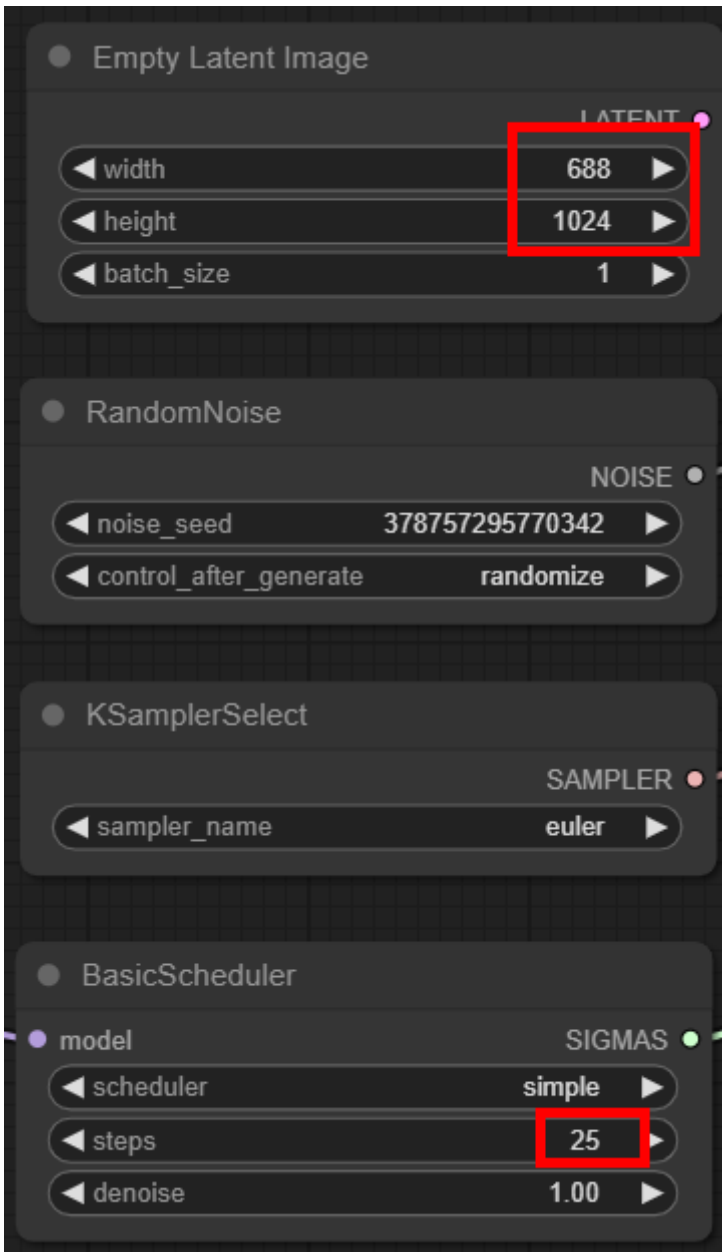
图 6-9 选择 clip 模型

7. 选择vae模型，如下图。

图 6-10 选择 vae 模型

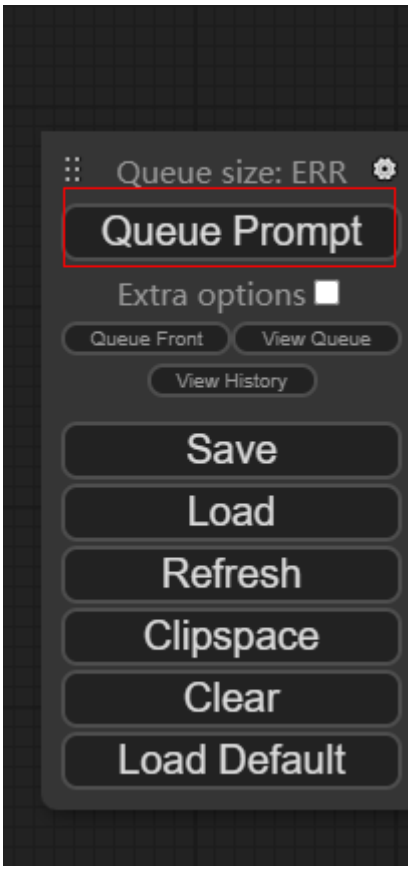
8. 配置推理的参数，如width、height、batch_size等，本文以 688*1024，25步为例，如下图所示。

图 6-11 配置推理参数



9. 单击Queue Prompt加入推理队列进行推理，如下图

图 6-12 推理队列



成功之后结果如下图所示。首次加载或切换模型推理时，需要加载模型并进行相关初始化工作，如果使用了图模式，则首次推理时间较长，请耐心等待。

图 6-13 推理成功

步骤七：Flux+Diffusers 0.30.2 适配

本章节介绍Flux模型使用Diffusers 0.30.2框架的推理过程。使用官方提供的已经训练好的模型进行推理，输入prompt生成指定像素的图片。

1. 使用如下命令登录huggingface，并输入个人账号的token，用于自动下载flux权重。
   ```
   huggingface-cli login
   ```
2. 下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，将该文件夹上传到宿主机上的工作目录下，例如 ${container_work_dir}/，并解压。
3. 安装ascend_diffusers插件
   ```
   pip install -e ${container_work_dir}/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers
   ```
4. 运行以下命令进入工作目录，安装所需依赖包。
   ```
   cd ${container_work_dir}

   pip install diffusers==0.30.2
   ```
5. 修改diffusers源码

   修改 /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/transformers/transformer_flux.py 文件中rope()方法，把scale计算中的dtype改成torch.float32。

图 6-14 修改 diffusers 源码

```
def rope(pos: torch.Tensor, dim: int, theta: int) -> torch.Tensor:
    assert dim % 2 == 0, "The dimension must be even."

    #scale = torch.arange(0, dim, 2, dtype=torch.float64, device=pos.device) / dim
    scale = torch.arange(0, dim, 2, dtype=torch.float32, device=pos.device) / dim
    omega = 1.0 / (theta**scale)

    batch_size, seq_length = pos.shape
    out = torch.einsum("...n,d->...nd", pos, omega)
    cos_out = torch.cos(out)
    sin_out = torch.sin(out)

    stacked_out = torch.stack([cos_out, -sin_out, sin_out, cos_out], dim=-1)
    out = stacked_out.view(batch_size, -1, dim // 2, 2, 2)
    return out.float()
```

6. 运行推理脚本。

```
sed -i 's/self.verify = True/self.verify = False/g' /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/requests/sessions.py

python ${container_work_dir}/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers/examples/sd_inference_example.py --flux --model_id black-forest-labs/FLUX.1-dev --prompt 'a dog' --num_inference_steps 25 --width 688  --height  1024
```

参数说明如下：

- --width ： 生成图片的宽
- --height： 生成图片的长
- --num_inference_steps：推理步数
- --dynamo: 使用图模式。如果使用该参数，则首次编译时间较长，请耐心等待。

推理完成后，生成的图片image_1024x688.png保存在当前路径下，如下图所示。

图 6-15 推理结果

```
(PyTorch-2.1.0) [root@devserver-bms-3459baaf Ascend_cloud_aigc_poc-911]# ll
total 732
-rw-------  1 root root    338 Nov 25 10:29 README.md
drwx------  4 root root     57 Nov 25 17:04 aigc_inference
drwx------  4 root root     52 Nov 25 17:04 aigc_train
drwx------  5 root root    123 Nov 25 17:04 benchmark
-rw-r-----  1 root root 738802 Nov 26 20:05 image_1024x688.png
drwx------ 16 root root   4096 Nov 25 17:04 multimodal_algorithm
```

6.2 FLUX.1 基于 DevSever 适配 PyTorch NPU Finetune&Lora 训练指导（6.3.911）

Flux是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展Flux模型的训练过程，包括基于kohya的Finetune训练和基于ai-toolkit的Lora训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**表6-3**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 6-3 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.911软件包中的AscendCloud-AIGC-6.3.911-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-4 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.911版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_3_ascend:pytorch_2.3.1-cann_8.0.rc3-py_3.10-hce_2.0.2409-aarch64-snt9b-20241114095658-d7e26d8	cann_8.0.rc3 pytorch_2.3.1 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数，Lora微调启动单卡，Fnetune全参启动八卡。
```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
docker run -itd --name ${container_name} -v ${work_dir}:${container_work_dir} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge ${image_name} bash
```

参数说明：

- ${image_name}：基础镜像地址，即表2提供的地址
- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

说明

- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

步骤三：下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.911-xxx.zip文件，获取路径参见**表6-3**。本案例使用的是解压到子目录/aigc_train/torch_npu/flux目录下的所有文件和文件夹，将flux目录下所有文件和文件夹全部上传到宿主机对应工作目录${work_dir}/flux下

步骤四：下载模型权重

1. 下载Finetune训练所需模型权重

   在**black-forest-labs/FLUX.1-dev at main (huggingface.co)**下载flux模型权重和vae模型权重flux1-dev.safetensors, ae.safetensors

   在**comfyanonymous/flux_text_encoders at main (huggingface.co)**下载t5xxl和clip权重clip_l.safetensors, t5xxl_fp16.safetensors

   以上4个权重文件全部上传到宿主机工作目录${work_dir}/下。

2. 下载lora训练所需模型权重

   FLUX.1-dev下载链接：**https://huggingface.co/black-forest-labs/FLUX.1-dev/tree/main**

   下载后全部上传到宿主机对应工作目录${work_dir}/FLUX.1-dev下。

步骤五：下载数据集

数据集下载地址：**PixArt-alpha/pixart-sigma-toy-dataset · Datasets at Hugging Face**

数据集下所有文件全部上传到宿主机对应工作目录${work_dir}/datasets/pixart-sigma-toy-dataset

修改数据集格式：

只需在数据集根目录创建个.py文件，读取其数据集格式做成flux数据集即可。

vim ${work_dir}/datasets/data.py

```
#---------------data.py 代码如下----------------------#
import json
import re
import shutil
import os
import copy
input_file_path = 'pixart-sigma-toy-dataset'
input_json = os.path.join(input_file_path, 'InternData', 'data_info.json')
if not os.path.exists('images_txt_datasets'):
    os.makedirs('images_txt_datasets')
with open(input_json, 'r', encoding='utf-8') as file:
    datas = json.load(file)
for i, data in enumerate(datas):
    print(f'--{i}')
    image_name = data['path']
    image_path = os.path.join(input_file_path, 'InternImgs', image_name)
    new_image_path = os.path.join('images_txt_datasets', image_name)
    shutil.copy2(image_path, new_image_path)
    txt_name = image_name.replace('png', 'txt')
    txt_path = os.path.join('images_txt_datasets', txt_name)
    with open(txt_path, 'w') as file:
        file.write(data['prompt'])
```

步骤六：设置宿主机文件权限

```
chmod -R 777 ${work_dir}
```

步骤七：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

步骤八：进入容器执行数据集格式调整脚本

```
cd ${container_work_dir}/datasets/
python data.py
```

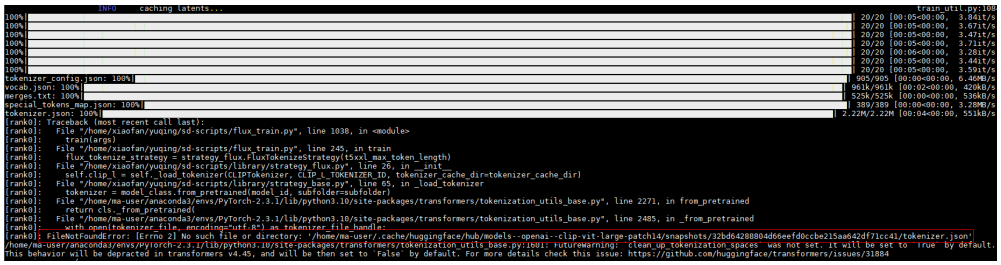
执行成功后，当前目录下会生成满足格式要求的数据集目录images_txt_datasets。

步骤九：进入容器运行 Finetune 训练

```
cd ${container_work_dir}/
\cp flux/fintune/* ./
sh prepare.sh
cd sd-scripts
sh run.sh
```

需要注意的是，如果报错如下，则重新执行sh run.sh即可。

图 6-16 报错



步骤十：进入容器运行 Lora 训练

```
cd ${container_work_dir}/
rm -rf sd-scripts
\cp flux/lora/* ./
sh prepare.sh
cd ai-toolkit
sh run.sh
```

在运行**步骤九：进入容器运行Finetune训练**后再做Lora训练时需要执行rm -rf sd-scripts命令，如果仅执行Lora训练则不需执行rm -rf sd-scripts命令。

6.3 Hunyuan-DiT 基于 Lite Server 部署适配 PyTorch NPU 推理指导（6.3.909）

混元DiT，一个基于Diffusion transformer的文本到图像生成模型，此模型具有中英文细粒度理解能力。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展Hunyuan-DiT使用diffusers框架的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.909版本，请参考**表6-5**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 6-5 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.909软件包中的AscendCloud-AIGC-6.3.909-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-6 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.909版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt9b-20240910112800-2a95df3	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

说明

购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=60g \
   -p 8585:8585 \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。
- -p 8585:8585: 映射端口号

说明

- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

步骤三：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

步骤四：下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.909-xxx.zip文件，获取路径参见**表6-5**。本案例使用的是解压到子目录/aigc_inference/torch_npu/目录下的所有文件，将该目录上传到宿主机上的工作目录下，例如：${container_work_dir}/aigc_inference/torch_npu/，目录结构如下：

```
(PyTorch-2.1.0) [root@devserver-bms-5cbce38e-tmp0831 torch_npu]# ll
total 0
drwx------ 4 root root 78 Sep 19 14:32 comfyui
drwx------ 5 root root 85 Sep 19 14:32 diffusers
drwx------ 3 root root 41 Sep 19 14:32 utils
drwx------ 4 root root 73 Sep 19 14:32 webui
```

步骤五：下载 Hunyuan-DiT 模型并安装依赖

1. **下载Hunyuan-DiT模型**

   从github下载Hunyuan-DiT模型，需要先安装lfs才能下载全部模型权重，Hunyuan-DiT-Diffusers权重文件夹大概27G。

   ```
   cd ${container_work_dir}
   git config --global http.sslVerify false

   # 先安装lfs
   wget https://github.com/git-lfs/git-lfs/releases/download/v3.5.1/git-lfs-linux-arm64-v3.5.1.tar.gz
   tar zxf git-lfs-linux-arm64-v3.5.1.tar.gz
   cd git-lfs-3.5.1/
   bash install.sh

   git lfs install
   git clone https://huggingface.co/Tencent-Hunyuan/HunyuanDiT-Diffusers
   ```

2. **复制执行脚本**

   ```
   cd ${container_work_dir}
   cp ${container_work_dir}/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers/examples/hunyuan_dit_example.py .
   ```

   修改hunyuan_dit_example.py脚本第16行，改为hunyuan-dit模型路径：

   ```
   15  # model select
   16  MODEL_NAME = "Tencent-Hunyuan/HunyuanDiT-Diffusers" # 实际运行建议先下载好，改为路径"/home/ma-user/wxl/dit/HunyuanDiT-Diffusers"
   17  h_list = [1024, 768, 864, 960, 1280, 1280]
   ```

3. **安装ascend_diffusers插件**

   执行以下命令安装华为侧插件ascend_diffusers。

   ```
   pip install -e ${container_work_dir}/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers
   ```

4. **安装依赖**

   运行以下命令进入工作目录，安装所需依赖包。

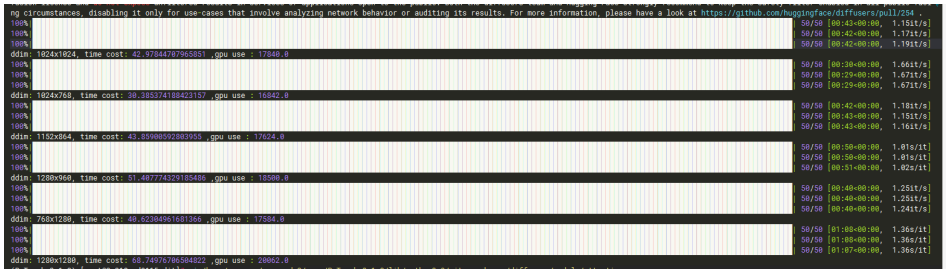
   ```
   cd ${container_work_dir}
   pip install accelerate==0.33.0 diffusers==0.30.3 transformers sentencepiece
   ```


步骤六：Hunyuan-DiT 推理调用

执行推理脚本

```
cd ${container_work_dir}
python hunyuan_dit_example.py
```

查看结果



6.4 SD3.5 基于 Lite Server 适配 PyTorch NPU 的推理指导（6.3.912）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

本文基于diffusers和comfyui两个框架进行适配。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SD3.5模型的推理过程。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B单机。

表 6-7 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 6-8 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表6-8**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

5. 获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参见表1：获取软件和镜像

```
docker pull {image_url}
```

6. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
docker run -itd \
--name ${container_name} \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-p 8443:8443 \
-v /etc/localtime:/etc/localtime \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
--shm-size 60g \
--device=/dev/davinci_manager \
--device=/dev/hisi_hdc \
--device=/dev/devmm_svm \
--device=/dev/davinci1 \
--network=bridge \
${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如sdxl-diffusers。
- --device=/dev/davinci1：挂载主机的/dev/davinci3到容器的/dev/davinci1。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
- ${image_name} 代表 ${image_name}。
- -p 8443:8443：容器内映射到宿主机的端口号，如果已被占用可以使用其他未占用的端口号

7. 进入容器。需要将${container_name}替换为实际的容器名称。

```
docker exec -it ${container_name} bash
```

步骤二：上传代码、权重和数据集到容器中

1. 安装插件代码包。

   将获取到的插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件上传到容器的/home/ma-user目录下，并解压。

```
cd /home/ma-user
unzip AscendCloud-AIGC-6.3.912-*.zip #解压
```

2. 下载模型权重，上传到容器的/home/ma-user目录下，官网下载地址（需登录）。

   - 对于Diffusers框架，需要下载huggingface全部文件。

     stabilityai/stable-diffusion-3.5-medium：**https://huggingface.co/stabilityai/stable-diffusion-3.5-medium/tree/main**

     stabilityai/stable-diffusion-3.5-large：**https://huggingface.co/stabilityai/stable-diffusion-3.5-large/tree/main**

     如果无法手动下载，可以先在容器内命令行输入以下命令，然后使用个人huggingface token进行登录：

```
huggingface-cli login
```

     登录成功后，直接启动步骤三中的Diffusers推理脚本即可实现自动下载。

- 对于ComfyUI框架，只需要下载safetensors文件即可，即

  https://huggingface.co/stabilityai/stable-diffusion-3.5-medium/blob/main/sd3.5_medium.safetensors

  https://huggingface.co/stabilityai/stable-diffusion-3.5-large/blob/main/sd3.5_large.safetensors

  此外ComfyUI需要额外下载三个text_encoder相关模型：

  https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_l.safetensors

  https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_g.safetensors

  https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/t5xxl_fp16.safetensors

  ComfyUI框架还需要下载推理所需的workflow：

  https://huggingface.co/stabilityai/stable-diffusion-3.5-medium/blob/main/SD3.5M_example_workflow.json

  https://huggingface.co/stabilityai/stable-diffusion-3.5-large/blob/main/SD3.5L_example_workflow.json

步骤三：使用 Diffusers 推理

1. 进入容器中/home/ma-user/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers路径下。

   ```
   cd /home/ma-user/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers
   ```

2. 安装所需依赖包。

   ```
   pip install -e .
   pip install diffusers==0.31.0
   ```

3. 开始推理。

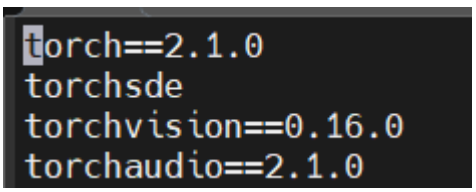
   ```
   export MODEL_NAME='下载好的huggingface模型路径，例如/home/ma-user/stable-diffusion-3.5-medium。如果未手动下载，想要自动下载的话直接配置模型名称即可，例如stabilityai/stable-diffusion-3.5-medium，见步骤二第2节'
   cd examples
   python sd_inference_example.py --sd35 --model_id ${MODEL_NAME} --prompt 'a dog' --num_inference_steps 28 --width 512 512 768 1024 768 --height 512 768 768 1024 1024 --dynamo
   ```

步骤四：使用 ComfyUI 推理

1. 拉取ComfyUI代码。

   ```
   cd /home/ma-user
   git clone -c http.sslVerify=false https://github.com/comfyanonymous/ComfyUI.git
   cd ComfyUI
   # 切换到0.2.7分支
   git reset --hard 6966729
   ```

   修改requirements.txt中的torch/torchvision/torchaudio版本号如下图：

   保存requirements.txt后安装所需依赖：

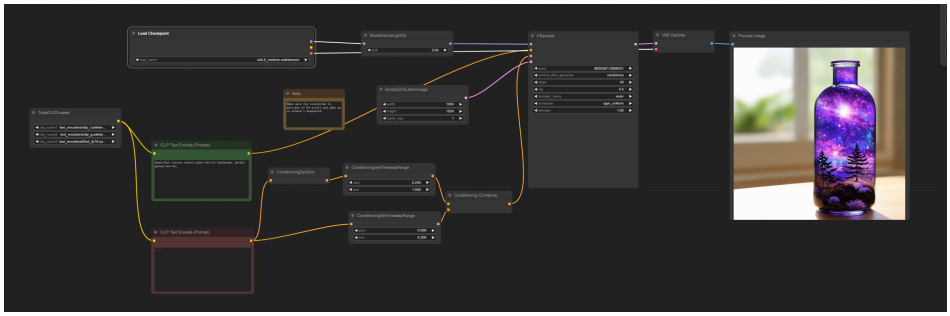
   ```
   # 安装依赖
   pip install -r requirements.txt
   ```

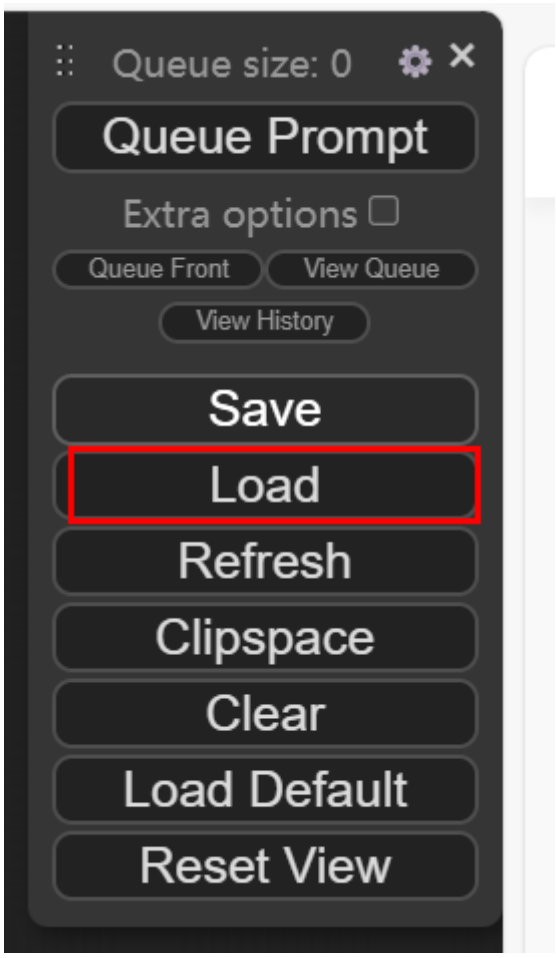

2. 进入容器中/home/ma-user/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers路径下。
```
cd /home/ma-user/aigc_inference/torch_npu/diffusers/0_21_2/ascend_diffusers
```
安装所需依赖包。
```
pip install -e .
pip install diffusers==0.31.0
```
3. 使用comfyui_ascend_node。
```
cp -r /home/ma-user/aigc_inference/torch_npu/comfyui/a82fae2/comfyui_ascend_node /home/ma-user/ComfyUI/custom_nodes/
```
4. 加载权重。

   将下载好的sd3.5_medium.safetensors，sd3.5_large.safetensors到/home/ma-user/ComfyUI/models/checkpoints 目录下；

   将下载好的text_encoder权重（clip_l.safetensors, clip_g.safetensors and t5xxl_fp16.safetensors），放到 /home/ma-user/ComfyUI/models/clip 目录下。
5. 启动ComfyUI。
```
cd /home/ma-user/ComfyUI
export GRAPH_MODE=1
export INF_NAN_MODE_ENABLE=0
python main.py --port 8443 --force-fp16 --listen
```
   --port 端口号与启动容器时映射到宿主机的端口号保持一致。
6. 发送服务请求。
   a. 从浏览器访问ComfyUI服务。在浏览器中输入 http://{ip}:{port}

      这里的ip为宿主机节点ip，port为启动ComfyUI使用的端口号。
   b. 访问界面，页面工作流示例如下图所示。

      
   c. 加载SD3.5workflow文件。单击“Load”，选择下载好的SD3.5M_example_workflow.json或者SD3.5L_example_workflow.json。



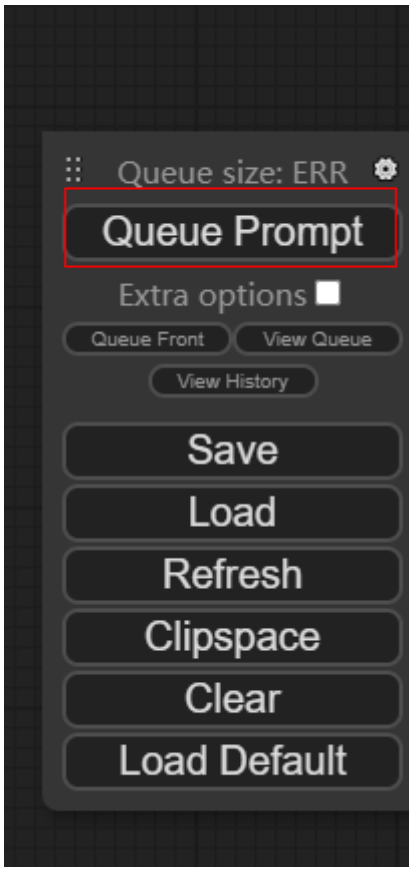
d. 选择diffusion model ，如下图。

如果加载的是SD3.5M_example_workflow.json，这里选择sd3.5_medium.safetensors；如果加载的是SD3.5L_example_workflow.json，这里选择sd3.5_large.safetensors。

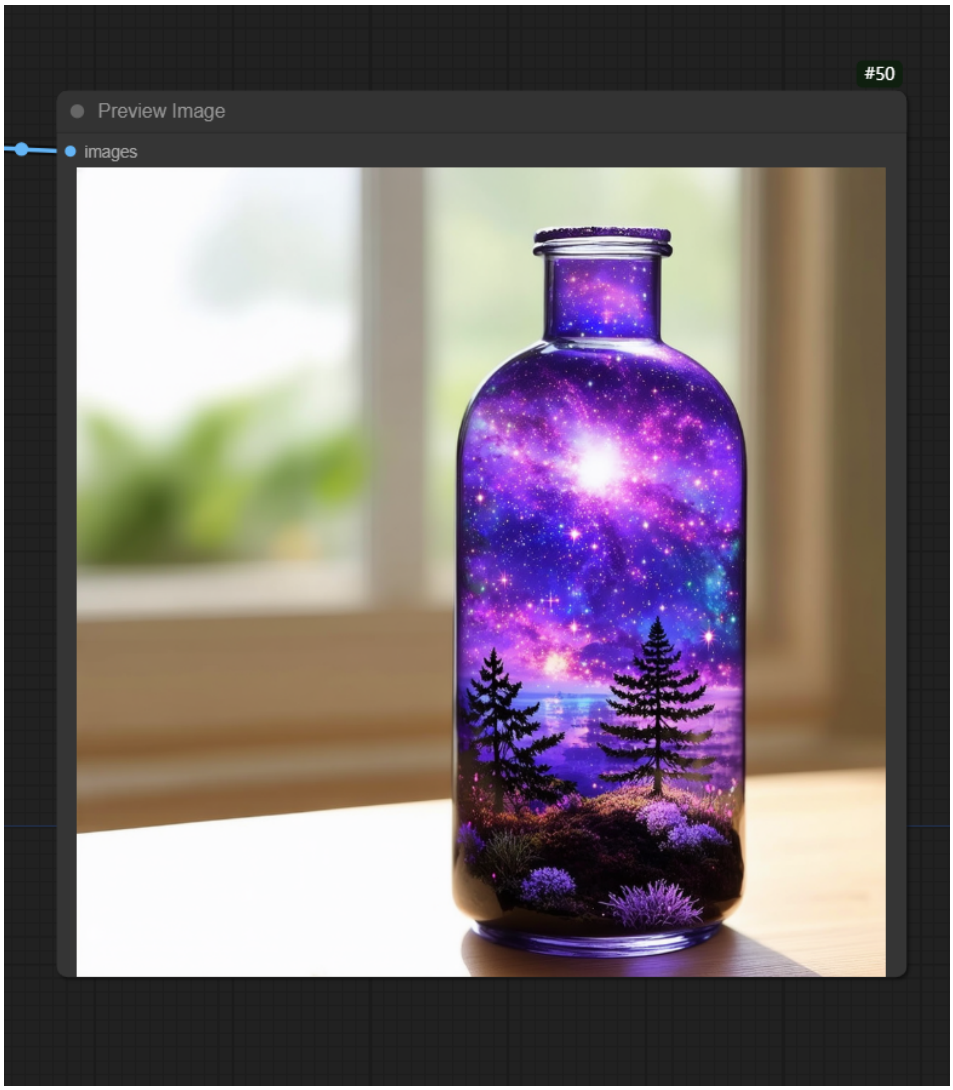
e. 选择clip 模型，如下图。

f. 配置推理的参数，如steps，width，height，batch_size等

g. 单击“Queue Prompt”加入推理队列进行推理，如下图。



成功之后结果如下图所示。首次加载或切换模型推理时，需要加载模型并进行相关初始化工作，首次推理时间较长，请耐心等待。



6.5 SD3 基于 Lite Server 适配 PyTorch NPU 的训练指导（6.3.912）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SD3-模型的训练过程。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B单机。

表 6-9 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 6-10 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	SWR上拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表6-10**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```
5. 获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参见**表2 获取软件和镜像**。
```
docker pull {image_url}
```
6. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。
```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
```
参数说明：
   - --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如sdxl-diffusers。
   - --device=/dev/davinci1：挂载主机的/dev/davinci1到容器的/dev/davinci1。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
   - -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下可存放项目所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同

```
docker run -itd \
--name ${container_name} \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v /etc/localtime:/etc/localtime \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--shm-size 60g \
--device=/dev/davinci_manager \
--device=/dev/hisi_hdc \
--device=/dev/devmm_svm \
--device=/dev/davinci1 \
--network=bridge \
${image_name} bash
```
7. 进入容器。需要将${container_name}替换为实际的容器名称。
```
docker exec -it ${container_name} bash
```

步骤二：上传代码、权重和数据集到容器中

1. 安装插件代码包。将获取到的插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件上传到容器的/home/ma-user目录下，并解压。
```
cd /home/ma-user
unzip AscendCloud-AIGC-6.3.912-*.zip #解压
```
2. 下载模型权重，上传到容器的/home/ma-user目录下，官网下载地址：https://huggingface.co/stabilityai/stable-diffusion-3-medium-diffusers/tree/main（需登录）

3. 下载公开数据集，上传到容器的/home/ma-user目录下，官网下载地址：https://huggingface.co/datasets/diffusers/dog-example/tree/main。

   下载好之后删除数据集中的.gitattributes文件。

步骤三：开始训练

1. 进入容器中/home/ma-user/aigc_train/torch_npu/sd3路径下

   ```
   cd /home/ma-user/aigc_train/torch_npu/sd3
   ```

2. 安装依赖

   ```
   sh prepare.sh
   ```

   如果这一步安装依赖失败，是部分依赖之间有冲突，手动在终端依次执行如下命令解决。

   ```
   pip install wandb
   pip install urllib3==1.26.7
   cp run.sh diffusers/examples/dreambooth/
   ```

3. 开始训练。

   a. 进入diffusers/examples/dreambooth

      ```
      cd diffusers/examples/dreambooth
      ```

   b. 修改run.sh文件的前两行为模型权重和数据集路径。bs，step，resolution用给定参数。

      ```
      export MODEL_NAME="/home/ma-user/sd3/stable-diffusion-3-medium-diffusers"
      export INSTANCE_DIR="/home/ma-user/sd3_works/dog"
      export OUTPUT_DIR="trained-sd3"
      export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
      export WANDB_MODE=offline
      export WANDB_DISABLED=TRUE

      accelerate launch train_dreambooth_sd3.py \
        --pretrained_model_name_or_path=$MODEL_NAME  \
        --instance_data_dir=$INSTANCE_DIR \
        --output_dir=$OUTPUT_DIR \
        --mixed_precision="fp16" \
        --instance_prompt="a photo of sks dog" \
        --resolution=1024 \
        --train_batch_size=1 \
        --learning_rate=1e-4 \
        --report_to="wandb" \
        --lr_scheduler="constant" \
        --lr_warmup_steps=0 \
        --max_train_steps=500 \
        --validation_epochs=1 \
        --seed="0
        "
      ```

   c. 运行run.sh

      ```
      sh run.sh
      ```

4. 训练过程中，终端弹出wandb选项，如下图所示，输入3即可。

   ```
   wandb: (1) Create a W&B account
   wandb: (2) Use an existing W&B account
   wandb: (3) Don't visualize my results
   wandb: Enter your choice: 3
   ```

   训练成功如下图所示。

6.6 SD3 Diffusers 框架基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.912）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SD3模型的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表6-11**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B。

软件配套版本

表 6-11 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912软件包中的AscendCloud-AIGC-6.3.912-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-12 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.912版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.912-xxx.zip文件，获取路径参见**表6-11**。本案例使用的是解压到子目录aigc_inference/torch_npu/diffusers/0.29.2/目录下的所有文件，将该目录上传到宿主机上。

步骤三：构建镜像

基于官方提供的基础镜像构建自定义镜像diffusers-sd3-inference:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见表6-12。

```
FROM {image_url}

RUN mkdir  /home/ma-user/diffusers
COPY --chown=ma-user:ma-group diffusers/0.29.2 /home/ma-user/diffusers

WORKDIR /home/ma-user/diffusers

RUN dos2unix diffusers_sd3.patch

RUN cd /home/ma-user/diffusers && sh prepare.sh

RUN cp attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py

RUN pip install transformers
RUN pip install accelerate
RUN pip install sentencepiece
```

构建自定义镜像diffusers-sd3-inference:0.0.1。

```
docker build -t diffusers-sd3-inference:0.0.1 .
```

步骤四：启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

步骤五：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

步骤六：启动推理

本章节介绍SD3模型的推理过程。使用官方提供的已经训练好的模型进行推理，输入prompt生成指定像素的图片。

1. 使用如下命令登录huggingface，并输入个人账号的token：

   ```
   huggingface-cli login
   ```

2. 执行如下命令运行推理脚本启动SD3服务：

   ```
   #配置环境变量
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
   ```

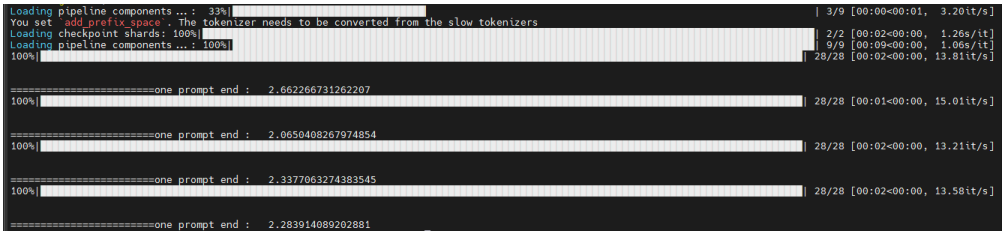
```
python run_inference.py
```

参数说明：

- height、width: 指定生成图片的长和宽，例如：512、960、1024
- prompt_list: prompt列表，可以自行修改。

推理执行成功如下图所示。

图 6-17 推理执行成功



6.7 SD1.5&SDXL Diffusers 框架基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.908）

6.7.1 训练场景和方案介绍

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SDXL和SD1.5模型的训练过程，包括Finetune训练、LoRA训练和Controlnet训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**表6-13**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Finetune训练使用单机8卡资源。
- Lora训练使用单机单卡资源。
- Controlnet训练使用单机单卡资源。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 6-13 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.908软件包中的AscendCloud-AIGC-6.3.908-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-14 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.908版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

6.7.2 准备镜像环境

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载模型包、依赖代码包和数据集并上传到宿主机

1. 下载**stable-diffusion-v1-5模型包**并上传到宿主机上，官网下载地址：https://huggingface.co/benjamin-paine/stable-diffusion-v1-5/tree/main（需登录）
2. 下载**stable-diffusion-xl-base-1.0模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
3. 下载**vae-fp16-fix模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
4. 下载开源数据集**naruto-blip-captions**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/naruto-blip-captions/tree/main**。用户也可以使用自己的数据集。
5. 下载开源数据集**fill50k**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/fusing/fill50k/tree/main**。用户也可以使用自己的数据集。
6. 下载华为侧插件代码包AscendCloud-AIGC-6.3.908-xxx.zip文件，获取路径参见**表6-13**。本案例使用的是解压到子目录aigc_train->torch_npu->diffusers的所有文件，将diffusers整个目录上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。
```
[root@devserver docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 diffusers
drwx------  3 root root   4096 May  7 10:50 datasets
  drwx------  3 root root   4096 May  7 10:50 naruto-blip-captions
  drwx------  3 root root   4096 May  7 10:50 fill50k
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3 构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-v1-5
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像diffusers-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表6-14**。
```
FROM {image_url}
```



```
COPY --chown=ma-user:ma-group diffusers /home/ma-user/diffusers
RUN cd /home/ma-user/diffusers && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-v1-5 /home/ma-user/stable-diffusion-v1-5
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
WORKDIR /home/ma-user/diffusers
```

构建自定义镜像diffusers-train:0.0.1。

```
docker build -t diffusers-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像，finetune全量微调需要启动8卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

启动容器镜像，lora微调和controlnet训练只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

参数说明：

--name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，finetune全量微调示例中挂载了8张卡davinci0~davinci7。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step5 进入容器

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```

6.7.3 Finetune 训练

本章节介绍SDXL&SD 1.5模型的Finetune训练过程。Finetune是指在已经训练好的模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能。

启动 SD1.5 Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_finetune_train.sh
```

启动 SDXL Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_sdxl_finetune_train.sh
```

说明

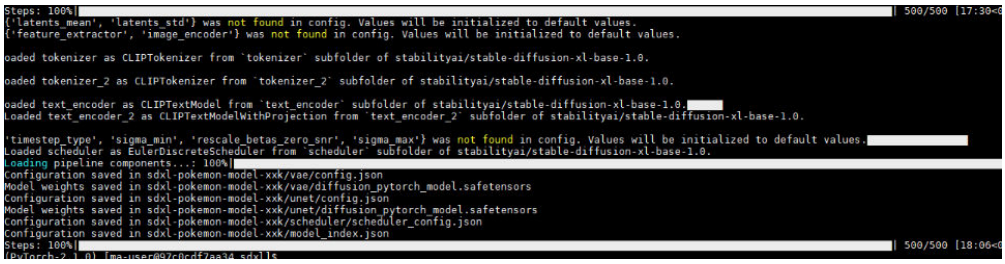
训练执行脚本中配置了保存checkpoint的频率，每500steps保存一次，如果磁盘空间较小，这个值可以改大到5000，避免磁盘空间写满，导致训练失败终止。

checkpoint保存频率的修改命令如下：

```
--checkpointing_steps=5000
```

训练执行成功如下图所示。

图 6-18 训练执行成功



6.7.4 LoRA 训练

本章节介绍SDXL&SD 1.5模型的LoRA训练过程。LoRA训练是指在已经训练好的模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。

启动 SD1.5 LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_lora_train.sh
```

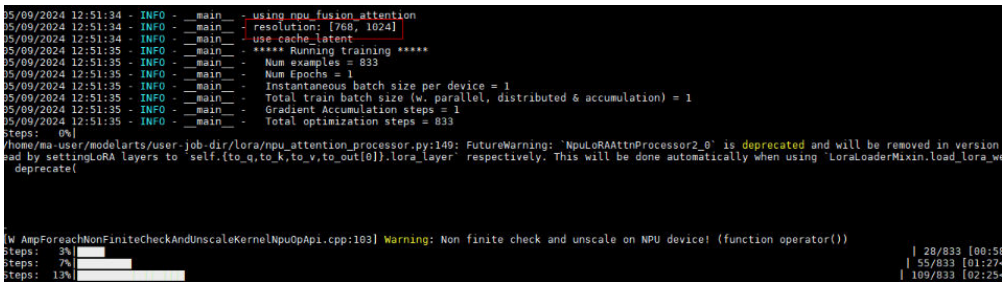
启动 SDXL LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_sdxl_lora_train.sh
```

训练执行成功如下图所示。

图 6-19 训练执行成功



6.7.5 Controlnet 训练

使用文本提示词可以生成一副精美的画作，然而无论再怎么精细地使用提示词来指导模型，也无法描述清楚人物四肢的角度、背景中物体的位置、光线照射的角度，使用Controlnet可以通过图像特征来为扩散模型的生成过程提供更加精细控制的方式。

将Controlnet适配到昇腾卡进行训练，可以提高能效、支持更大模型和多样化部署环境，提升昇腾云在图像生成和编辑场景下的竞争力。

本章节介绍SDXL&SD 1.5模型的Controlnet训练过程。

Step1 处理 fill50k 数据集

使用ma-user用户在容器上执行如下命令解压数据集。

```
cd /home/ma-user/datasets/fill50k
unzip conditioning_images.zip
unzip images.zip
```

接着修改fill50k.py文件，如果机器无法访问huggingface网站，则需要将脚本文件中下载地址替换为容器本地目录。

```
56    def _split_generators(self, dl_manager):
57        #metadata_path = dl_manager.download(METADATA_URL)
58        #images_dir = dl_manager.download_and_extract(IMAGES_URL)
59        #conditioning_images_dir = dl_manager.download_and_extract(
60        #    CONDITIONING_IMAGES_URL
61        #)
62        metadata_path = "/home/ma-user/datasets/fill50k/train.jsonl"
63        images_dir = "/home/ma-user/datasets/fill50k"
64        conditioning_images_dir = "/home/ma-user/datasets/fill50k"
```

Step2 启动 SD1.5 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
cd /home/ma-user/diffusers
sh diffusers_controlnet_train.sh
```

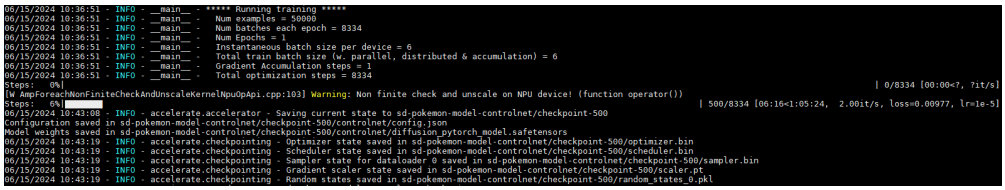
Step3 启动 sdxl 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
cd /home/ma-user/diffusers
sh diffusers_sdxl_controlnet_train.sh
```

训练执行成功如下图所示。

图 6-20 训练执行成功



6.8 SD1.5&SDXL Kohya 框架基于 DevServer 适配 PyTorch NPU 训练指导（6.3.908）

6.8.1 训练场景和方案介绍

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SDXL和SD1.5模型的训练过程，包括Finetune训练、LoRA训练和Controlnet训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**表6-15**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Finetune训练使用单机8卡资源。
- Lora训练使用单机单卡资源。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 6-15 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.908软件包中的AscendCloud-AIGC-6.3.908-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-16 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.908版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

6.8.2 准备镜像环境

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v  #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 下载模型包、依赖代码包和数据集并上传到宿主机

1. 下载**stable-diffusion-v1-5模型包**并上传到宿主机上，官网下载地址：https://huggingface.co/benjamin-paine/stable-diffusion-v1-5/tree/main （需登录）

2. 下载**stable-diffusion-xl-base-1.0模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
3. 下载**sdxl-vae-fp16-fix模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
4. 下载开源数据集**pokemon-dataset**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/sayannath/pokemon-dataset/tree/main**。用户也可以使用自己的数据集。
5. 下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表6-15**。本案例使用的是解压到子目录aigc_train->torch_npu->koyha_ss的所有文件，将koyha_ss整个目录上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。

```
[root@devserver docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 koyha_ss
drwx------  3 root root   4096 May  7 10:50 datasets
  drwx------  3 root root   4096 May  7 10:50 pokemon-dataset
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3 构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-v1-5
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
drwx------  2 root root   4096 Apr 30 15:17 sdxl-vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像koyha_ss-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表6-16**。

```
FROM {image_url}

COPY --chown=ma-user:ma-group koyha_ss /home/ma-user/koyha_ss
RUN cd /home/ma-user/koyha_ss && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-v1-5 /home/ma-user/stable-diffusion-v1-5
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group sdxl-vae-fp16-fix /home/ma-user/sdxl-vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
WORKDIR /home/ma-user/koyha_ss
```

构建自定义镜像diffusers-train:0.0.1。

```
docker build -t koyha_ss-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数，Lora微调启动单卡，finetune微调启动八卡。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge koyha_ss-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

说明

- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step4 进入容器

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
```
docker exec -it ${container_name} bash
```
2. 上传代码文件到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

6.8.3 Finetune 训练

本章节介绍SDXL&SD 1.5模型的Finetune训练过程。Finetune是指在已经训练好的模型基础上，使用新的数据集进行微调（ fine-tuning ）以优化模型性能。

训练前需要修改数据集路径、模型路径。数据集路径格式为/datasets/pokemon-dataset/image_0.png，脚本里写到pokemon-dataset路径即可。

将kohya_finetune.toml文件里数据集路径更改为pokemon-dataset路径。

```
cd koyha_ss
cp run_* sd-scripts
cp kohya_finetune.toml sd-scripts
cd sd-scripts
vim run_finetune.sh
vim kohya_finetune.toml
python finetune/make_captions.py {数据集路径pokemon-dataset路径}
python finetune/merge_captions_to_metadata.py {数据集路径pokemon-dataset路径} meta_cap.json
```

创建default_config.yaml文件，并将以下配置粘贴进去。

```
compute_environment: LOCAL_MACHINE
debug: false
distributed_type: MULTI_NPU
downcast_bf16: 'no'
gpu_ids: all
machine_rank: 0
main_training_function: main
mixed_precision: fp16
num_machines: 1
num_processes: 8
rdzv_backend: static
same_network: true
tpu_env: []
tpu_use_cluster: false
tpu_use_sudo: false
use_cpu: false
```

启动 SD1.5 Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。
```
sh run_finetune.sh
```

所有数据保存在auto_log/avg_step_time.txt文本中 auto_log/log/目录下存放各个shapes的数据

6.8.4 LoRA 训练

本章节介绍SDXL&SD 1.5模型的LoRA训练过程。LoRA训练是指在已经训练好的模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。

训练前需要修改数据集路径、模型路径。脚本里写到datasets路径即可。

run_lora_sdxl中的vae路径要准确写到sdxl_vae.safetensors文件路径。

```
vim run_lora.sh
vim run_lora_sdxl.sh
```

启动 SD1.5 LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh run_lora.sh
```

所有数据保存在auto_log/avg_step_time.txt文本中 auto_log/log/目录下存放各个shapes的数据。

启动 SDXL LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh run_lora_sdxl.sh
```

所有数据保存在autoxl_log/avg_step_time.txt文本中 autoxl_log/log/目录下存放各个shapes的数据。

6.9 SDXL 基于 Standard 适配 PyTorch NPU 的 LoRA 训练指导（6.3.908）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。SDXL LoRA是指在已经训练好的SDXL模型基础上，使用新的数据集进行LoRA微调。

本文档主要介绍如何在ModelArts Standard上，利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL LoRA训练。

获取软件和镜像

表 6-17 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.908软件包中的AscendCloud-AIGC-6.3.908-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2312-aarch64-snt9b-20240824153350-cebb080	SWR上拉取。

表 6-18 模型镜像版本

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6
PyTorch	2.1.0

约束限制

- 本文档适配昇腾云ModelArts 6.3.908版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练作业使用单机单卡资源。
- 确保容器可以访问公网。
- 本案例仅支持在专属资源池上运行。

Step1 创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

- 硬盘空间：至少200GB。
- 昇腾资源规格：Ascend: 8*ascend-snt9b表示昇腾8卡规格。
- 推荐使用“西南-贵阳一”Region上的昇腾资源。

Step2 创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档需要将运行代码以及输入输出数据存放OBS，请提前创建OBS（参考**创建OBS桶**），例如桶名：sdxl-train。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

Step3 准备代码

在**获取软件和镜像**中，下载并解压代码包。本文档主要使用aigc_train->torch_npu->diffusers下的部分文件，请利用**OBS Browser+工具**将文件夹中内容上传至OBS的代码文件夹code中。

```
obs://<bucket_name>/code
├── diffusers-train.patch
├── prepare.sh
└── diffusers_sdxl_lora_train.sh
```

Step4 下载模型依赖包

请在如下链接中下载好模型依赖包。

- 下载stable-diffusion-xl-base-1.0，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
- 下载vae-fp16-fix，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**

Step5 下载数据集

本案例使用Huggingface提供的**naruto-blip-captions**数据集，官网下载地址：**https://huggingface.co/datasets/lambdalabs/naruto-blip-captions/tree/main**

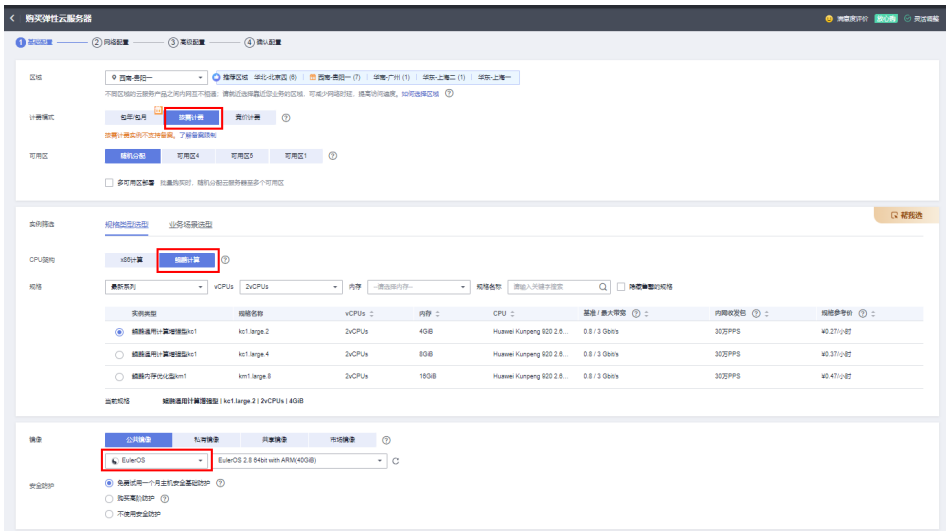
Step6 准备镜像

步骤1 创建ECS。

参考**ECS文档**购买弹性云服务器。网络配置、高级配置等后续步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，并在控制台发送后续步骤中的远程命令。

注意：创建的ECS虚拟机使用ARM镜像创建。

图 6-21 购买 ECS



步骤2 安装Docker。

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤3 构建自定义镜像。

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**获取软件和镜像**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /
home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_sdxl_lora_train.sh /home/ma-user/sdxl-train/user-job-dir/code/
diffusers_sdxl_lora_train.sh

COPY --chown=ma-user:ma-group diffusers-train.patch /home/ma-user/sdxl-train/diffusers-train.patch
COPY --chown=ma-user:ma-group prepare.sh /home/ma-user/sdxl-train/prepare.sh
RUN cd /home/ma-user/sdxl-train && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
```

把上述代码文件、模型依赖包、数据集、Dockerfile文件都上传至ECS，上传步骤可参考**本地Windows主机使用WinSCP上传文件到Linux云服务器**。

文件上传后目录如下：

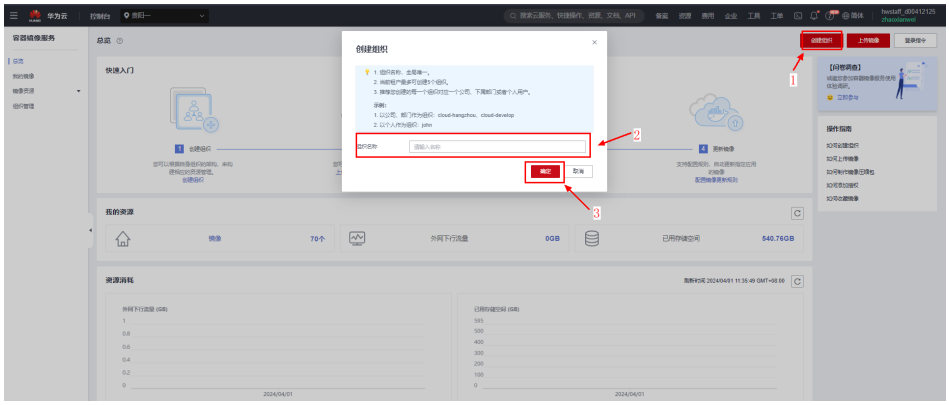
```
<ECS_folder>
├── diffusers_sdxl_lora_train.sh        #  华为侧提供的代码文件
├── diffusers-train.patch     #  华为侧提供的代码文件
├── prepare.sh                        #  华为侧提供的代码文件
├── Dockerfile                         #  Dockerfile文件
├── vae-fp16-fix                       #  模型依赖包vae-fp16-fix
├── stable-diffusion-xl-base-1.0       #  模型依赖包stable-diffusion-xl-base-1.0
├── datasets                          #  新建datasets文件夹，naruto-blip-captions数据集放在该目录下
    ├── naruto-blip-captions
```

在该目录下执行命令构建自定义镜像：

```
docker build -t sdxl-train:0.0.1 .
```

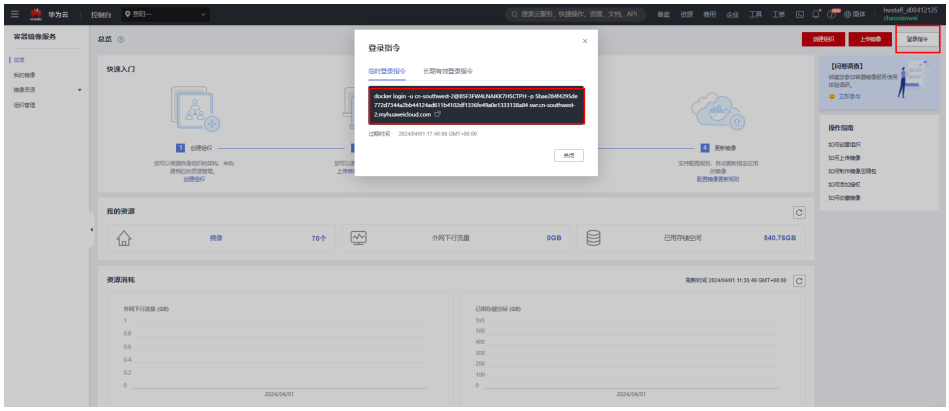
步骤4 在SWR服务页面创建镜像组织。

图 6-22 创建镜像组织



步骤5 在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中复制临时登录指令，即可完成登录。

图 6-23 复制登录指令



步骤6 修改并上传镜像。

在ECS中输入上一步的登录指令后，使用下列示例命令：

```
docker tag {image_url} swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
docker push swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

<组织名称>：步骤4中创建的组织名称。

<镜像名称>:<tag>：定义镜像名称。示例：sdxl-train:0.0.1。

----结束

Step7 创建训练作业

创建训练作业，填下如下参数。

- 创建方式：选择自定义算法，启动方式选择自定义，然后选择上传到SWR的自定义镜像。
- 代码目录：选择上传到OBS的代码文件夹，例如/sdxl-train/code。若用户需要修改代码文件，可修改OBS桶中代码文件，创建训练作业时，会将OBS的code目录复制到训练容器的/home/ma-user/sdxl-train/user-job-dir/目录下，覆盖容器中原有的code目录。

- 启动命令：将华为侧优化后代码文件复制到工作目录后，运行启动脚本文件diffusers_sdxl_lora_train.sh。
  ```
  cd /home/ma-user/sdxl-train/user-job-dir/code && cp /home/ma-user/sdxl-train/train_text_to_image_lora_sdxl.py ./ && sh diffusers_sdxl_lora_train.sh
  ```
- 本地代码目录：保持默认即可。
- 工作目录：选择代码文件目录，例如/home/ma-user/sdxl-train/user-job-dir/code/。
- 输出：单击“增加训练输出”，将模型保存到OBS中。参数名称为output，数据存储位置选择OBS桶中指定文件夹，例如sdxl-train/checkpoint，获取方式选择环境变量，/home/ma-user/modelarts/outputs/output_0下的模型文件会保存到OBS中。

图 6-24 选择镜像

- 资源池：选择专属资源池，规格选择Ascend: 1*ascend-snt9b。

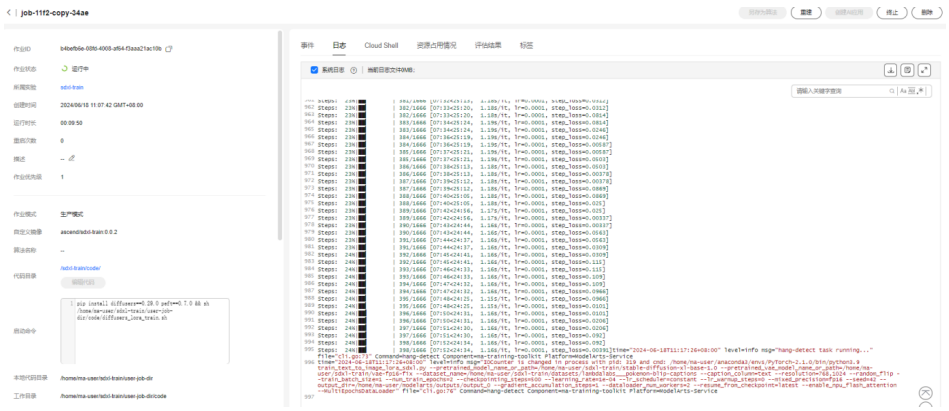
图 6-25 选择资源池规格

- 作业日志路径：选择输出日志到OBS的指定目录。

图 6-26 选择作业日志路径

填写参数完成后，提交创建训练任务，训练完成后，作业状态会显示为已完成。

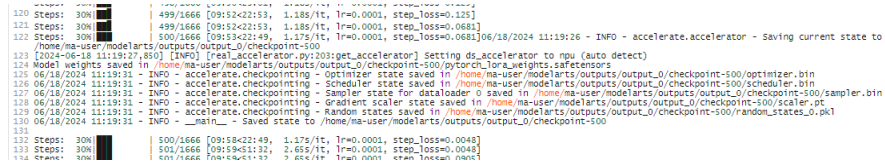
图 6-27 训练启动成功



Step8 断点续训

查看训练日志，在训练任务启动后，当训练超过500步后开始保存checkpoint文件，保存成功后，手动终止训练任务。

图 6-28 保存 checkpoint 文件

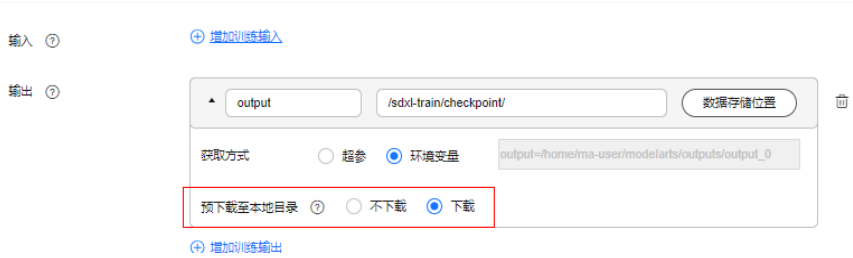


然后单击重建后提交。

图 6-29 重建训练作业

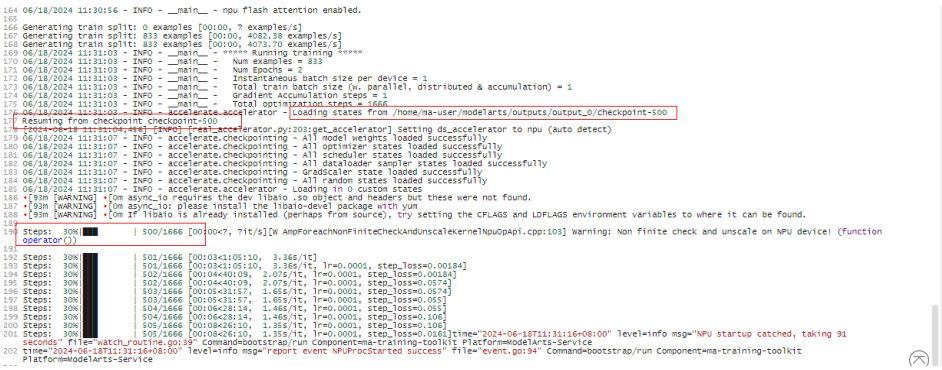
提交新的任务时，注意将预下载到本地目录勾上。

图 6-30 勾选预下载到本地目录



观察启动日志，启动会读取最新的checkpoint模型文件，接着上次保存的step位置开始训练。

图 6-31 读取最新的 checkpoint 模型文件



6.10 SD3 Diffusers 框架基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.907）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。

方案概览

本方案介绍了在ModelArts Lite Server上使用昇腾计算资源Ascend Snt9B开展SD3模型的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表6-19**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 6-19 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 6-20 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表6-19**。本案例使用的是解压到子目录aigc_inference/torch_npu/diffusers/0.29.2/目录下的所有文件，将该目录上传到宿主机上。

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像diffusers-sd3-inference:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表6-20**。

```
FROM {image_url}

RUN mkdir  /home/ma-user/diffusers
COPY --chown=ma-user:ma-group diffusers/0.29.2 /home/ma-user/diffusers

WORKDIR /home/ma-user/diffusers

RUN dos2unix diffusers_sd3.patch

RUN cd /home/ma-user/diffusers && sh prepare.sh

RUN cp attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py

RUN pip install transformers
RUN pip install accelerate
RUN pip install sentencepiece
```

构建自定义镜像diffusers-sd3-inference:0.0.1。

```
docker build -t diffusers-sd3-inference:0.0.1 .
```

Step4 启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。
```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

说明

- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step4 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

Step5 启动推理

本章节介绍SD3模型的推理过程。使用官方提供的已经训练好的模型进行推理，输入prompt生成指定像素的图片。

1. 使用如下命令登录huggingface，并输入个人账号的token：

   ```
   huggingface-cli login
   ```

2. 执行如下命令运行推理脚本启动SD3服务：

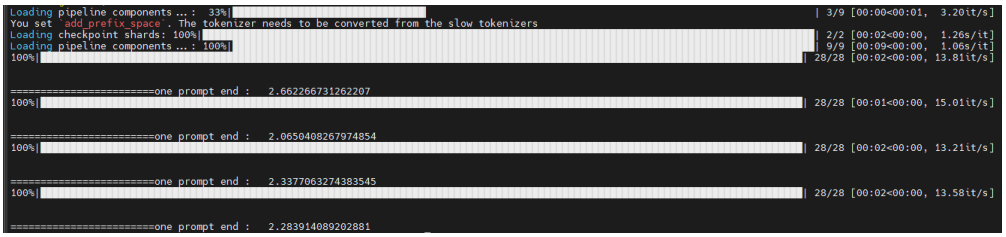
   ```
   #配置环境变量
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

   python run_inference.py
   ```

   参数说明：

   - height、width: 指定生成图片的长和宽，例如：512、960、1024
   - prompt_list: prompt列表，可以自行修改。

   推理执行成功如下图所示。

   图 6-32 推理执行成功

6.11 SDXL&SD1.5 ComfyUI 基于 Lite Cluster 适配 NPU 推理指导（6.3.906）

ComfyUI是一款基于节点工作流的Stable Diffusion操作界面。通过将Stable Diffusion的流程巧妙分解成各个节点，成功实现了工作流的精确定制和可靠复现。每一个节点都有特定的功能，可以通过调整节点连接达到不同的出图效果。在图像生成方面，它不仅比传统的WebUI更迅速，而且显存占用更为经济。

本文档主要介绍如何在ModelArts Lite的Cluster环境中部署ComfyUI，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的Lite Cluster上使用昇腾计算资源部署**ComfyUI**用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Cluster资源。

本方案目前仅适用于企业客户，并且需要用户具备k8s集群相关技能。

资源规格要求

推荐使用“西南-贵阳一”Region上的Cluster资源

表 6-21 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

获取软件和镜像

表 6-22 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

请参考**表6-22**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

本方案使用需要用户具备k8s集群相关技能。

Step1 准备环境

1. 请参考**Cluster资源开通**，购买Cluster资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. 配置Cluster资源，确保可以通过公网访问Cluster机器，具体配置请参见**配置Lite Cluster网络**。

3. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
4. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表6-22**。

```
docker pull {image_url}
```

Step3 下载并安装软件

1. 在宿主机上创建目录/root/comfyui，将下面步骤中所有的文件放到/root/comfyui目录下。
2. 下载模型，模型下载地址：**SD1.5模型地址** ，**SDXL下载地址**。根据自己的需要下载对应的模型。
3. 将获取到的ComfyUI插件AscendCloud-AIGC-6.3.906-xxx.zip文件上传到/root/comfyui，并解压。获取路径参见**表6-22**。
```
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
mv AscendCloud/aigc_inference/torch_npu/comfyui/831511a1eecbe271/comfyui_ascend_node ./
rm -rf AscendCloud*
```
4. 编写dockerfile
```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

RUN cd /home/ma-user && git clone https://github.com/comfyanonymous/ComfyUI.git -c http.sslVerify=false && cd ComfyUI/ && git reset --hard 831511a1eecbe271e302f2f2053f285f00614180 && pip install -r requirements.txt

COPY --chown=ma-user:ma-group v1-5-pruned-emaonly.safetensors /home/ma-user/ComfyUI/models/checkpoints
COPY --chown=ma-user:ma-group sd_xl_base_1.0.safetensors /home/ma-user/ComfyUI/models/checkpoints
COPY --chown=ma-user:ma-group comfyui_ascend_node /home/ma-user/ComfyUI/custom_nodes/comfyui_ascend_node

ENTRYPOINT cd /home/ma-user/ComfyUI && source /usr/local/Ascend/ascend-toolkit/set_env.sh && python main.py --port 30027 --listen 0.0.0.0 --force-fp16
```
5. 基于dockerfile进行build
```
docker build -t comfyui:v1 .
```

Step4 上传镜像到容器镜像服务

参考**pull/push 镜像体验**章节，将上一步build的镜像上传到容器镜像服务上。

Step5 使用 CCE 进行部署

在CCE上**创建工作负载**，创建工作负载时所需的yaml文件可参考**在Lite Cluster资源池上使用Snt9B完成推理任务**。

在CCE上**创建服务**。

6.12 SDXL 基于 Standard 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。SDXL Finetune是指在已经训练好的SDXL模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何在ModelArts Standard上，利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL Finetune训练。

获取软件和镜像

表 6-23 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

表 6-24 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练作业至少需要单机8卡。
- 确保容器可以访问公网。
- 本案例仅支持在专属资源池上运行。

Step1 创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

- 硬盘空间：至少200GB。
- 昇腾资源规格：Ascend: 8*ascend-snt9b表示昇腾8卡规格。
- 推荐使用“西南-贵阳一”Region上的昇腾资源。

Step2 创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档需要将运行代码以及输入输出数据存放OBS，请提前创建OBS（参考**创建OBS桶**），例如桶名：sdxl-train。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

Step3 准备代码

在**获取软件和镜像**中，下载并解压代码包。本文档主要使用ascendcloud-aigc-poc-sdxl-finetune文件夹中的文件，请利用**OBS Browser+工具**将文件夹中内容上传至OBS的代码文件夹code中。

```
obs://<bucket_name>/code
├── attention_processor.py
├── config.yaml
├── diffusers_finetune_train.sh
├── train_text_to_image_sdxl-0212.py
```

Step4 下载模型依赖包

请在如下链接中下载好模型依赖包。

- 下载stable-diffusion-xl-base-1.0，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
- 下载vae-fp16-fix，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**

Step5 下载数据集

本案例使用Huggingface提供的pokemon-blip-captions数据集，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**

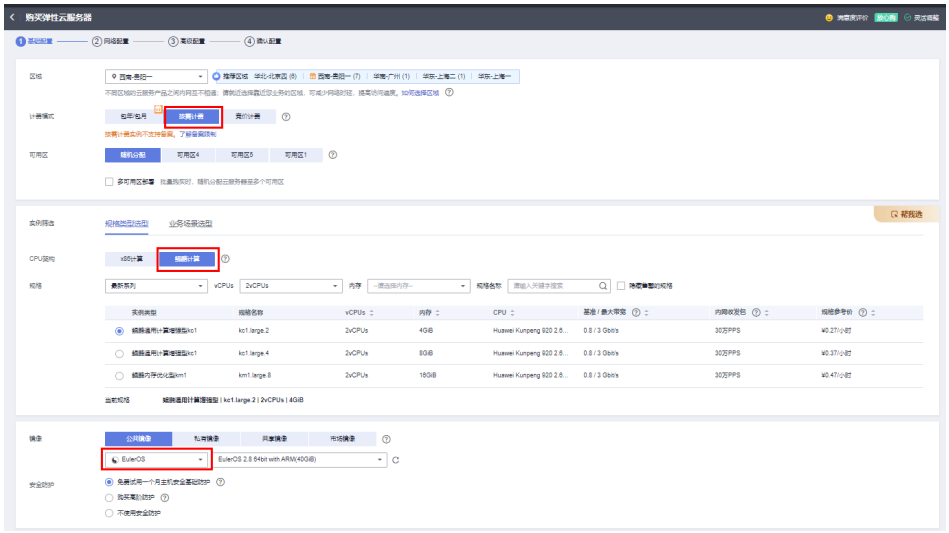
Step6 准备镜像

步骤1 创建ECS。

参考**ECS文档**购买弹性云服务器。网络配置、高级配置等后续步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，并在控制台发送后续步骤中的远程命令。

注意：创建的ECS虚拟机使用ARM镜像创建。

图 6-33 购买 ECS



步骤2 安装Docker。

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤3 构建自定义镜像。

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**获取软件和镜像**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /
home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_finetune_train.sh /home/ma-user/sdxl-train/user-job-dir/code/
diffusers_finetune_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/
code/train_text_to_image_sdxl-0212.py
COPY --chown=ma-user:ma-group config.yaml /home/ma-user/sdxl-train/user-job-dir/code/config.yaml

COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-
base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
COPY --chown=ma-user:ma-group attention_processor.py /home/ma-user/anaconda3/envs/
PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
```


把ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件、模型依赖包、数据集、Dockerfile文件都上传至ECS，上传步骤可参考**本地Windows主机使用WinSCP上传文件到Linux云服务器**。

文件上传后目录如下：

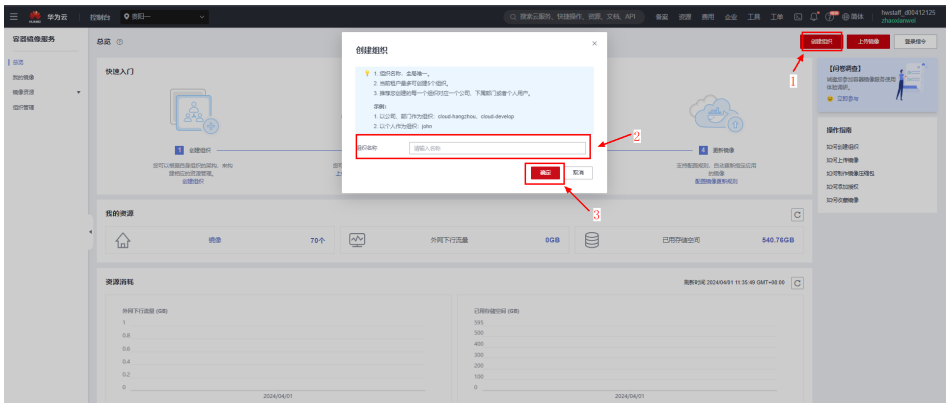
```
<ECS_folder>
├── attention_processor.py              #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── config.yaml                         #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── diffusers_finetune_train.sh         #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── train_text_to_image_sdxl-0212.py    #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── Dockerfile                          #  Dockerfile文件
├── vae-fp16-fix                        #  模型依赖包vae-fp16-fix
├── stable-diffusion-xl-base-1.0x       #  模型依赖包stable-diffusion-xl-base-1.0x
├── datasets                            #  新建datasets文件夹，pokemon-blip-captions数据集放在该目录下
    ├── lambdalabs___pokemon-blip-captions
```

在该目录下执行命令构建自定义镜像：

```
docker build -t sdxl-train:0.0.1 .
```

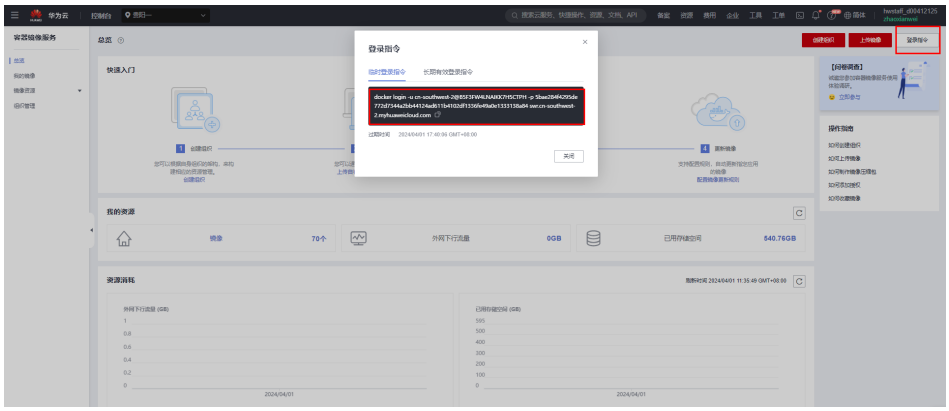
步骤4 在SWR服务页面创建镜像组织。

图 6-34 创建镜像组织



步骤5 在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 6-35 复制登录指令



步骤6 修改并上传镜像。

在ECS中输入上一步的登录指令后，使用下列示例命令：

```
docker tag {image_url} swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
docker push swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

<组织名称>：步骤4中创建的组织名称。

<镜像名称>:<tag>：定义镜像名称。示例：sdxl-train:0.0.1。

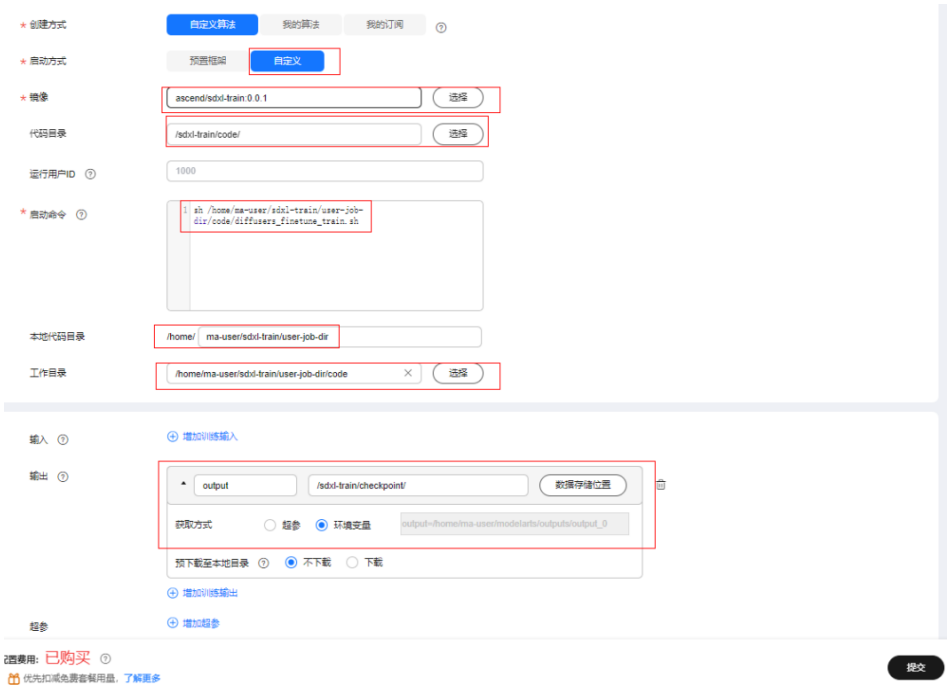
----结束

Step7 创建训练作业

创建训练作业，填下如下参数。

- 创建方式：选择自定义算法，启动方式选择自定义，然后选择上传到SWR的自定义镜像。
- 代码目录：选择上传到OBS的代码文件夹，例如/sdxl-train/code。若用户需要修改代码文件，可修改OBS桶中代码文件，创建训练作业时，会将OBS的code目录复制到训练容器的/home/ma-user/sdxl-train/user-job-dir/目录下，覆盖容器中原有的code目录。
- 启动命令：直接运行启动脚本文件diffusers_finetune_train.sh。
  ```
  sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
  ```
- 本地代码目录：保持默认即可。
- 工作目录：选择代码文件目录，例如/home/ma-user/sdxl-train/user-job-dir/code/。
- 输出：单击“增加训练输出”，将模型保存到OBS中。参数名称为output，数据存储位置选择OBS桶中指定文件夹，例如sdxl-train/checkpoint，获取方式选择环境变量，/home/ma-user/modelarts/outputs/output_0下的模型文件会保存到OBS中。

图 6-36 选择镜像



- 资源池：选择专属资源池，规格选择Ascend: 8*ascend-snt9b。如果需要多机训练，增加计算节点个数即可，启动脚本文件diffusers_finetune_train.sh支持多机训练。

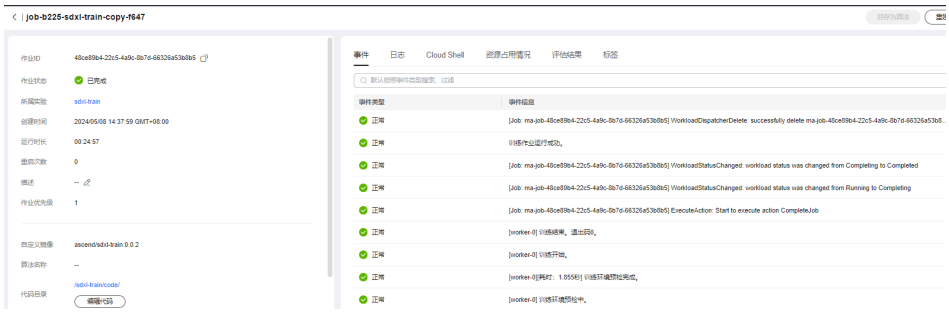
图 6-37 选择资源池规格

- 作业日志路径：选择输出日志到OBS的指定目录。

图 6-38 选择作业日志路径

填写参数完成后，提交创建训练任务，训练完成后，作业状态会显示为已完成。

图 6-39 训练完成



6.13 SDXL 基于 Lite Server 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。SDXL Finetune是指在已经训练好的SDXL模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

表 6-25 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

获取软件和镜像

表 6-26 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表6-26**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练资源需要使用单机8卡。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载代码包、依赖模型包和数据集

1. 下载stable-diffusion-xl-base-1.0模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
2. 下载vae-fp16-fix模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
3. 下载开源数据集并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**。用户也可以使用自己的数据集。
4. 下载SDXL插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，获取路径参见**获取软件和镜像**。本案例使用的是AscendCloud-3rdAIGC-6.3.905-xxx.zip文件中的ascendcloud-aigc-poc-sdxl-finetune.tar.gz代码包。解压后上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。
```
[root@devserver-ei-cto-office-ae06cae7-tmp1216 docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 attention_processor.py
-rw-------  1 root root    430 May  8 09:31 config.yaml
drwx------  3 root root   4096 May  7 10:50 datasets
-rw-------  1 root root   1356 May  8 16:30 diffusers_finetune_train.sh
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
-rw-------  1 root root  58048 May  8 17:48 train_text_to_image_sdxl-0212.py
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表6-26**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_finetune_train.sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/code/train_text_to_image_sdxl-0212.py
COPY --chown=ma-user:ma-group config.yaml /home/ma-user/sdxl-train/user-job-dir/code/config.yaml
```

```
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
COPY --chown=ma-user:ma-group attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
```

构建自定义镜像sdxl-train:0.0.1。

```
docker build -t sdxl-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge sdxl-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

进入容器。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it sdxl-train bash
```

Step5 修改算法脚本

进入容器后，修改启动脚本文件。

```
vi /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
```

在第2行增加export MA_NUM_HOSTS=1 即可，如：

```
#!/bin/bash
export MA_NUM_HOSTS=1
if [[ $MA_NUM_HOSTS == 1 ]]; then
```

Step6 启动训练服务

执行如下命令运行训练脚本。

```
cd /home/ma-user/sdxl-train/user-job-dir/code
sh diffusers_finetune_train.sh
```

说明

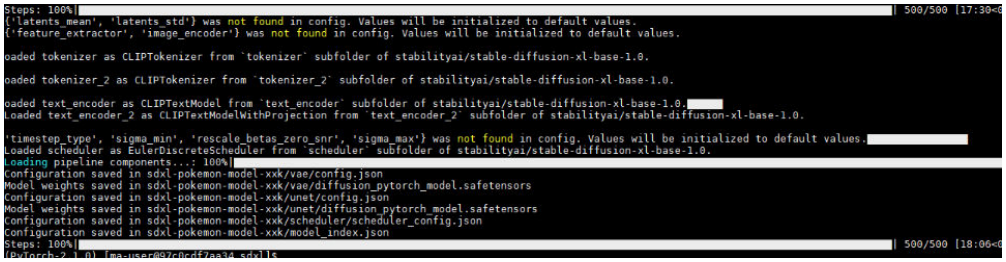
训练执行脚本中配置了保存checkpoint的频率，每500steps保存一次，如果磁盘空间较小，这个值可以改大到5000，避免磁盘空间写满，导致训练失败终止。

checkpoint保存频率的修改命令如下：

```
--checkpointing_steps=5000
```

训练执行成功如下图所示。

图 6-40 训练执行成功



6.14 SDXL 基于 Lite Server 适配 PyTorch NPU 的 LoRA 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助用户生成图像。SDXL LoRA训练是指在已经训练好的SDXL模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL的LoRA微调训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

表 6-27 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

获取软件和镜像

表 6-28 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表6-28**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Lora训练使用单机单卡资源。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 下载代码包、依赖模型包和数据集

1. 下载stable-diffusion-xl-base-1.0模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
2. 下载vae-fp16-fix模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
3. 下载开源数据集并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**。用户也可以使用自己的数据集。
4. 下载SDXL插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，获取路径参见**获取软件和镜像**。本案例使用的是AscendCloud-3rdAIGC-6.3.905-xxx.zip文件中的ascendcloud-aigc-poc-sdxl-lora-train.tar.gz代码包。解压后上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。

   ```
   [root@devserver-ei-cto-office-ae06cae7-tmp1216 docker_build]# ll
   total 192
   ```

```
-rw-------  1 root root 108286 May  6 16:56 npu_attention_processor.py
drwx------  3 root root   4096 May  7 10:50 datasets
-rw-------  1 root root   1356 May  8 16:30 diffusers_lora_train.sh
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
-rw-------  1 root root  58048 May  8 17:48 train_text_to_image_lora_sdxl-0212.py
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表6-28**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_lora_train.sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_lora_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_lora_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/code/train_text_to_image_lora_sdxl-0212.py
COPY --chown=ma-user:ma-group npu_attention_processor.py /home/ma-user/sdxl-train/user-job-dir/code/npu_attention_processor.py

COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
```

构建自定义镜像sdxl-train:0.0.1。

```
docker build -t sdxl-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge sdxl-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0：挂载NPU设备，单卡即可。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

进入容器。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it sdxl-train bash
```

Step5 安装依赖

安装pip依赖。

```
pip install diffusers==0.21.2
```

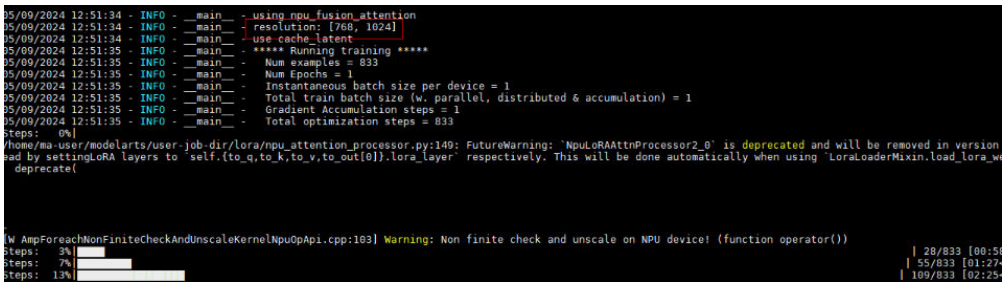
Step6 启动训练服务

执行如下命令启动训练脚本diffusers_lora_train.sh。

```
cd /home/ma-user/sdxl-train/user-job-dir/code
sh diffusers_lora_train.sh
```

训练执行成功如下图所示。

图 6-41 训练执行成功



6.15 SD1.5 基于 Lite Server 适配 PyTorch NPU Finetune 训练指导（6.3.904）

Stable Diffusion（简称SD）是一种基于Latent Diffusion（潜在扩散）模型，应用于文生图场景。对于输入的文字，它将会通过一个文本编码器将其转换为文本嵌入，然后和一个随机高斯噪声，一起输入到U-Net网络中进行不断去噪。在经过多次迭代后，最终模型将输出和文字相关的图像。

SD1.5 Finetune是指在已经训练好的SD1.5模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，对Stable Diffusion模型下不同数据集进行高性能训练调优，同时启用多卡作业方式提升训练速度，完成SD1.5 Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

表 6-29 环境要求

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取软件和镜像

表 6-30 获取软件和镜像

分类	名称	获取路径
插件代码包	ascendcloud-aigc-6.3.904-xxx.tar.gz 文件名中的xxx表示具体的时间戳，以包的实际时间为准。	获取路径：**Support-E网站**。 **说明** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	SWR上拉取

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参考**表6-30**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像地址"
// 启动一个容器去运行镜像
docker run -itd \
        --device=/dev/davinci0 \
        --device=/dev/davinci1 \
        --device=/dev/davinci2 \
        --device=/dev/davinci3 \
        --device=/dev/davinci4 \
        --device=/dev/davinci5 \
        --device=/dev/davinci6 \
        --device=/dev/davinci7 \
        --device=/dev/davinci_manager \
        --device=/dev/devmm_svm \
        --device=/dev/hisi_hdc \
        -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
        -v /usr/local/dcmi:/usr/local/dcmi \
        -v /etc/ascend_install.info:/etc/ascend_install.info \
        -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
        -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
        --shm-size 32g \
        --net=bridge \
        -v ${work_dir}:${container_work_dir} \
        --name ${container_name} \
        ${image_name} bash
```

参数说明：

- work_dir：工作目录，目录下存放着训练所需代码、数据等文件。
- container_work_dir：容器工作目录，一般同work_dir。
- container_name：自定义容器名。
- image_name：容器镜像的名称。

3. 进入容器。需要将${container_name}替换为实际的容器名称。

```
docker exec -it ${container_name} bash
```

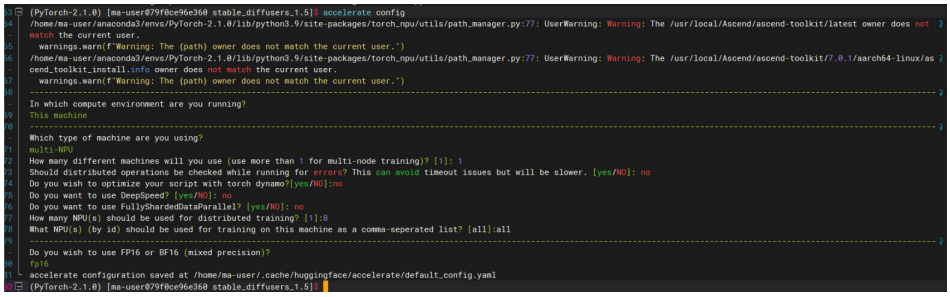
Step3 获取 SD1.5 插件代码包并安装依赖

1. 将下载的SD1.5插件代码包ascendcloud-aigc-xxx-xxx.tar.gz文件，上传到容器的/home/ma-user/目录下，解压并安装相关依赖。插件代码包获取路径参见**表6-30**。

```
mkdir -p /home/ma-user/stable_diffusers_1.5    #创建stable_diffusers_1.5目录
cd /home/ma-user/stable_diffusers_1.5          #进入stable_diffusers_1.5目录

tar -zxvf ascendcloud-aigc-*.tar.gz
tar -zxvf ascendcloud-aigc-poc-stable_diffusers_1.5.tar.gz
rm -rf ascendcloud-aigc-xxx-xxx

pip install -r requirements.txt    #安装依赖
```

2. 启动前配置。有两种方式修改配置文件：
   - 方式一：可以参考解压出来的default_config.yaml或者deepspeed_default_config.yaml文件，再通过在启动脚本命令中增加--config_file=xxx.yaml参数来指定其为配置文件。
   - 方式二：通过命令accelerate config进行配置，如下图所示。

图 6-42 通过命令 accelerate config 进行配置



3. （可选）文件替换。

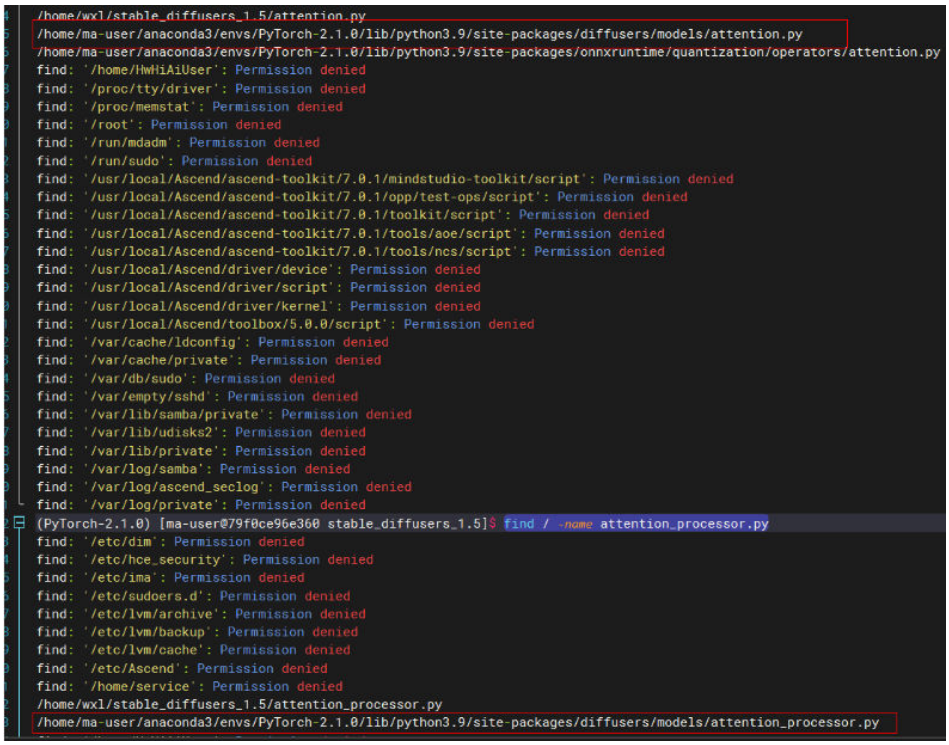
   因增加nfa和使用npu_geglu算子（用于训练和推理加速），将diffusers源码包中的attention.py和attention_processor.py替换成代码包中对应的文件。

图 6-43 文件替换

可以使用find命令来查找diffusers源码包位置。

```
find / -name attention.py
find / -name attention_processor.py
```

图 6-44 查找 diffusers 源码包位置



找到具体位置后可以cp替换，替换前可对diffusers原始文件做备份，如果没有备份则可以通过删除diffusers包重新安装的方式获取原始文件。

4. 执行bash stable_diffusers_train.sh。

```
bash stable_diffusers_train.sh
```

Step4 下载模型和数据集

数据集下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions**。

启动脚本前的两个声明为本次训练的模型和数据集，第一次执行程序时若本地没有模型和数据集，会自动下载。但由于lambdalabs/pokemon-blip-captions数据集下载现在需要登录HuggingFace账号，请先下载数据集到本地，再挂载到对应目录。

```
export MODEL_NAME="runwayml/stable-diffusion-v1-5"
export DATASET_NAME="lambdalabs/pokemon-blip-captions"
```

Step5 启动训练服务

train_text_to_image_0304.py是训练的核心代码，通过stable_diffusers_train.sh来启动。

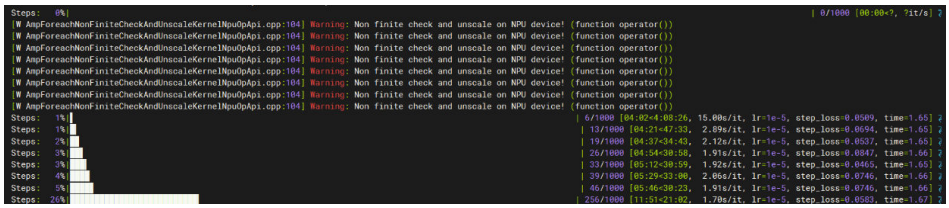
```
sh stable_diffusers_train.sh
```

说明

如果启动前配置采用的是**•可以参考解压出来的default_config...**方式指定配置文件，就是在此stable_diffusers_train.sh脚本中增加--config_file=xxx.yaml参数。

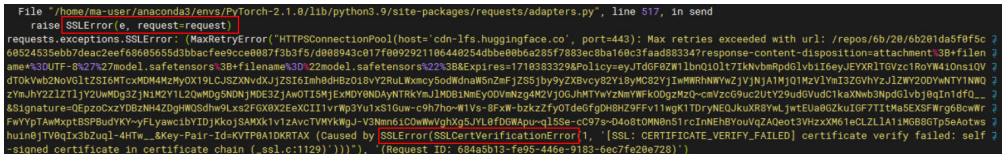
刚开始会报一些Warning，可忽略。正常启动如下图所示，出现Steps: 1%字样。

图 6-45 启动服务



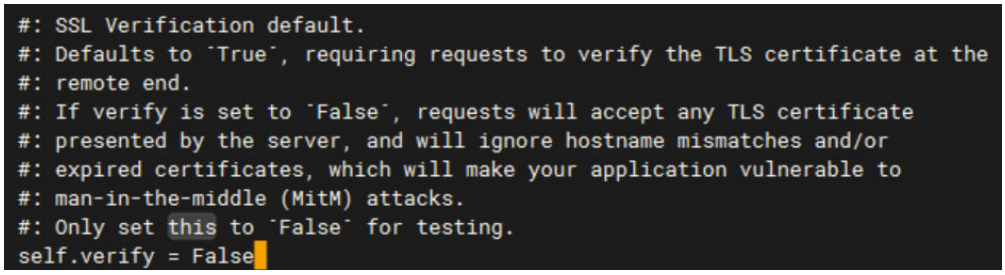
如果启动过程中报SSL相关错误，如下图所示。

图 6-46 启动过程中报 SSL 相关错误



请修改相应路径下的/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/requests/sessions.py文件，将self.verify的值由True改成False，如下图所示。

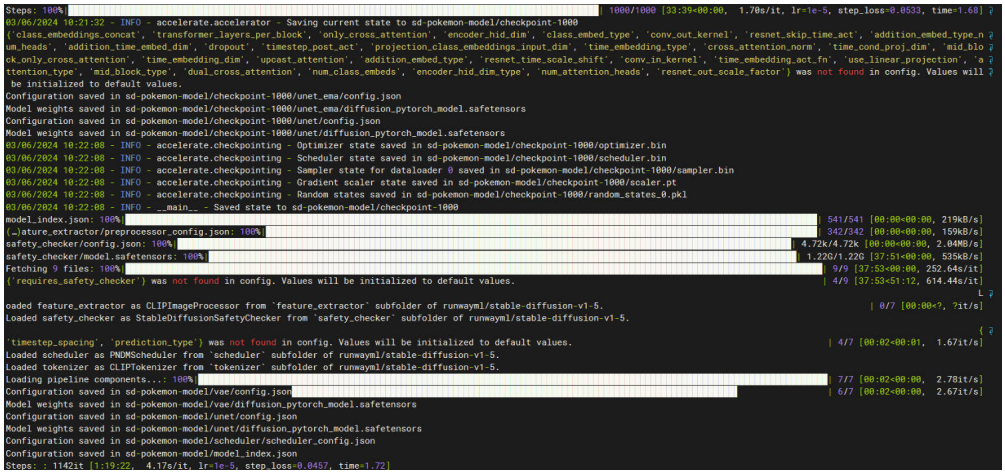
图 6-47 修改 self.verify 参数值



Step6 保存并查看训练结果

正常运行完成训练，会显示如下内容。

图 6-48 训练完成



精度一般问题不大，step_loss都是一个较小值。

训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具TrainingLogParser查看loss收敛情况。

其它注意事项

- 默认500step保存一个checkpoint，可以通过在启动脚本里添加参数--checkpointing_steps=num修改。
- 若显存较低可以调整batch_size保证正常运行，改为8或者更小。
- 本次训练step为1000，训练时间较长，可以改为500。
- 如开启deepspeed训练时，需要设置参数checkpointing_steps>max_train_steps（严格大于），否则会报错。

6.16 Open-Clip 基于 Lite Server 适配 PyTorch NPU 训练指导

Open-Clip广泛应用于AIGC和多模态视频编码器的训练。

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾NPU计算资源开展Open-clip训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B单机单卡。

表 6-31 环境要求

模型	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取镜像

表 6-32 获取镜像

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	从SWR拉取。

获取软件

本教程使用的是Open-clip源码包。

昇腾适配过程通过修改训练脚本方式实现，不涉及其他软件获取。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. 检查环境。

   a. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   b. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   c. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取镜像

获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参考**获取镜像**。

```
docker pull ${image_url}
```

Step3 启动容器

启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /etc/localtime:/etc/localtime \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 32g \
      --net=bridge \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- ${image_name} 代表镜像地址。

通过容器名称进入容器中。使用默认用户ma-user启动容器。

```
docker exec -it ${container_name} bash
```

Step4 下载并安装 Open-clip 源码包

1. 从官网下载Open-clip源码包。

   ```
   git clone https://github.com/mlfoundations/open_clip.git
   cd open_clip
   git reset --hard 37b2c6b321ee697df4c709ca95d6dc849fc7d214
   ```

   37b2c6b321ee697df4c709ca95d6dc849fc7d214是commit号。
2. 复制Open-clip源码包到容器/home/ma-user目录下。

   ```
   docker cp open_clip open-clip:/home/ma-user/
   ```
3. 修改文件夹权限（注意：此处需要重新启动一个终端，使用root用户登录容器，修改文件夹权限，修改完后关闭这个终端。）

   ```
   docker exec -it --user root open-clip bash
   chown -R ma-user:ma-group open_clip
   exit
   ```
4. 在步骤2打开的终端中，使用默认用户ma-user安装源码。

   ```
   cd open_clip
   make install
   ```
5. 在步骤2打开的终端中，使用默认用户ma-user安装依赖。

```
pip install -r requirements-training.txt
pip install -r requirements-test.txt
pip install tensorboard
```

Step5 获取训练数据集

使用img2dataset工具下载数据集。首先需要在容器安装img2dataset，安装命令如下。

```
pip install img2dataset
```

参考**官方指导**下载开源mscoco数据集。

```
#下载metadata
wget https://huggingface.co/datasets/ChristophSchuhmann/MS_COCO_2017_URL_TEXT/resolve/main/mscoco.parquet
#使用img2dataset工具下载数据集
img2dataset --url_list mscoco.parquet --input_format "parquet"\
        --url_col "URL" --caption_col "TEXT" --output_format webdataset\
          --output_folder mscoco --processes_count 16 --thread_count 64 --image_size 256\
           --enable_wandb True
```

Step6 训练 Open clip 模型

1. 适配昇腾代码。

   在目录/home/ma-user/open_clip/src/training下，修改main.py文件，在第10行添加如下代码。

   ```
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   同样，修改train.py文件，在第11行添加如上代码，如**图6-49**所示。

   图 6-49 修改 train.py 文件

   ```
   1 import json
   2 import logging
   3 import math
   4 import os
   5 import time
   6
   7 import numpy as np
   8 import torch
   9 import torch.nn.functional as F
   10 from torch.nn.parallel.distributed import DistributedDataParallel
   11
   12 import torch_npu
   13 from torch_npu.contrib import transfer_to_npu
   14
   ```

2. 单卡训练。

   训练命令参考如下。

   ```
   cd /home/ma-user/open_clip
   python -m training.main \
       --save-frequency 1 \
       --zeroshot-frequency 1 \
       --report-to tensorboard \
       --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
       --train-num-samples 102400 \
       --dataset-type webdataset \
       --warmup 10000 \
       --batch-size=256 \
       --lr=1e-3 \
       --wd=0.1 \
       --epochs=30 \
   ```



```
    --workers=8 \
    --model ViT-B-32
```

参数说明：

- save-frequency：指定运行多少个epoch就保存模型参数，可以调大。
- report-to tensorboard：指定输出loss指标到tensorboard，一般需要做精度评估才需要带上。
- train-num-samples：指定每个epoch需要训练的样本个数，不超过总样本个数。
- batch-size：指定一次处理的数据batch。
- epochs：指定训练的epoch个数。

训练结束后，模型输出目录为：

/home/ma-user/open_clip/logs/xxx-model_ViT-B-32-lr_0.001-b_32-j_8-p_amp/checkpoints

3. 多卡训练

   训练命令参考如下。

```
cd /home/ma-user/open_clip/src
torchrun --nproc_per_node 4 -m training.main \
    --save-frequency 1 \
    --zeroshot-frequency 1 \
    --report-to tensorboard \
    --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
    --train-num-samples 102400 \
    --dataset-type webdataset \
    --warmup 10000 \
    --batch-size=256 \
    --lr=1e-3 \
    --wd=0.1 \
    --epochs=30 \
    --workers=8 \
    --model ViT-B-32
```

Step7 推理验证

首先将上面训练的最终模型文件epoch_xx.pt（ xx取决于训练的epoch个数 ）复制到/home/ma-user/open_clip目录下，然后在/home/ma-user/open_clip下，执行如下命令。

```
vi inference.py
```

将下面的代码复制进去后保存。代码中的epoch_29.pt请替换成实际值。

```
import os
import torch
from PIL import Image
import open_clip

if 'DEVICE_ID' in os.environ:
    print("DEVICE_ID:", os.environ['DEVICE_ID'])
else:
    os.environ['DEVICE_ID'] = "0"

model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', pretrained='/home/ma-user/open_clip/epoch_29.pt')
model = model.to("npu")
tokenizer = open_clip.get_tokenizer('ViT-B-32')

image = preprocess(Image.open("./docs/CLIP.png")).unsqueeze(0)
text = tokenizer(["a diagram", "a dog", "a cat"])
```



```
print("input image shape:", image.shape)
print("input text shape:", text.shape)

with torch.no_grad(), torch.cuda.amp.autocast():
    image = image.to("npu")
    text = text.to("npu")
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)

    print("output image shape:", image_features.shape)
    print("output text shape:", text_features.shape)

    image_features /= image_features.norm(dim=-1, keepdim=True)
    text_features /= text_features.norm(dim=-1, keepdim=True)

    text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-1)

print("Label probs:", text_probs)  # prints: [[1., 0., 0.]]
```

运行推理脚本。

```
python inference.py
```

由于./docs/CLIP.png图片是一张图表，因此结果值和第一个文本"a diagram"吻合，结果值会接近[[1., 0., 0.]]。

Step8 精度评估

1. 关闭数据集shuffle，保证训练数据一致。

   修改/home/ma-user/open_clip/src/training/data.py文件，搜索get_wds_dataset函数，将两处shuffle关闭，修改代码如下。

   ```
   if is_train:
           if not resampled:
               print("dataset unshuffled.")
               #pipeline.extend([
               #    detshuffle2(
               #        bufsize=_SHARD_SHUFFLE_SIZE,
               #        initial=_SHARD_SHUFFLE_INITIAL,
               #        seed=args.seed,
               #        epoch=shared_epoch,
               #    ),
               #    wds.split_by_node,
               #    wds.split_by_worker,
               #])
           print("wds unshuffled.")
           pipeline.extend([
               # at this point, we have an iterator over the shards assigned to each worker at each node
               tarfile_to_samples_nothrow,  # wds.tarfile_to_samples(handler=log_and_continue),
           #    wds.shuffle(
           #        bufsize=_SAMPLE_SHUFFLE_SIZE,
           #        initial=_SAMPLE_SHUFFLE_INITIAL,
           #    ),
           ])
   ```

2. 重新训练1个epoch。脚本参考内容如下。

   ```
   cd /home/ma-user/open_clip
   python -m training.main \
       --save-frequency 1 \
       --zeroshot-frequency 1 \
       --report-to tensorboard \
       --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
       --train-num-samples 102400 \
       --dataset-type webdataset \
       --warmup 10000 \
       --batch-size=256 \
       --lr=1e-3 \
       --wd=0.1 \
   ```

```
  --epochs=1 \
  --workers=8 \
  --model ViT-B-32
```

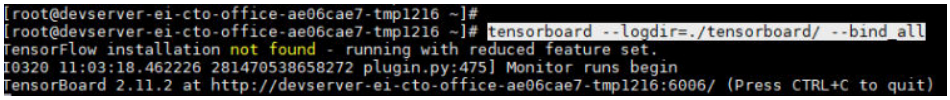
训练完成后，tensorboard统计的记录会保存在/home/ma-user/open_clip/logs/xxx-model_ViT-B-32-lr_0.001-b_32-j_8-p_amp/tensorboard目录下。

3. 通过docker cp命令将容器内tensorboard子目录复制到宿主机 /home下。
4. 在宿主机上安装tensorboard并启动。

```
pip install tensorboard  #安装
tensorboard --logdir=/home/tensorboard --bind_all  #启动
```

启动成功后如下图所示。

图 6-50 启动 tensorboard



5. 在浏览器访问http://{宿主机ip}:6006/。将train/loss导出为json，和GPU训练下导出的文件比较。

6.17 AIGC 工具 tailor 使用指导

tailor 简介

tailor是AIGC场景下用于模型转换（onnx到mindir）和性能分析的辅助工具，当前支持以下功能。

表 6-33 功能总览

功能大类	具体功能
模型转换	• 固定shape转模型 • 动态shape传入指定档位转模型 • 支持fp32 • 支持AOE优化
benchmark	• 支持测试性能 • 支持精度测试
profiling	支持分析算子的profiling

环境准备

本工具支持x86和ARM的系统环境，使用前需要安装以下软件。

表 6-34 安装软件及步骤

软件	安装步骤
mindspore-lite	安装版本：2.2.10
	下载地址：https://www.mindspore.cn/lite/docs/zh-CN/r2.2/use/downloads.html 需要下载的安装包与操作系统有关，请根据需要选择合适的安装包。 • 如果操作系统为Linux aarch64，请下载**mindspore-lite-2.2.10-linux-aarch64.tar.gz**。 • 如果操作系统为Linux x86_64，请下载**mindspore-lite-2.2.10-linux-x64.tar.gz**。

	安装方式如下：
	MindSpore Lite云侧推理包解压缩后，设置LITE_HOME环境变量为解压缩的路径，例如：
	`export LITE_HOME=$some_path/mindspore-lite-2.2.10-linux-aarch64`
	设置环境变量LD_LIBRARY_PATH：
	`export LD_LIBRARY_PATH=$LITE_HOME/runtime/lib:$LITE_HOME/runtime/third_party/dnnl:$LITE_HOME/tools/converter/lib:$LD_LIBRARY_PATH`
	如果需要使用convert_lite或者benchmark工具，则需要设置环境变量PATH。
	`export PATH=$LITE_HOME/tools/converter/converter:$LITE_HOME/tools/benchmark:$PATH`

cann	安装版本：CANN 7.0.0
	下载地址：https://support.huawei.com/enterprise/zh/ascend-computing/cann-pid-251168373/software/258923273?idAbsPath=fixnode01%7C23710424%7C251366513%7C22892968%7C251168373 请下载toolkit和对应机器的kernels包，以Snt9B为例则下载“Ascend-cann-toolkit_7.0.0_linux-aarch64.run”和“Ascend-cann-kernels-型号_7.0.0_linux.run”。

	安装命令（以Snt9B的cann安装为例）：
	`./Ascend-cann-toolkit_7.0.0_linux-aarch64.run --full`
	`./Ascend-cann-kernels-型号_7.0.0_linux.run --install`
	请安装在默认路径下：/usr/local/Ascend，暂不支持安装在自定义路径下。

tailor	安装版本：0.3.4
	下载地址： https://cneast3-modelarts-sdk.obs.cn-east-3.myhuaweicloud.com/tailor-0.3.4-py3-none-any.whl SHA-256： https://cneast3-modelarts-sdk.obs.cn-east-3.myhuaweicloud.com/tailor-0.3.4-py3-none-any.whl/1713929258832/tailor-0.3.4-py3-none-any.whl.sha256
	安装命令： `pip install tailor-0.3.4-py3-none-any.whl`

使用指导

tailor支持“命令行”和“Python API”两种方式使用。

- **命令行方式**

  命令行运行示例：

  ```
  tailor --model_path="./resnet50-v2-7.onnx"--config_path="./config.ini"--
  input_shape="data:1,3,224,224"--output_path="/home/"--accuracy="fp32"--aoe=True
  ```

  config.ini参考内容如下：

  ```
  [ascend_context]
  input_shape=data:[-1,3,224,224]
  dynamic_dims=[1],[2],[3]
  ```

  表 6-35 参数说明

参数名称	功能描述	参数类型	是否必填	默认值	备注
--model_path	指定onnx模型路径。	string	是	-	-
--config_path	指定模型配置文件路径。	string	否	-	tailor支持动态分档转换功能，需要指定配置文件路径，需要注意即便有配置文件，只要是动态模型就需要指定--input_shape参数。
--input_shape	指定模型转换的shape。	string	否	-	固定shape模型转换可以不填，动态模型转换必填。
--output_path	指定结果输出路径。	string	否	默认为当前目录下。	-

参数名称	功能描述	参数类型	是否必填	默认值	备注
--aoe	是否在转换时进行AOE优化。	bool	否	False	AOE优化可以提升模型性能，但不是一定有提升，需要注意开启AOE，会导致模型转换耗时极大延长。
--accuracy	指定模型精度，只支持fp16和fp32。	string	否	fp16	-

- **Python API**
  - 导入包并创建tailor对象。
    ```
    from tailor.tailor import Tailor
    onnx_model_path = "./resnet50-v2-7.onnx"    # 相对路径或者绝对路径均可以。
    t = Tailor(onnx_model_path)
    ```
  - 查询onnx模型的输入信息。
    ```
    # 查询onnx模型的输入信息
    t.get_model_input_info()
    ```

    图 6-51 查询 onnx 模型的输入输出信息

    ```
    In [3]: t.get_model_input_info()
    Out[3]:
    [{'name': 'data',
      'type': 'tensor(float)',
      'shape': ['N', 3, 224, 224],
      'np_type': numpy.float32}]
    ```
  - 查询onnx模型的输出信息。
    ```
    # 查询模型的输出信息
    t.get_model_output_info()
    ```

    图 6-52 查询 onnx 模型的输出信息

    ```
    In [4]: t.get_model_output_info()
    Out[4]:
    [{'name': 'resnetv24_dense0_fwd',
      'type': 'tensor(float)',
      'shape': ['N', 1000]}]
    ```
  - 固定shape模型，可以直接运行。

```
t.run()
```

- 指定档位信息运行。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape)
```

- 动态档位执行config_path运行。需要注意，只要是动态模型，就必须要传入input_shape，因为转换模型后的benchmark和profiling都依赖单个shape操作。

```
input_shape="data:1,3,224,224"
config_path = "./resnet/config.ini"
t.run(input_shape=input_shape, config_path=config_path)
```

- 指定精度为fp32。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape, accuracy='fp32')
```

- 开启AOE优化。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape, aoe=True)
```

- 指定输出位置。

```
input_shape="data:1,3,224,224"
# 不指定输出路径，则默认在当前执行目录存储结果
t.run(input_shape=input_shape, output_path="/home/xxx")
```

运行结果将存储在output文件夹中，如果用户指定了output_path，会指定位置保存，如果不指定则在当前代码执行目录生成文件夹保存输出。整体运行的结果都存放在output文件夹中，每转一次模型就会根据模型名称以及相关参数生成结果文件，如下图所示。

图 6-53 output 文件

```
ll output/
total 8
drwx------ 6 root 4096 Nov  6 15:21 resnet50-v2-7_fp16_20231106152024/
drwx------ 6 root 4096 Nov 15 10:06 resnet50-v2-7_fp16_20231115100511/
drwx------ 6 root 4096 Nov  6 23:57 resnet50-v2-7_fp16_aoe_20231106194012/
drwx------ 6 root 4096 Nov  6 15:33 unet_fp16_20231106152314/
drwx------ 6 root 4096 Nov 10 22:36 unet_fp16_aoe_20231107061337/
drwx------ 6 root 4096 Nov  6 15:23 yolox_m_mmyolo_fp16_20231106152141/
drwx------ 6 root 4096 Nov  7 06:13 yolox_m_mmyolo_fp16_aoe_20231106235726/
```

在每次运行的结果文件中，分为三部分：convert、benchmark、profiling，相关的文件及存储内容如下。

表 6-36 输出文件介绍（以模型名称为 resnet50-v2-7.onnx 为例）

类别	文件名称	是否一定生成	文件存储内容
convert	resnet50-v2-7.mindir	是	转换后的mindir模型。
	resnet50-v2-7.om	否	转换过程中的om文件，不是必定生成。
	onnx_to_mindspore.sh	是	模型转换命令，可以本地直接运行。

类别	文件名称	是否一定生成	文件存储内容
	resnet50-v2-7_convert.log	是	模型转换过程的日志。
	config.ini	否	配置文件，在指定fp32精度或者AOE打开时会生成。
	onnx_to_mindspore_aoe.sh	否	在打开AOE功能时会生成。
benchmark	run_benchmark.sh	是	运行benchmark的脚本，可本地直接运行。
	run_benchmark_accuracy.sh	是	benchmark运行精度的脚本，可本地直接运行。
	performance.txt	是	benchmark性能测试结果。
	accuracy.txt	是	精度测试结果。
	*.bin	是	自动构造的输入随机bin文件，可能存在多个。
	resnet50-v2-7_output.txt	是	上述bin文件作为输入时onnx模型运行的结果。
profiling	run_profiling.sh	是	运行profiling的脚本，可本地直接运行。
	profiling.config	是	运行profiling的配置文件。
	profiling.json	是	运行profiling的配置文件。
	PROF_xxx开头的文件夹	是	运行profiling的结果文件夹。
	run_aggregate.sh	是	运行数据聚合的脚本，可直接本地运行。
	run_profiling.log	是	存储运行profiling的日志信息。

7 文生视频模型训练推理

7.1 CogVideoX1.5 5b 模型基于 Lite Server 适配 PyTorch NPU 全量训练指导（6.3.912）

本文档主要介绍如何在ModelArts的Lite Server环境中，使用NPU卡对CogVideoX模型进行全量微调。本文档中提供的脚本，是基于原生CogVideoX的代码基础适配修改，可以用于NPU芯片训练。

CogVideo是一个94亿参数的Transformer模型，用于文本到视频生成。通过继承一个预训练的文本到图像模型CogView2，还提出了多帧速率分层训练策略，以更好地对齐文本和视频剪辑。作为一个开源的大规模预训练文本到视频模型，CogVideo性能优于所有公开可用的模型，在机器和人类评估方面都有很大的优势。

方案概览

本方案介绍了在ModelArts的Server上使用昇腾计算资源开展CogVideoX1.5 5b全量微调的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Server资源和Ascend Snt9B单机。

表 7-1 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 7-2 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.912-xxx.zip软件包中的AscendCloud-AIGC-6.3.912-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.912 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241213131522-aafe527	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.912版本，请参考**表7-2**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

步骤一：准备环境

1. 请参考**Lite Server资源开通**，购买Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表7-2**。

```
docker pull {image_url}
```

步骤三：启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=256g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下可存放项目所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。
- --device=/dev/davinci0 ： 挂载对应卡到容器，当需要挂载多卡，请依次添加多项该配置

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。


```
docker exec -it ${container_name} bash
```

步骤四：安装依赖和软件包

1. git clone和git lfs下载大模型可以参考如下操作。
   a. 由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到容器的/home/ma-user目录下。
   ```
   https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```
   或直接下载到容器，这样在容器中可以直接使用。
   ```
   cd /home/ma-user
   wget https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```
   b. 进入容器，执行安装git lfs命令。
   ```
   cd /home/ma-user
   tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
   cd git-lfs-3.2.0
   sudo sh install.sh
   ```
   c. 设置git配置去掉ssl校验。
   ```
   git config --global http.sslVerify false
   ```
2. 从github拉取finetrainers代码。
   ```
   cd /home/ma-user
   git clone https://github.com/a-r-r-o-w/finetrainers.git
   cd /home/ma-user/finetrainers
   git checkout 80d1150a0e233a1
   ```
3. 若进行训练微调需依赖decord包，arm版本可参考附录安装编译。
4. 由于当前CogVideoX1.5版本依赖的diffuser暂未合入主线，需安装分支版本diffuser
   ```
   cd /home/ma-user
   git clone https://github.com/zRzRzRzRzRzR/diffusers
   cd /home/ma-user/diffusers/
   git checkout cogvideox1.1-5b
   git checkout ea166f85ad0090d182ec5f0
   pip install -e .
   ```
5. 安装CogVideo Ascend软件包。
   a. 将获取到的CogVideo Ascend软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径参见**获取软件和镜像**。
   b. 解压AscendCloud-AIGC-*.zip文件，解压后将里面指定文件与对应CogVideo文件进行替换，执行以下命令即可。
   ```
   cd /home/ma-user
   unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
   cd AscendCloud/multimodal_algorithm/CogVideoX_1_5/
   dos2unix install.sh
   bash install.sh
   ```

   说明

   AscendCloud-AIGC-*.zip后面的*表示时间戳，请按照实际替换。

   CogVideo Ascend软件包内容如下：
   ```
   .
   |---- install.sh          安装torch-npu适配修改脚本
   |---- modify.patch        适配CogVideo训练代码git patch文件
   |---- README.md           适配文档基于官方代码commit id说明
   |---- requirements.txt    python依赖包
   ```

步骤五：CogVideo 微调

1. **下载模型权重**

   下载CogVideoX1.5 5b模型，huggingface地址如下


```
https://huggingface.co/THUDM/CogVideoX1.5-5B
```

2. **准备数据集**

   数据集可参考使用如下数据集

   ```
   https://huggingface.co/datasets/Wild-Heart/Tom-and-Jerry-VideoGeneration-Dataset
   ```

3. **进行data cache**

   由于CogVideoX1.5对显存需求较大，直接训练显存不足，训练采用data cache，将text encoder和vae两个不参与训练的模型对数据集进行预编码处理。

   ```
   cd /home/ma-user/finetrainers
   ```

   对/home/ma-user/finetrainers/prepare_dataset.sh文件进行修改，配置对应的参数：

   ```
   MODEL_ID="path/CogVideoX1.5-5B" # 模型路径
   DATA_ROOT="path/Tom-and-Jerry-VideoGeneration-Dataset"  # 数据集路径
   CAPTION_COLUMN="captions.txt"   # 数据集提示词文件名
   VIDEO_COLUMN="videos.txt"       # 数据集视频名文件名
   OUTPUT_DIR="path/preprocessed-Tom-dataset"   # 预编码处理的tensor数据集输出路径
   ```

   修改后，执行prepare_dataset.sh脚本

   ```
   bash prepare_dataset.sh
   ```

4. **进行模型训练**

   ```
   cd /home/ma-user/finetrainers
   ```

   对finetrainers/train_text_to_video_sft.sh文件进行修改，配置训练使用的超参数：

   ```
   MAX_TRAIN_STEPS=("20000")    # 最大训练步数
   DATA_ROOT="path/preprocessed-Tom-dataset"  # 预编码处理的tensor数据集路径，即data cache的输出路径
   CAPTION_COLUMN="prompts.txt"        # 数据集提示词文件名
   VIDEO_COLUMN="videos.txt"           # 数据集视频名文件名
   MODEL_PATH="THUDM/CogVideoX1.5-5B"   # 模型路径
   output_dir="/path/to/my/models/cogvideox-sft # 模型输出路径
   ```

   修改后，执行train_text_to_video_sft.sh脚本

   ```
   bash train_text_to_video_sft.sh
   ```

以上微调文档提示来自官方文档，有关可用微调脚本参数及其功能的全面文档，您可以参考**官方finetrainers中CogVideo训练文档**。

附：decord 在 arm 版本安装参考

由于训练使用decord的Python包没有arm版本，请自行进行编译安装，以下给出安装编译教程仅供参考：

- decord包编译

  可参考如下脚本：

  ```
  bash install_decord.sh # install_decord.sh内容如下
  ```

  install_decord.sh脚本内容如下：

  ```
  cd /home/ma-user/
  git config --global http.sslVerify false
  git clone  --recursive https://github.com/dmlc/decord --depth 1
  cd decord
  mkdir build && cd build
  cmake ..  -DCMAKE_BUILD_TYPE=Release -DFFMPEG_DIR:PATH="/usr/local/ffmpeg/"
  make
  cd ../python
  ### option 1: add python path to $PYTHONPATH, you will need to install numpy separately
  echo "PYTHONPATH=$PYTHONPATH:/home/ma-user/decord/python" >> ~/.bashrc
  source ~/.bashrc
  ### option 2: install with setuptools
  python3 setup.py install --user
  ```


7.2 CogVideoX 模型基于 Lite Server 适配 PyTorch NPU 全量训练指导（6.3.911）

本文档主要介绍如何在ModelArts的Lite Server环境中，使用NPU卡对CogVideoX模型基于sat框架进行全量微调。本文档中提供的脚本，是基于原生CogVideoX的代码基础适配修改，可以用于NPU芯片训练。

CogVideo是一个94亿参数的Transformer模型，用于文本到视频生成。通过继承一个预训练的文本到图像模型CogView2，还提出了多帧速率分层训练策略，以更好地对齐文本和视频剪辑。作为一个开源的大规模预训练文本到视频模型，CogVideo性能优于所有公开可用的模型，在机器和人类评估方面都有很大的优势。

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展CogVideoX-2b/5b全量微调的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B单机。

表 7-3 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 7-4 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.911-xxx.zip软件包中的AscendCloud-AIGC-6.3.911-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.911 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.911版本，请参考**表7-4**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

步骤一：准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表7-4**。

```
docker pull {image_url}
```

步骤三：启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下可存放项目所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。
- --device=/dev/davinci0 ： 挂载对应卡到容器，当需要挂载多卡，请依次添加多项该配置

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

步骤四：安装依赖和软件包

1. git clone和git lfs下载大模型可以参考如下操作。

   a. 由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到容器的/home/ma-user目录下。

   ```
   https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```

   或直接下载到容器，这样在容器中可以直接使用。

   ```
   cd /home/ma-user
   wget https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```


b. 进入容器，执行安装git lfs命令。

```
cd /home/ma-user
tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
cd git-lfs-3.2.0
sudo sh install.sh
```

c. 设置git配置去掉ssl校验。

```
git config --global http.sslVerify false
```

2. 从github拉取CogVideoX代码。

```
cd /home/ma-user
git clone https://github.com/THUDM/CogVideo.git
cd /home/ma-user/CogVideo
git checkout v1.0
```

3. 若进行训练微调需依赖decord和triton包，arm版本可参考附录安装编译。
4. 安装CogVideo Ascend软件包。

a. 将获取到的CogVideo Ascend软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径参见**获取软件和镜像**。

b. 解压AscendCloud-AIGC-*.zip文件，解压后将里面指定文件与对应CogVideo文件进行替换，执行以下命令即可。

```
cd /home/ma-user
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
cd AscendCloud/multimodal_algorithm/CogVideo_v1_sft/
dos2unix install.sh
bash install.sh
```

说明

AscendCloud-AIGC-*.zip后面的*表示时间戳，请按照实际替换。

CogVideo Ascend软件包内容如下：

```
.
|---- install.sh          安装torch-npu适配修改脚本
|---- modify.patch        适配CogVideo训练代码git patch文件
|---- README.md           适配文档基于官方代码commit id说明
|---- requirements.txt    python依赖包
|---- vae_cache.py        vae_cache文件
|---- vae_cache.sh        vae_cache脚本
```

步骤五：CogVideo 微调

1. **下载模型权重**

首先，前往SAT镜像下载模型权重。对于CogVideoX-2B模型，请按照如下方式下载。

```
mkdir CogVideoX-2b-sat
cd CogVideoX-2b-sat
wget https://cloud.tsinghua.edu.cn/f/fdba7608a49c463ba754/?dl=1
mv 'index.html?dl=1' vae.zip
unzip vae.zip
wget https://cloud.tsinghua.edu.cn/f/556a3e1329e74f1bac45/?dl=1
mv 'index.html?dl=1' transformer.zip
unzip transformer.zip
```

请按如下链接方式下载CogVideoX-5B模型的transformers文件（VAE 文件与 2B 相同）：**CogVideoX-5B**。

接着，需要将模型文件排版成如下格式。

```
.
├── transformer
│   ├── 1000 (or 1)
│   │   └── mp_rank_00_model_states.pt
│   └── latest
└── vae
    └── 3d-vae.pt
```


将对应模型下载至容器内。

```
#CogVideoX-2b模型地址
https://huggingface.co/THUDM/CogVideoX-2b
#CogVideoX-5b模型地址
https://huggingface.co/THUDM/CogVideoX-5b
```

克隆T5模型，该模型不用做训练和微调，但是必须使用。

```
5b模型参考如下，2b模型请修改对应路径
mkdir t5-v1_1-xxl
mv CogVideoX-5b/text_encoder/* CogVideoX-5b/tokenizer/* t5-v1_1-xxl
```

通过上述方案，将会得到一个safetensor格式的T5文件，确保在Deepspeed微调过程中读入的时候不会报错。

```
├── added_tokens.json
├── config.json
├── model-00001-of-00002.safetensors
├── model-00002-of-00002.safetensors
├── model.safetensors.index.json
├── special_tokens_map.json
├── spiece.model
└── tokenizer_config.json

0 directories, 8 files
```

2. **准备数据集**

   数据集格式应该如下：

```
.
├── labels
│   ├── 1.txt
│   ├── 2.txt
│   └── ...
└── videos
    ├── 1.mp4
    ├── 2.mp4
    └── ...
```

   每个 txt 与视频同名，为视频的标签。视频与标签应该一一对应。通常情况下，不使用一个视频对应多个标签。

   如果为风格微调，请准备至少50条风格相似的视频和标签，以利于拟合。

3. **修改CogVideo/sat/configs/cogvideox_*.yaml文件**

   如果希望使用 Lora 微调，需要修改cogvideox_<模型参数>_lora 文件，修改参考如下：

```
***
  conditioner_config:
    target: sgm.modules.GeneralConditioner
    params:
      emb_models:
        - is_trainable: false
          input_key: txt
          ucg_rate: 0.1
          target: sgm.modules.encoders.modules.FrozenT5Embedder
          params:
            model_dir: "t5-v1_1-xxl" # CogVideoX-5b/t5-v1_1-xxl 权重文件夹的绝对路径
            max_length: 226

  first_stage_config:
    target: vae_modules.autoencoder.VideoAutoencoderInferenceWrapper
    params:
      cp_size: 1
      ckpt_path: "CogVideoX-5b-sat/vae/3d-vae.pt" # CogVideoX-5b-sat/vae/3d-vae.pt文件夹的绝对路径
      ignore_keys: [ 'loss' ]
```

4. **修改CogVideo/sat/configs/sft.yaml配置文件**

支持Lora和全参数微调两种方式。请注意，两种微调方式都仅对transformer部分进行微调。不改动VAE部分，T5仅作为Encoder使用。请按照以下方式修改configs/sft.yaml(全量微调) 中的文件。

```
 # checkpoint_activations: True ## using gradient checkpointing (配置文件中的两个checkpoint_activations都需要设置为True)
 model_parallel_size: 1 # 模型并行大小
 experiment_name: lora-disney  # 实验名称(不要改动)
 mode: finetune # 模式(不要改动)
 load: "{your_CogVideoX-5b-sat_path}/transformer" ## Transformer 模型路径
 no_load_rng: True # 是否加载随机数种子
 train_iters: 500 # 训练迭代次数
 eval_iters: 1 # 验证迭代次数
 eval_interval: 300    # 验证间隔
 eval_batch_size: 1  # 验证集 batch size
 save: ckpts # 模型保存路径
 save_interval: 100 # 模型保存间隔
 log_interval: 1 # 日志输出间隔
 train_data: [ "your train data path" ] #训练集路径
 valid_data: [ "your val data path" ] # 训练集和验证集可以相同
 split: 1,0,0 # 训练集，验证集，测试集比例
 num_workers: 8 # 数据加载器的工作线程数
 force_train: True # 在加载checkpoint时允许missing keys (T5 和 VAE 单独加载)
 only_log_video_latents: True # 避免VAE decode带来的显存开销
 deepspeed:
   bf16:
     enabled: False # For CogVideoX-2B Turn to False and For CogVideoX-5B Turn to True
   fp16:
     enabled: True  # For CogVideoX-2B Turn to True and For CogVideoX-5B Turn to False
```

5. **修改运行脚本**

   编辑CogVideo/sat/finetune_multi_gpus.sh，选择配置文件。下面是两个例子:

   a. 如果您想使用CogVideoX-5B模型并使用Lora方案，需要修改finetune_multi_gpus.sh中参数参考如下:

   ```
   --base configs/cogvideox_5b_lora.yaml
   ```

   b. 如果您想使用CogVideoX-5B模型并使用全量微调方案，需要修改finetune_multi_gpus.sh中参数参考如下:

   ```
   --base configs/cogvideox_5b.yaml
   ```

6. **开始训练微调**

   修改CogVideo/sat/vae_cache.sh脚本，将vae模型部分进行预先缓存，也需要对vae_cache.sh脚本中参数修改参考如下：

   ```
   --base configs/cogvideox_5b.yaml
   ```

   运行微调脚本finetune_multi_gpus.sh

   ```
   cd /home/ma-user/CogVideo/sat/
   bash vae_cache.sh
   bash finetune_multi_gpus.sh
   ```

以上微调文档提示来自官方文档，有关可用微调脚本参数及其功能的全面文档，您可以参考**官方CogVideo文档**。

附：decord 和 triton 在 arm 版本安装参考

由于训练使用decord和triton的Python包没有arm版本，请自行进行编译安装，以下给出安装编译教程仅供参考：

- decord包编译

  可参考如下脚本：

  ```
  bash install_decord.sh # install_decord.sh内容如下
  ```

  install_decord.sh脚本内容如下：


```
cd /home/ma-user/
git config --global http.sslVerify false
git clone  --recursive https://github.com/dmlc/decord --depth 1
cd decord
mkdir build && cd build
cmake ..  -DCMAKE_BUILD_TYPE=Release -DFFMPEG_DIR:PATH="/usr/local/ffmpeg/"
make
cd ../python
### option 1: add python path to $PYTHONPATH, you will need to install numpy separately
echo "PYTHONPATH=$PYTHONPATH:/home/ma-user/decord/python" >> ~/.bashrc
source ~/.bashrc
### option 2: install with setuptools
python3 setup.py install --user
```

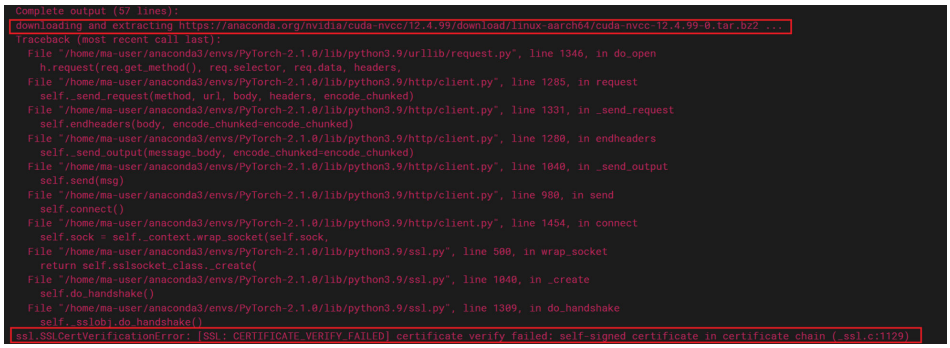
- triton包编译：

  本文档triton包基于commit id 8a3fb7e3fd6b87f09bcb4ebc6编译测试

```
cd /home/ma-user
git clone https://github.com/triton-lang/triton.git
cd /home/ma-user/triton
git checkout 8a3fb7e3fd6b87f09bcb4ebc6
cd /home/ma-user/triton/python
pip install ninja cmake wheel pybind11
pip install -e .
```

若编译过程出现所依赖的tar包下载失败，如下图所示：

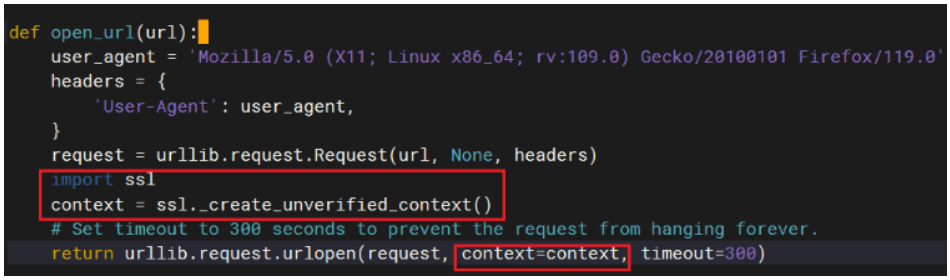
图 7-1 tar 包下载失败



可设置ssl忽略证书验证，修改/home/ma-user/triton/python/setup.py文件，open_url()方法：

```
# 新增ssl忽略证书验证
    import ssl
    context = ssl._create_unverified_context()
    # Set timeout to 300 seconds to prevent the request from hanging forever.
    return urllib.request.urlopen(request, context=context, timeout=300)
```

图 7-2 设置 ssl 忽略证书验证



7.3 Open-Sora1.2 基于 Lite Server 适配 PyTorch NPU 训练推理指导（6.3.910）

本文档主要介绍如何在ModelArts Lite Server上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora 1.2 训练和推理。

方案概览

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的Cann版本是cann_8.0.rc3。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.910版本，请参考表7-5获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 7-5 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.910软件包中的AscendCloud-AIGC-6.3.910-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910 版本。 **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 7-6 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.910版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	cann_8.0.rc3 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.910-xxx.zip文件，获取路径参见**表7-5**。本案例使用的是压缩包中的multimodal_algorithm/OpenSora1.2/，将OpenSora1.2整个文件夹上传到宿主机上。

Step3 构建标准镜像和容器环境

说明

Step3 构建标准镜像和容器环境 和 **Step4 构建与代码解耦的镜像和容器环境** 都是搭建容器环境，任选其中一个即可。

一、构建镜像

基于官方提供的基础镜像构建自定义镜像opensora1.2:1.0。参考如下命令在宿主机上编写Dockerfile文件。镜像地址{image_url}请参见**表7-6**。

镜像Dockerfile:

```
FROM {image_url}

COPY --chown=ma-user:ma-group OpenSora1.2/* /home/ma-user/OpenSora1.2/

RUN cd /home/ma-user/OpenSora1.2 && bash prepare.sh
WORKDIR /home/ma-user/OpenSora1.2
```

构建自定义镜像opensora1.2:1.0。

```
docker build -t opensora1.2:1.0 .
```

二、启动镜像

启动容器镜像，训练需要8卡，推理分为单卡推理和多卡推理，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 300g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7  --security-opt seccomp=unconfined --network=bridge -v ${work_dir}:${container_work_dir}  opensora1.2:1.0 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7，可根据需要选择挂载卡数。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

三、进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```


Step4 构建与代码解耦的镜像和容器环境

说明

Step3 构建标准镜像和容器环境 和 **Step4 构建与代码解耦的镜像和容器环境** 都是搭建容器环境，任选其中一个即可。

一、启动镜像

启动容器镜像，训练需要8卡，推理分为单卡推理和多卡推理，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 300g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7  --security-opt seccomp=unconfined --network=bridge -v ${work_dir}:${container_work_dir}  {image_url}  bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7，可根据需要选择挂载卡数。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。注意：将OpenSora1.2放入挂载目录下。
- {image_url}镜像地址请参见**表7-6**。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

二、进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

三、构建环境

需要将OpenSora1.2放入挂载目录下，container_work_dir 是挂载目录。并保证文件权限能被ma-user用户使用和修改。

```
cp -r {container_work_dir}/OpenSora1.2 /home/ma-user/
cd /home/ma-user/OpenSora1.2
bash prepare.sh
```

Step5 下载权重文件

建议手动下载所需的权重文件，保证文件权限能被ma-user用户使用和修改，在/home/ma-user/OpenSora1.2/目录下进行操作。

1. 创建文件夹存放不同的权重文件。
   ```
   mkdir weights
   ```
2. 下载 OpenSora-VAE-v1.2权重，将下载好的权重放在 ./weights 目录下。

   OpenSora-VAE-v1.2 官网下载地址：**https://huggingface.co/hpcai-tech/OpenSora-VAE-v1.2/tree/main**

3. 下载 OpenSora-STDiT-v3权重，将下载好的权重放在 ./weights 目录下。

   OpenSora-STDiT-v3 官网下载地址：**https://huggingface.co/hpcai-tech/OpenSora-STDiT-v3/tree/main**

4. 下载 t5-v1_1-xxl 权重，将下载好的权重放在 ./weights 目录下。

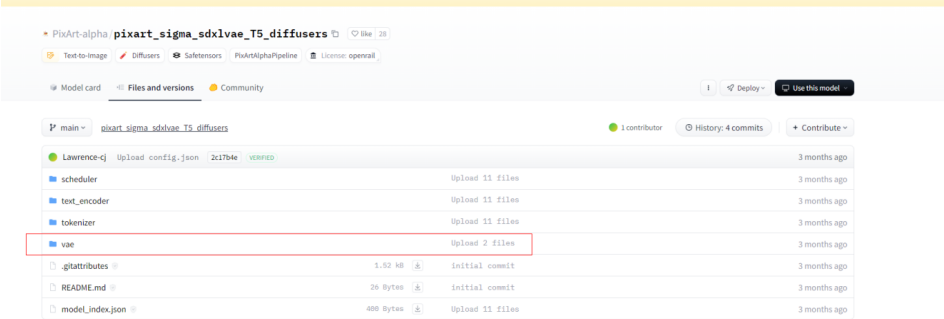
   t5-v1_1-xxl 官网下载地址： **https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**

5. 下载 pixart_sigma_sdxlvae_T5_diffusers的vae权重，将下载好的权重放在 ./weights 目录下。

   pixart_sigma_sdxlvae_T5_diffusers 官网下载地址：**https://huggingface.co/PixArt-alpha/pixart_sigma_sdxlvae_T5_diffusers/tree/main**

   下载下图中vae文件夹的内容。注意：本地下载文件时配置文件会变成vae_config.json，修改为config.json

   图 7-3 下载 vae 文件夹的内容

6. 下载vgg权重，将下载好的权重放在 ./weights 目录下。

   vgg16-397923af.pth 官网下载地址： **https://download.pytorch.org/models/vgg16-397923af.pth**

   vgg.pth 官网下载地址：**https://heibox.uni-heidelberg.de/f/607503859c864bc1b30b/?dl=1**

   将权重vgg16-397923af.pth复制到 /home/ma-user/.cache/torch/hub/checkpoints/下，这个文件夹需要自己创建。

   ```
   cp weights/vgg16-397923af.pth /home/ma-user/.cache/torch/hub/checkpoints/vgg16-397923af.pth
   ```

   下载完成的权重文件如下图所示：

   图 7-4 下载完成的权重文件

   ```
   [root@devserver-bms-1ba73d0f-l00572774 open-sora-1.2]# ls weights/
   OpenSora-STDiT-v3  OpenSora-VAE-v1.2  pixart_sigma_sdxlvae_T5_diffusers  t5-v1_1-xxl  vgg16-397923af.pth  vgg.pth
   ```

Step6 下载数据集

下载原始数据集。

```
cd /home/ma-user/OpenSora1.2/
mkdir -p datasets/webvid
wget https://anon-datasets.s3.amazonaws.com/results_2M_val.csv
python download_datasets.py
```

download_datasets.py的内容。

```
import os
import pandas as pd
```

```
for idx, row in pd.read_csv('results_2M_val.csv').iterrows():
    os.system(f"wget -O './datasets/webvid/{row['videoid']}.mp4' --no-check-certificate {row['contentUrl']}")
```

预处理数据。

```
python -m tools.datasets.convert video ./datasets/webvid --output ./datasets/webvid_meta.csv
python -m tools.datasets.datautil ./datasets/webvid_meta.csv --info --fmin 1
python merge_data.py
```

merge_data.py的内容。

```
import pandas as pd

merged_df = pd.merge(pd.read_csv('./datasets/webvid_meta_info_fmin1.csv'), pd.read_csv('./
results_2M_val.csv')[['videoid', 'name']], left_on='id', right_on='videoid', how='left')
merged_df = merged_df.rename(columns={'name': 'text'})
merged_df.to_csv('./datasets/webvid_meta_data.csv', index=False)
```

Step7 VAE 训练

vae训练分为3个阶段，后两次训练根据其前一次训练的结果继续训练。

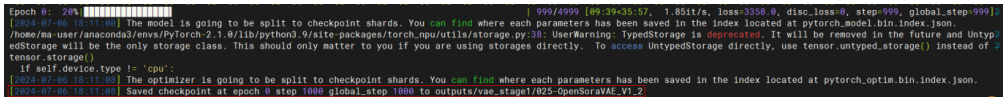
1. 第一阶段训练

   ```
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage1.py --data-path ./
   datasets/webvid_meta_data.csv
   ```

   训练完后的权重文件保存在 ./outputs/vae_stage1 文件夹下 （ 例如 ./outputs/vae_stage1/XXX-OpenSoraVAE_V1_2/epochX-global_step1000/model，X为按顺序自动生成的数字 ）

   具体位置打印在日志中，注意：输出文件夹是自动生成，只有日志中打印的位置才是保存权重的位置。

   图 7-5 VAE 第一阶段训练日志

2. 第二阶段训练

   ```
   export pretrain_path="上阶段训练的权重，例如./outputs/vae_stage1/XXX-OpenSoraVAE_V1_2/epochX-
   global_step1000/model"
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage2.py --data-path ./
   datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
   ```

   训练完后的权重文件保存在 ./outputs/vae_stage2 文件夹下 （ 例如 ./outputs/vae_stage2/XXX-OpenSoraVAE_V1_2/epochX-global_step1000/model，X为按顺序自动生成的数字 ）

3. 第三阶段训练

   ```
   export pretrain_path="上阶段训练的权重，例如./outputs/vae_stage2/XXX-OpenSoraVAE_V1_2/epochX-
   global_step1000/model"
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage3.py --data-path ./
   datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
   ```

   训练完后的权重文件保存在 ./outputs/vae_stage3 文件夹下 （ 例如 ./outputs/vae_stage3/XXX-OpenSoraVAE_V1_2/epochX-global_step1000/model，X为按顺序自动生成的数字 ）

Step8 DIT 训练

dit训练分为3个阶段，后两次训练根据其前一次训练的结果继续训练。

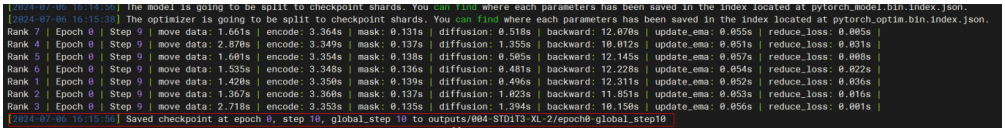
1. 第一阶段训练


```
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage1.py --data-path ./datasets/webvid_meta_data.csv
```

训练完后的权重文件保存在 ./outputs 文件夹下（例如 ./outputs/XXX-STDiT3-XL-2/epochX-global_step200/model，X为按顺序自动生成的数字）。

具体位置打印在日志中：

图 7-6 DIT 第一阶段训练日志



训练完成后在目录底下生成loss.txt文件(例如./outputs/XXX-STDiT3-XL-2/epochX-global_step200/model)记录每一步的loss。

2. 第二阶段训练

```
export pretrain_path="上阶段训练的权重，例如./outputs/XXX-STDiT3-XL-2/epochX-global_step200/model"
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage2.py --data-path ./datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
```

训练完后的权重文件保存在 ./outputs 文件夹下（例如 ./outputs/001-STDiT3-XL-2/epochX-global_step200/model，X为按顺序自动生成的数字），具体位置打印在日志中。

3. 第三阶段训练

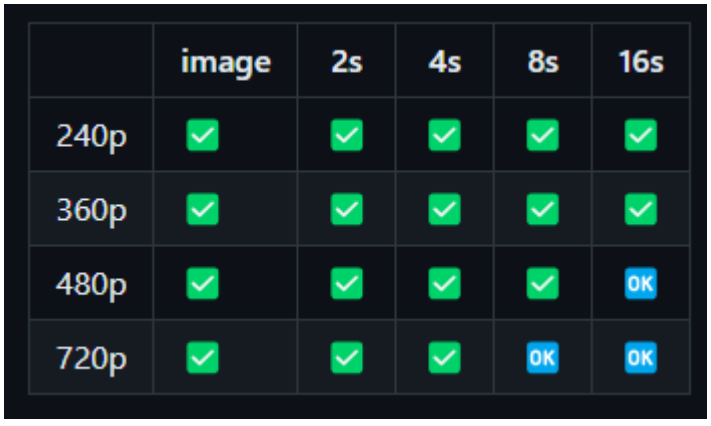
```
export pretrain_path="上阶段训练的权重，例如 ./outputs/XXX-STDiT3-XL-2/epochX-global_step200/model"
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage3.py --data-path ./datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
```

训练完后的权重文件保存在 ./outputs 文件夹下（例如 ./outputs/XXX-STDiT3-XL-2/epochX-global_step200/model，X为按顺序自动生成的数字），具体位置打印在日志中。

Step9 推理

对于大尺寸、长时间的视频强制需要多卡推理，具体要求见下图，绿色允许只用单卡推理，蓝色至少双卡推理。

图 7-7 推理视频要求



单卡推理

```
python inference.py configs/opensora-v1-2/inference/sample.py --num-frames 4s --resolution 720p --aspect-ratio 9:16 --prompt "a beautiful waterfall"
```

多卡推理

```
torchrun --nproc_per_node 2 inference.py configs/opensora-v1-2/inference/sample.py --num-frames 16s --resolution 720p --aspect-ratio 9:16 --prompt "a beautiful waterfall"
```

最终结果保存在samples/samples/sample_0000.mp4。

7.4 Open-Sora-Plan1.0 基于 Lite Server 适配 PyTorch NPU 训练推理指导（6.3.907）

本文档主要介绍如何在ModelArts Lite Server上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora-Plan1.0训练和推理。

方案概览

本方案目前仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

适配的Cann版本是cann_8.0.rc2。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表7-7**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

软件配套版本

表 7-7 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 7-8 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表7-7**。本案例使用的是解压到子目录 multimodal_algorithm/OpenSoraPlan1.0/ 目录下的所有文件，将该目录上传到宿主机上。

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像Open-Sora-Plan1.0:1.0。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表7-8**。

```
FROM {image_url}

COPY --chown=ma-user:ma-group OpenSoraPlan1.0/* /home/ma-user/Open-Sora-Plan1.0

RUN cd /home/ma-user/Open-Sora-Plan1.0 && bash prepare.sh

WORKDIR /home/ma-user/Open-Sora-Plan1.0
```

构建自定义镜像Open-Sora-Plan1.0:1.0。

```
docker build -t Open-Sora-Plan1.0:1.0 .
```

Step4 启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge Open-Sora-Plan1.0:1.0 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step5 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

Step6 安装 Decord

Decord是一个高性能的视频处理库，在昇腾环境中安装需要修改一些源码进行适配。

Decord建议安装在 /home/ma-user/lib中。

1. 安装x264

   ```
   mkdir /home/ma-user/lib && cd /home/ma-user/lib
   # X264 install
   git clone https://github.com/mirror/x264.git
   export PKG_CONFIG_PATH=/home/ma-user/lib/lib/pkgconfig
   cd x264
   ./configure --enable-shared --prefix=/home/ma-user/lib/
   make
   make install
   cd ..
   ```

2. 安装ffmpeg

   ```
   # ffmpeg install
   git clone https://github.com/FFmpeg/FFmpeg.git
   export TMPDIR=../tmp-ffmpeg
   cd FFmpeg
   ```



```
git checkout 78f55457c9be420f4109da45de42a36338d56aca
# git checkout release/4.0
./configure --enable-shared --enable-swscale --enable-gpl --enable-nonfree --enable-pic --prefix=/home/ma-user/lib/ --enable-version3 --enable-postproc --enable-pthreads --enable-static --enable-libx264
make
make install
cd ..
```

3. 安装decord

   a. 下载decord代码。

   ```
   git clone --recursive https://github.com/dmlc/decord
   cd decord
   ```

   b. 第一处修改

   ```
   vim src/video/ffmpeg/ffmpeg_common.h
   ```

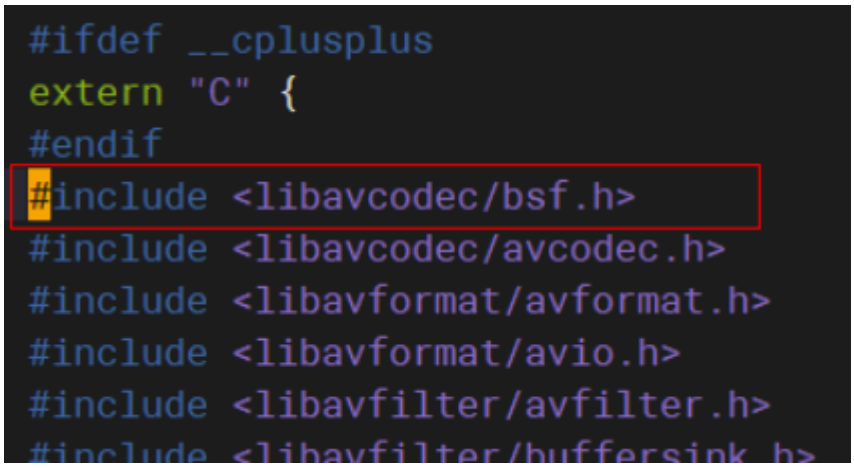
   在文件ffmpeg_common.h的23行，添加如下内容

   ```
   #include <libavcodec/bsf.h>
   ```

   图 7-8 文件 ffmpeg_common.h 修改前

   图 7-9 文件 ffmpeg_common.h 修改后

   c. 第二处修改：

```
vim src/video/video_reader.cc
```

在文件video_reader.cc的149行，进行修改：

```
    const AVCodec** dec_const = const_cast<const AVCodec**>(&dec);
    // int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1,
&dec, 0);
    int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1,
dec_const, 0);
```

图 7-10 文件 video_reader.cc 修改前

```
void VideoReader::SetVideoStream(int stream_nb) {
    if (!fmt_ctx_) return;
    AVCodec *dec;
    int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1, &dec, 0);
    // LOG(INFO) << "find best stream: " << st_nb;
    CHECK_GE(st_nb, 0) << "ERROR cannot find video stream with wanted index: " << stream_nb;
    // initialize the mem for codec context
    CHECK(codecs_[st_nb] == dec) << "Codecs of " << st_nb << " is NULL";
    // LOG(INFO) << "codecs of stream: " << codecs_[st_nb] << " name: " <<  codecs_[st_nb]->name;
```

图 7-11 文件 video_reader.cc 修改后

```
void VideoReader::SetVideoStream(int stream_nb) {
    if (!fmt_ctx_) return;
    AVCodec *dec;
    const AVCodec** dec_const = const_cast<const AVCodec**>(&dec);
    // int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1, &dec, 0);
    int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1, dec_const, 0);
    // LOG(INFO) << "find best stream: " << st_nb;
    CHECK_GE(st_nb, 0) << "ERROR cannot find video stream with wanted index: " << stream_nb;
    // initialize the mem for codec context
    CHECK(codecs_[st_nb] == dec) << "Codecs of " << st_nb << " is NULL";
    // LOG(INFO) << "codecs of stream: " << codecs_[st_nb] << " name: " <<  codecs_[st_nb]->name;
    ffmpeg::AVCodecParametersPtr codecpar;
```

d. 执行如下命令编译安装decord。

```
mkdir build && cd build
cmake .. -DUSE_CUDA=0 -DCMAKE_BUILD_TYPE=Release -DFFMPEG_DIR=/home/ma-user/lib/
make
cd ../python
python3 setup.py install --user
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/ma-user/lib/lib
echo "export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/home/ma-user/lib/lib" >> ~/.bashrc
```

如果重启docker后，环境变量需要重新配置，否则会报错找不到 libavformat.so.60: cannot open shared object file: No such file or directory

配置方式如下：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/ma-user/lib/lib
```

Step7 下载数据集

先创建文件夹用来存放数据集。

```
mkdir datasets
cd datasets
```

训练使用的开源数据集链接：https://huggingface.co/datasets/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main。

由于数据集比较大，可以自行选择部分数据集手动下载解压，并放入 ./datasets文件夹下。

例如：这里下载了上述链接中mixkit.tar.gz和sharegpt4v_path_cap_64x512x512.json。

（备注：如果只下载了部分数据集，需要对应修改sharegpt4v_path_cap_64x512x512.json文件）

解压数据集：

```
tar -xzvf mixkit.tar.gz
```

解压后的数据集结果如图所示。

图 7-12 解压后的数据集文件

```
total 26607068
drwxr-x--- 10 root root         147 Jul 30 15:54 mixkit
-rw-------  1 root root 27242973015 Jul 30 15:54 mixkit.tar.gz
-rw-------  1 root root     2659687 Jul 30 15:54 sharegpt4v_path_cap_64x512x512.json
```

Step8 下载权重文件

建议手动下载所需的权重文件，在/home/ma-user/Open-Sora-Plan1.0/目录下进行操作。

1. 创建文件夹存放不同的权重文件。

   ```
   mkdir weights
   mkdir weights_t5
   mkdir cache_dir
   ```

2. 下载基础模型权重**t2v.pt**放到cache_dir文件夹下。

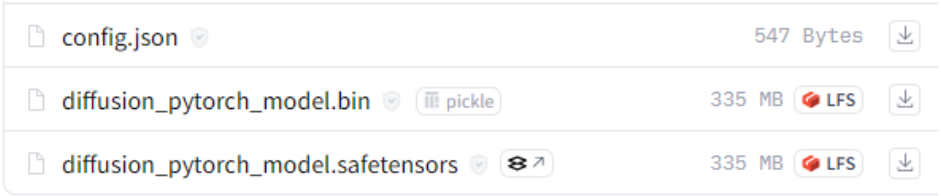
   ```
   mkdir Latte
   cd Latte
   git clone  -c http.sslVerify=false  https://huggingface.co/maxin-cn/Latte
   cd Latte
   git reset  --hard  83bdc71f7211963153464859d03d46d707e77865
   ```

   ```
   total 24
   -rw-r----- 1 root root 175 Aug 15 09:50 README.md
   -rw-r----- 1 root root 135 Aug 15 09:48 ffs.pt
   drwxr-x--- 2 root root  72 Aug 15 09:48 sd-vae-ft-ema
   drwxr-x--- 2 root root  72 Aug 15 09:48 sd-vae-ft-mse
   -rw-r----- 1 root root 135 Aug 15 09:48 skytimelapse.pt
   -rw-r----- 1 root root 135 Aug 15 09:50 t2v.pt
   -rw-r----- 1 root root 135 Aug 15 09:48 [illegible]nd.pt
   -rw-r----- 1 root root 135 Aug 15 09:48 ucf101.pt
   drwxr-x--- 2 root root  72 Aug 15 09:48 vae
   ```

   然后将该目录下的t2v.pt文件复制到/home/ma-user/Open-Sora-Plan1.0/cache_dir目录下。

3. 下载VAE权重**vae-ft-mse-840000-ema-pruned.ckpt**和配置文件**config.json**，放在weights文件夹下。

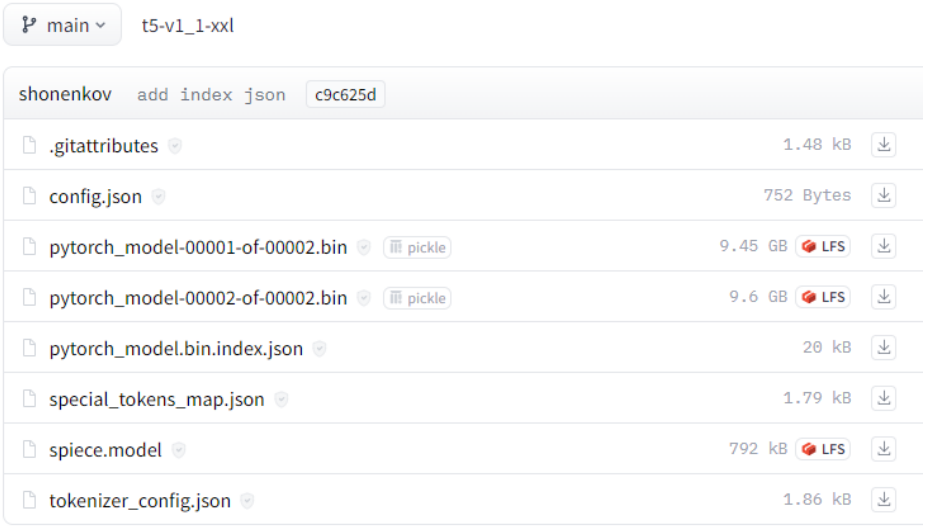
   下载链接：https://huggingface.co/stabilityai/sd-vae-ft-ema/tree/main

4. 下载**text_encoder**权重，放在weights_t5文件夹下。

   下载链接：**https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**，手动下载如**图7-13**所示文件，并放到weights_t5文件夹下

图 7-13 Huggingface 中 t5-v1_1-xxl 模型目录内容



Step9 启动训练服务

在/home/ma-user/Open-Sora-Plan1.0/目录下进行操作

训练至少需要单机8卡。

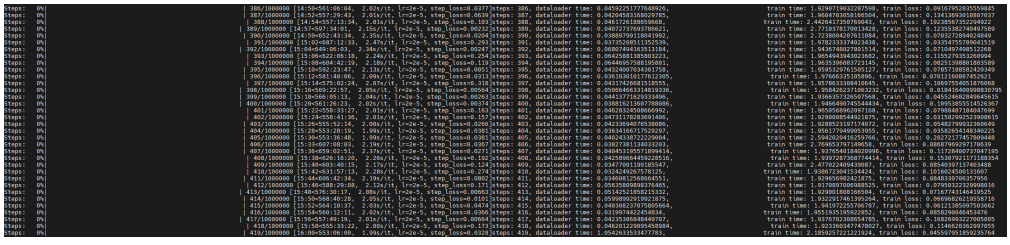
1. 命令启动训练脚本。

   例如：训练65帧的视频，拼接4张图片，则执行如下命令：

   ```
   bash train_videoae_65x512x512.sh
   ```

   正常训练过程如下图所示。训练完成后，关注loss值，loss曲线收敛，记录总耗时和单步耗时。训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

   图 7-14 正常训练过程

   训练完成后权重保存在自动生成的目录，例如：t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5/epoch1-global_step2000/checkpoint-2000/model。

Step10 推理部署

- **使用官方权重文件推理**

  官方权重下载链接：https://huggingface.co/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main

  创建文件夹存放官方权重文件，

  ```
  mkdir weights_inference
  cd weights_inference
  ```

```
mkdir vae
mkdir 65x512x512
```

以65x512x512为例：

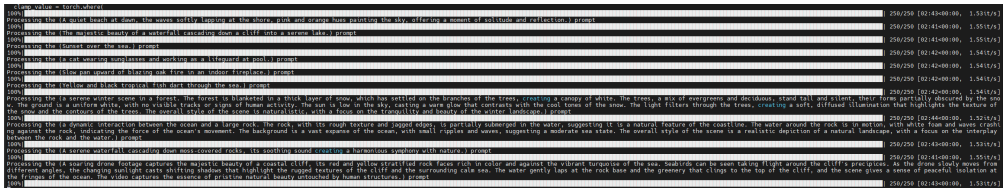
1，进入链接https://huggingface.co/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main

2，手动下载65x512x512目录下的权重文件diffusion_pytorch_model.safetensors和配置文件config.json，并放到weights_inference/65x512x512目录下

3，手动下载vae目录下的权重文件diffusion_pytorch_model.safetensors和配置文件config.json，并放到weights_inference/vae目录下。

4，执行如下命令使用官方权重推理。

```
bash sample_video_65.sh
```



- **使用训练生成的权重文件推理**

  在**Step7 启动训练服务**完成后，会在工作目录/home/ma-user/Open-Sora-Plan1.0/下自动生成一个t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5文件夹，训练后生成的权重文件存放在t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5文件夹中，例如t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5/010-F16S3-STDiT-XL-2/checkpoint-2000/model。

  修改推理配置文件中的权重文件sample_video_65.sh中的路径参数：--model_path，并执行如下推理命令。

  ```
  bash sample_video_65.sh
  ```

7.5 Open-Sora 1.0 基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.905）

本文档主要介绍如何在ModelArts Lite Server上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora训练和推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。训练至少需要单机8卡，推理需要单机单卡。

表 7-9 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

获取软件和镜像

表 7-10 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表7-10**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 本文档适配的是
- 训练至少需要单机8卡，推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**表7-10**。

```
docker pull {image_url}
```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练至少需要单机8卡，推理需要单机单卡。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
// 启动一个容器去运行镜像
docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 80g \
      --net=bridge \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小，建议不低于80GB。
- name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- ${image_name}：代表镜像地址。

说明

- 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

3. 进入容器。需要将${container_name}替换为实际的容器名称。
```
docker exec -it ${container_name} bash
```

说明

启动容器默认使用ma-user用户。后续所有命令执行也建议使用ma-user用户。

Step3 获取代码包并安装依赖

1. 下载插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，上传到容器的/home/ma-user/目录下，解压并安装相关依赖。获取路径参见**获取软件和镜像**。
```
mkdir -p /home/ma-user/ascendcloud-aigc-algorithm-open_sora    #创建目录
cd /home/ma-user/ascendcloud-aigc-algorithm-open_sora/         #进入目录

unzip -zxvf AscendCloud-3rdAIGC-6.3.905-*.zip
tar -zxvf ascendcloud-aigc-algorithm-open_sora.tar.gz
rm -rf AscendCloud-3rdAIGC-6.3.905-*
```
2. 安装Python环境。
```
pip install -r requirements.txt
cp attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
cp low_level_optim.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/colossalai/zero/low_level/low_level_optim.py
```

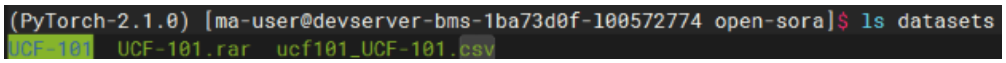
Step4 下载数据集

训练使用的开源数据集UCF101.rar，执行如下命令下载数据集并处理。数据集相关介绍参见**https://www.crcv.ucf.edu/data/UCF101.php**。

```
mkdir datasets
cd datasets
wget https://www.crcv.ucf.edu/data/UCF101/UCF101.rar
unrar x UCF101.rar
cd ..
python -m tools.datasets.convert_dataset ucf101 ./datasets/ --split UCF-101
mv ucf101_UCF-101.csv datasets/
```

处理完数据集后的结果如**图7-15**所示。

图 7-15 处理后的数据文件



Step5 启动训练服务

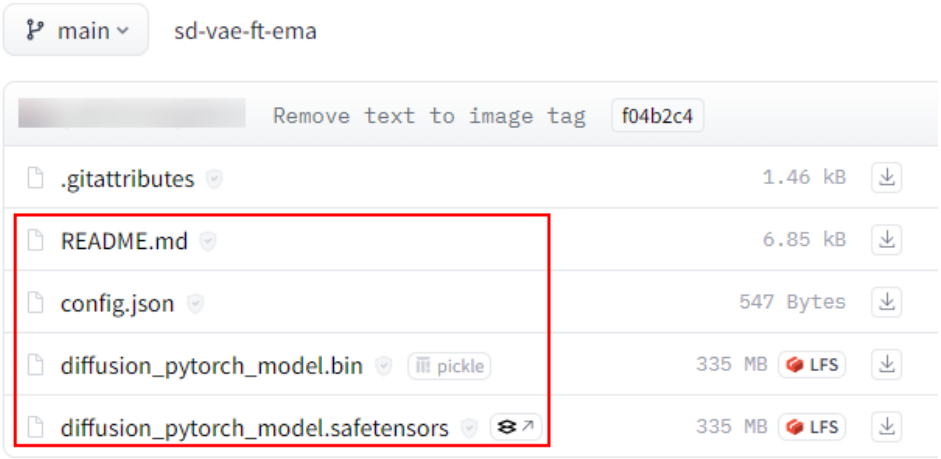
训练至少需要单机8卡。建议手动下载所需的权重文件，放在weights文件夹下。在/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录下进行操作。

1. 创建weights文件夹。
```
mkdir weights
```
2. 下载基础模型权重：PixArt-XL-2-512x512.pth和PixArt-XL-2-256x256.pth
```
cd weights
# 下载PixArt-XL-2-512x512.pth和PixArt-XL-2-256x256.pth
wget https://huggingface.co/PixArt-alpha/PixArt-alpha/resolve/main/PixArt-XL-2-512x512.pth
wget https://huggingface.co/PixArt-alpha/PixArt-alpha/resolve/main/PixArt-XL-2-256x256.pth
```
3. 下载VAE权重：sd-vae-ft-ema

   在weights文件夹下创建sd-vae-ft-ema文件夹。
```
mkdir sd-vae-ft-ema
```

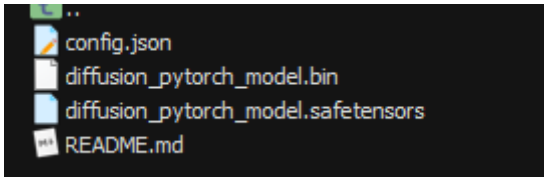
然后进入官网地址： **https://huggingface.co/stabilityai/sd-vae-ft-ema/tree/main**，手动下载如**图7-16**所示四个文件，并上传到服务器的/home/ma-user/ascendcloud-aigc-algorithm-open_sora/weights/sd-vae-ft-ema/目录下。

图 7-16 Huggingface 中 sd-vae-ft-ema 模型目录内容



上传完成后，weights/sd-vae-ft-ema/目录内容如**图7-17**所示。

图 7-17 服务器 weights/sd-vae-ft-ema/目录内容



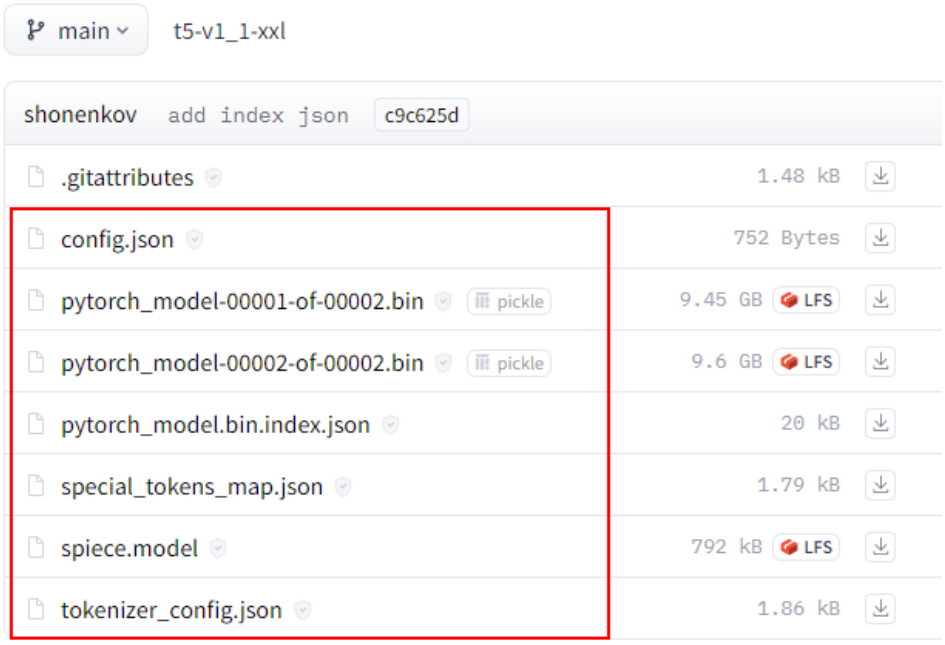
4. 下载Encoder模型权重：DeepFloyd/t5-v1_1-xxl

   在weights文件夹下创建t5-v1_1-xxl文件夹。

   ```
   mkdir t5-v1_1-xxl
   ```

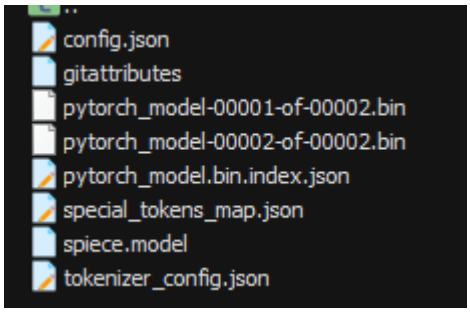
   然后进入官网地址 **https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**，手动下载如**图7-18**所示文件，并放到 /home/ma-user/ascendcloud-aigc-algorithm-open_sora/weights/t5-v1_1-xxl 文件夹下。

图 7-18 Huggingface 中 t5-v1_1-xxl 模型目录内容



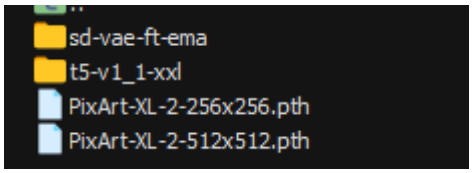
上传完成后，weights/t5-v1_1-xxl/目录下内容如图7-19所示。

图 7-19 服务器 weights/t5-v1_1-xxl/目录内容



最后weights文件夹下内容目录如图7-20所示。

图 7-20 服务器 weights 目录



从weights目录下返回到代码目录下。

```
cd ..
```

5. 在/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录下执行如下命令启动训练脚本。

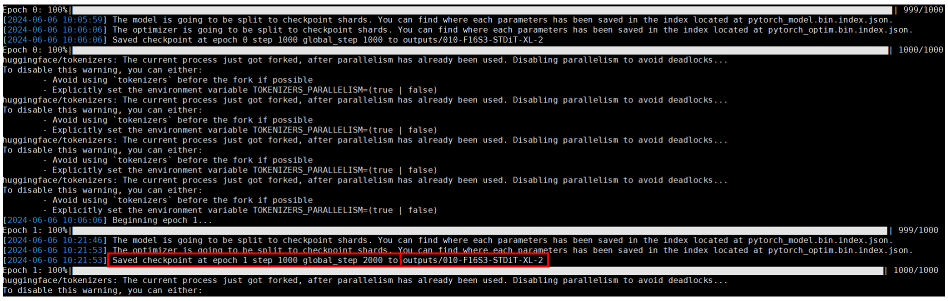
```
torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
```


正常训练过程如下图所示。训练完成后，关注loss值，loss曲线收敛，记录总耗时和单步耗时。训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

图 7-21 正常训练过程

训练完成后权重保存在自动生成的目录，例如：outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/。

图 7-22 训练完成后权重保存信息



Step6 推理

执行如下命令使用官方权重推理。推理脚本inference.py 会自动下载官方权重文件。

```
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```

说明

如果自动下载官方权重文件OpenSora-v1-HQ-16x512x512.pth失败，建议手动下载权重文件并上传到容器/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录中。

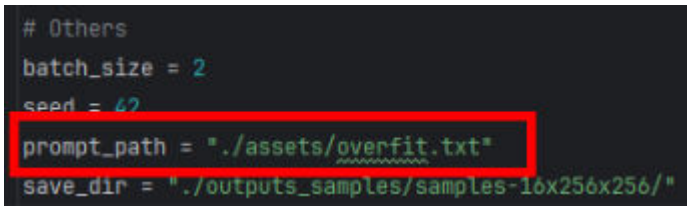
"OpenSora-v1-HQ-16x512x512.pth": "**https://huggingface.co/hpcai-tech/Open-Sora/resolve/main/OpenSora-v1-HQ-16x512x512.pth**"

执行如下命令使用训练后生成的权重推理。训练完成后会在工作目录/home/ma-user/ascendcloud-aigc-algorithm-open_sora/下自动生成一个outputs文件夹，训练后生成的权重文件存放在outputs文件夹中，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/。

```
export CKPT_PATH=./outputs/.../ #由训练日志中获得
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

如果要使用自己的prompt进行推理，可以修改用户自己推理脚本配置文件中prompt_path。例如在configs/opensora/inference/64x512x512.py配置文件中，使用了自己的prompt文件overfit.txt。

图 7-23 修改 prompt_path



Step7 精度对比

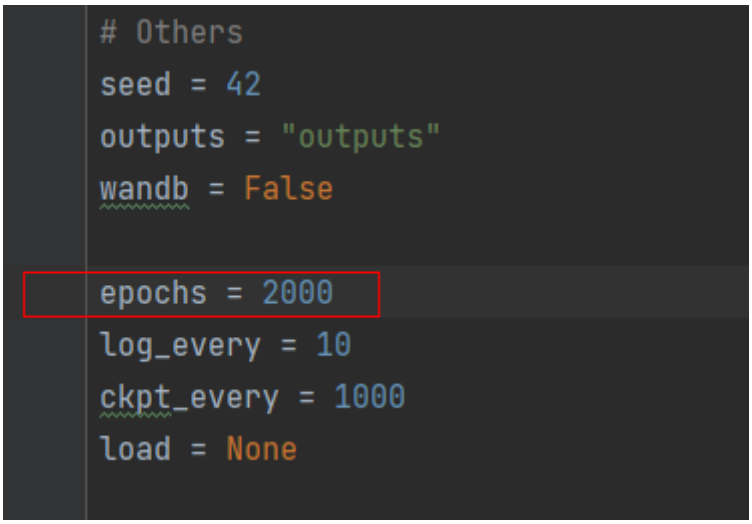
由于NPU和GPU生成的随机数不一样，需要固定二者的随机数再进行精度对比。通常的做法是先用GPU单卡跑一遍训练，生成固定下来的随机数。然后NPU和GPU都用固定的随机数进行单机8卡训练，比较精度。

1. 训练精度对齐。对齐前2000步的loss，观察loss在极小误差范围内。

   GPU环境下，使用Github中的官方代码跑训练任务。Github中的官方代码下载路径：**https://github.com/hpcaitech/Open-Sora/tree/v1.0.0**

   在NPU代码 configs/opensora/train/64x512x512.py中把 epochs = 200000 临时改成 epochs = 2000

   图 7-24 配置文件 64x512x512.py 修改训练步数

   将NPU代码中configs/opensora/train/64x512x512.py文件和configs/opensora/inference/64x512x512.py文件复制到GPU代码目录中，使用相同的参数配置文件。

   将NPU代码目录中的opensora/schedulers/iddpm/__init__.py文件和opensora/schedulers/iddpm/gaussian_diffusion.py文件复制到GPU代码目录中，添加固定随机数功能。

   进行GPU单机八卡训练，生成固定训练随机数，随机数会保存在noise文件夹中。

   ```
   mkdir noise_train  #创建文件夹noise_train，用于存放生成的随机数
   export LOCK_RAND=True  #是否固定随机数
   export SAVE_RAND=True  #是否保存生成的随机数
   export NOISE_PATH="./noise_train"  #将生成的随机数保存在"./noise_train"目录
   torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
   ```


说明

正常训练时不需要增加如下命令，只有训练精度对比时需要。

```
export LOCK_RAND=True   #是否固定随机数
export SAVE_RAND=True   #是否保存生成的随机数
export NOISE_PATH="./noise_train"    #将生成的随机数保存在"./noise_train"目录
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机8卡训练，比较在相应目录下生成的loss.txt文件。在NPU训练前，需要将上面GPU单机单卡训练生成的"./noise_train"文件夹移到NPU相同目录下。NPU和GPU的训练命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_train"
torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
```

GPU和NPU训练脚本中的参数要保持一致，除了参数dtype。NPU环境下，dtype="fp16"，GPU环境下，dtype="bf16"。

2. 基于NPU训练后的权重文件和GPU训练后的权重文件，对比推理精度。推理精度对齐流程和训练精度对齐流程相同，先在GPU固定推理的随机数。

```
mkdir noise_test1  #创建文件夹noise_test1，用于存放生成的随机数
export LOCK_RAND=True   #是否固定随机数
export SAVE_RAND=True    #是否保存生成的随机数
export NOISE_PATH="./noise_test1"    #将生成的随机数保存在"./noise_test1"目录
export CKPT_PATH=./outputs/.../ #由训练日志中获得，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机单卡推理，比较生成的视频是否一致。在NPU推理前，需要将上面GPU单机单卡推理生成的"./noise_test1"文件夹移到NPU相同目录下。NPU和GPU的推理命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_test1"
export CKPT_PATH=./outputs/.../ #由训练日志中获得，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

3. 基于官方权重文件分别在GPU和NPU进行推理，对比推理精度。推理精度对齐流程和训练精度对齐流程相同，先在GPU固定推理的随机数。

```
mkdir noise_test2  #创建文件夹noise_test2，用于存放生成的随机数
export LOCK_RAND=True  #是否固定随机数
export SAVE_RAND=True   #是否保存生成的随机数
export NOISE_PATH="./noise_test2"  #将生成的随机数保存在"./noise_test2"目录
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机单卡推理，比较生成的视频是否一致。在NPU推理前，需要将上面GPU单机单卡推理生成的"./noise_test2"文件夹移到NPU相同目录下。NPU和GPU的推理命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_test2"
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```


8 数字人模型训练推理

8.1 Wav2Lip 推理基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.907）

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源部署Wav2Lip模型用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B单机单卡。

表 8-1 环境要求

名称	版本
PyTorch	pytorch_2.1.0
驱动	23.0.6

获取软件和镜像

表 8-2 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907-xxx.zip软件包中的AscendCloud-AIGC-6.3.907-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表8-2**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表8-2**。

```
docker pull {image_url}
```

Step3 获取代码并上传

上传推理代码AscendCloud-AIGC-6.3.907-xxx.zip到宿主机的工作目录中，包获取路径请参见**表8-2**。

Step4 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -itd  --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step5 下载并适配代码

1. 在容器中解压代码包。

```
unzip AscendCloud-AIGC-6.3.907-*.zip
rm -rf AscendCloud-AIGC-6.3.907-*
```

2. 执行wav2lip推理插件的安装脚本。

```
cd multimodal_algorithm/Wav2Lip/inference/f361e9527b917a435928a10931fee9ac7be109cd
source install.sh
```


3. 从Github官网下载Wav2lip权重文件和Wav2Lip+GAN权重文件（**下载链接**），并放在容器的checkpoints目录下。上一步执行完source install.sh命令后，会自动生成checkpoints目录。

   图 8-1 下载权重文件

Model	Description	Link to the model
Wav2Lip	Highly accurate lip-sync	Link
Wav2Lip + GAN	Slightly inferior lip-sync, but better visual quality	Link

4. 从官网下载模型**s3fd-619a316812.pth**，并重命名为s3fd.pth，放在容器路径face_detection/detection/sfd下。上一步执行完source install.sh命令后，会自动生成face_detection/detection/sfd目录。

Step6 服务调用

1. 提前准备人物图片，支持'jpg', 'png', 'jpeg'格式。推荐测试图片大小1280*720或1920*1080。
2. 提前准备音频文件audio，支持'wav', 'mp3', 'mp4'格式。
3. 在代码根目录Wav2lip下创建test_wav2lip.sh, 复制以下内容粘贴至test_wav2lip.sh中，参数参照下方说明进行配置。

   ```
   #!/bin/bash
   start_time=$(date +%s)
   python inference.py --checkpoint_path <ckpt_path> --face <jpg_path> --audio <audio_path> --outfile <output_path>
   end_time=$(date +%s)
   execution_time=$((end_time - start_time))
   echo "wav2lip cost: $execution_time s"
   ```

   <ckpt_path>：模型权重路径 checkpoints/wav2lip.pth或 checkpoints/wav2lip_gan.pth。

   <jpg_path>: 人物图片路径，需要指定到具体的文件，例如 xxx/xxx.jpg。

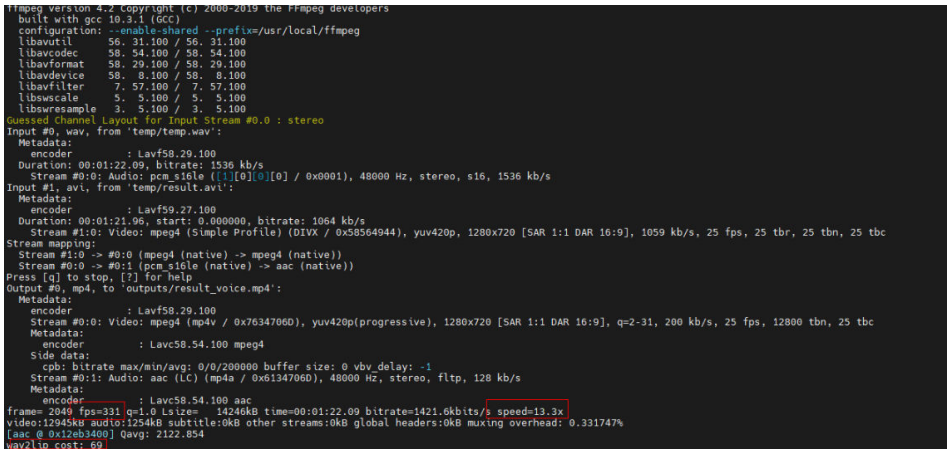
   <audio_path>：音频路径，需要指定到具体的文件，例如 xxx/xxx.mp4。

   <output_path>：视频结果输出路径，需要指定到具体的输出文件名，例如 xxx/xxx.mp4。

4. 执行test_wav2lip.sh脚本进行推理。

   ```
   cd Wav2Lip
   bash test_wav2lip.sh
   ```

图 8-2 输出日志截图



8.2 Wav2Lip 训练基于 Lite Server 适配 PyTorch NPU 训练指导（6.3.907）

本文档主要介绍如何在ModelArts Lite的Lite Server环境中，使用NPU卡训练Wav2Lip模型。本文档中提供的Wav2Lip模型，是在原生Wav2Lip代码基础上适配后的模型，可以用于NPU芯片训练。

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

Wav2Lip模型的输入为任意的一段视频和一段语音，输出为一段唇音同步的视频。

Wav2Lip的网络模型总体上分成三块：生成器、判别器和一个预训练好的唇音同步判别模型Pre-trained Lip-sync Expert。

- 生成器是基于encoder-decoder的网络结构，分别利用2个encoder（speech encoder和identity encoder）去对输入的语音和视频人脸进行编码，并将二者的编码结果进行拼接，送入到face decoder中进行解码得到输出的视频帧。
- 判别器Visual Quality Discriminator对生成结果的质量进行规范，提高生成视频的清晰度。
- 引入预训练的唇音同步判别模型Pre-trained Lip-sync Expert，作为衡量生成结果的唇音同步性的额外损失，可以更好的保证生成结果的唇音同步性。

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾计算资源开展Wav2Lip训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B单机单卡。

表 8-3 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 8-4 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907-xxx.zip软件包中的AscendCloud-AIGC-6.3.907-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表8-4**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查docker是否安装。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表8-4**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。
```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   -p 8080:8080 \
   --device=/dev/davinci0 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```
参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -p 8080:8080：开启一个端口，可以web访问（如冲突，可自行更换其他端口）。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。
```
docker exec -it ${container_name} bash
```

Step4 安装依赖和软件包

1. 从github拉取Wav2Lip代码。
```
cd /home/ma-user
git clone https://github.com/Rudrabha/Wav2Lip.git
cd /home/ma-user/Wav2Lip
git reset --hard f361e9527b917a435928a10
```
如果出现报错SSL certificate problem: self signed certificate in certificate chain

图 8-3 报错 SSL certificate problem

可采取忽略SSL证书验证：使用以下命令来克隆仓库，它将忽略SSL证书验证。
```
git clone -c http.sslVerify=false https://github.com/Rudrabha/Wav2Lip.git
```
2. 安装Wav2Lip Ascend软件包。
   a. 将获取到的Wav2Lip Ascend软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径：**Support网站**。
   b. 解压AscendCloud-AIGC-*.zip文件，解压后将里面指定文件与对应Wave2Lip文件进行替换。
```
cd  /home/ma-user
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
cp AscendCloud/multimodal_algorithm/Wav2Lip/train/f361e9527b917a435928a10/* /home/ma-user/Wav2Lip/
rm -rf AscendCloud*
```
> **说明**
>
> AscendCloud-AIGC-*.zip后面的*表示时间戳，请按照实际替换。

要替换的文件目录结构如下所示：
```
|---Wav2Lip_code/
      --- requirements.txt          #建议的依赖包版本

注：需要对以下文件进行修改
 --- color_syncnet_train.py     #训练expert discriminator唇形同步鉴别器
 --- wav2lip_train.py           #训练 Wav2Lip 模型
 --- preprocess.py              #对初始视频数据进行推理
      在以上三个文件内import末尾增加import如下：
        import torch_npu
        from torch_npu.contrib import transfer_to_npu
```
3. 安装Python依赖包，文件为requirements.txt文件。
```
pip install -r requirements.txt
```

Step5 训练 Wav2Lip 模型

1. 准备预训练模型。下载需要使用的预训练模型。
   - 人脸检测预训练模型，**下载链接**。
   - 专家唇形同步鉴别器，**下载链接** ，此链接是官方提供的预训练模型。训练Wav2Lip模型时需要使用专家唇形同步鉴别器，用户可以用自己的数据训练，也可以直接使用官方提供的预训练模型。
2. 处理初始视频数据集。

a. 将下载好的**人脸检测预训练模型**修改名字为s3fd.pth，上传到/home/ma-user/Wav2Lip/face_detection/detection/sfd/s3fd.pth目录。

b. 下载**LRS2数据集**。数据集文件夹结构如下：

```
├── LRS2_partly
|   ├── main
|   |   ├── five-digit numbered video IDs ending with (.mp4)
|   |   ├── 00001.mp4
|   |   ├── 00002.mp4
```

c. 对数据集进行预处理。具体命令如下。

```
python preprocess.py --data_root ./LRS2_partly --preprocessed_root lrs2_preprocessed/
```

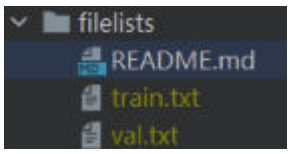
data_root参数为原始视频根目录，preprocessed_root参数为处理后生成的数据集目录。

处理后数据目录如下所示。

```
preprocessed_root (lrs2_preprocessed)
 ├── main
|   ├── Folders with five-digit numbered video IDs ( 00001 )
|   |   ├── *.jpg
|   |   ├── audio.wav
|   ├── 00001
|   |   ├── *.jpg
|   |   ├── audio.wav
```

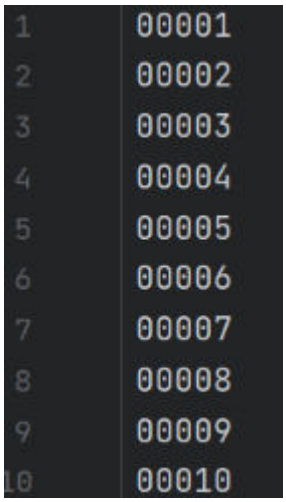
d. 将LRS2文件列表中的.txt文件（train、val）放入该filelists文件夹中。

图 8-4 filelists 文件夹



train.txt和val.txt内容参考如下，为处理后视频数据的目录名字。

图 8-5 train.txt 和 val.txt 内容



3. 训练专家唇形同步鉴别器。

如果使用LRS2数据集，可选择跳过此步骤。如果使用自己的数据集，训练命令参考如下。

```
python color_syncnet_train.py --data_root ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel/
syncnet_model/ --checkpoint_path ./checkpoints/lipsync_expert.pth
```

参数说明：

- --data_root： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
- --checkpoint_dir： 此目录用于保存模型。
- -checkpoint_path： （可选）可基于此目录的lipsync_expert模型继续进行训练，如果重新训练则不需要此参数。

默认每10000 step保存一次模型。

4. 训练Wav2Lip模型。

   训练Wav2Lip模型时需要使用专家唇形同步鉴别器。可以使用上一步**3**中的训练结果，也可以直接下载官方提供的**预训练权重**来使用。

   具体训练命令如下。

   ```
   python wav2lip_train.py --data_root  ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel --
   syncnet_checkpoint_path ./checkpoints/lipsync_expert.pth --checkpoint_path ./checkpoints/wav2lip.pth
   ```

   首次训练会进行模型评估，默认为700 step，请耐心等待，结束之后会进行正式训练。

   参数说明：

   - --data_root： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
   - --checkpoint_dir： 此目录用于保存模型。
   - --syncnet_checkpoint_path： 专家鉴别器的目录。
   - --checkpoint_path： （可选）可基于此目录的Wav2Lip模型继续进行训练，如果重新训练则不需要此参数。

   默认每3000 step保存一次模型。

   注：

   - 专家鉴别器的评估损失应降至约 0.25，Wav2Lip评估同步损失应降至约 0.2，以获得良好的结果。
   - 可以在文件设置其他不太常用的超参数hparams.py，常用超参如下：

     ```
     nepochs  训练总步数
     checkpoint_interval  Wav2Lip模型保存间隔步数
     eval_interval       Wav2Lip模型评估间隔步数
     syncnet_eval_interval          专家鉴别器模型评估间隔步数
     syncnet_checkpoint_interval     专家鉴别器模型保存间隔步数
     ```


9 内容审核模型训练推理

9.1 Bert 基于 Lite Server 适配 MindSpore Lite 推理指导 (6.3.910)

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾Atlas 300I Duo推理卡计算资源，部署Bert-base-chinese模型推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Lite Server资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Atlas 300I Duo。

表 9-1 资源规格要求

名称	版本
资源规格	Atlas 300I Duo
PyTorch	2.1.0
驱动	24.1.RC2.3
Python	3.9
CANN	8.0.RC3
MindSpore Lite	2.3.0
OS	arm

获取软件和镜像

表 9-2 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.910-xxx.zip软件包中的AscendCloud-CV-6.3.910-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**，在此路径中查找下载ModelArts 6.3.910 版本。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt3p-20240906180137-154bd1b	从SWR拉取。

约束限制

本文档适配昇腾云ModelArts 6.3.910版本，请参考**表9-2**的软件包和镜像，请严格遵照版本配套关系使用本文档。

确保容器可以访问公网。

准备容器环境

1. 请参考**Lite Server**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先释放被挂载的NPU或者联系华为方技术支持。

3. 检查驱动版本。

   运行如下命令查询驱动版本，回显信息中的“Software Version”字段值表示驱动版本。*NPU ID*表示设备编号，可通过**npu-smi info -l**命令查询。

   ```
   npu-smi info -t board -i NPU ID
   ```

   如果Atlas 300I Duo推理卡的驱动版本低于24.1.RC2.3，请参考**升级文档**升级驱动(24.1.RC2.3升级操作和24.1.RC2相同)，24.1.RC2.3驱动软件包获取地址参考**驱动软件包**。

4. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

5. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表9-2**。

```
docker pull {image_url}
```

启动容器镜像

启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -it --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7，可根据需要选择挂载卡数。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。

driver及npu-smi需同时挂载至容器。

不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

获取代码并上传

上传推理代码AscendCloud-CV-6.3.910-xxx.zip到宿主机的工作目录中，包获取路径请参见表2。

上传代码到宿主机时使用的是root用户，此处需要在容器中执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

准备推理环境

1. 安装transformers，用于转换模型和推理。
   ```
   pip install transformers==4.45.2
   ```
2. 获取推理代码。
   ```
   cd ${container_work_dir}
   unzip AscendCloud-CV-6.3.910-*.zip
   cd Bert/bert_infer/mindspore_lite
   ```
3. 获取bert-base-chinese模型文件。
   ```
   mkdir bert-base-chinese
   wget -P bert-base-chinese https://huggingface.co/google-bert/bert-base-chinese/resolve/main/pytorch_model.bin
   wget -P bert-base-chinese https://huggingface.co/google-bert/bert-base-chinese/resolve/main/config.json
   wget -P bert-base-chinese https://huggingface.co/google-bert/bert-base-chinese/resolve/main/tokenizer.json
   wget -P bert-base-chinese https://huggingface.co/google-bert/bert-base-chinese/resolve/main/tokenizer_config.json
   wget -P bert-base-chinese https://huggingface.co/google-bert/bert-base-chinese/resolve/main/vocab.txt
   ```
4. pt模型转onnx模型。
   ```
   python pth2onnx.py ./bert-base-chinese/ ./bert_model.onnx
   python modify_onnx.py ./bert_model.onnx
   ```

   说明

   该转换脚本用于Fill-Mask 任务，若是其他类型任务请按实际场景修改转换脚本。

5. onnx模型转mindir格式，执行如下命令，转换完成后会生成bert_model.mindir文件。
   ```
   converter_lite --fmk=ONNX --modelFile=bert_model.onnx --outputFile=bert_model --inputShape='input_ids:1,512;attention_mask:1,512;token_type_ids:1,512' --saveType=MINDIR --optimize=ascend_oriented
   ```

   动态seq_len场景下需要创建转换配置文件convert_config.ini，将如下内容写入配置文件：

   ```
   [acl_build_options]
   input_format="ND"
   input_shape="input_ids:1,-1;attention_mask:1,-1;token_type_ids:1,-1"
   ge.dynamicDims="50,50,50;100,100,100;150,150,150;200,200,200;250,250,250;300,300,300;350,350,350;400,400,400;450,450,450;512,512,512"
   ```


其中input_shape中的-1表示设置动态seq_len，ge.dynamicDims表示支持的seq_len值，可根据实际业务场景选取要支持的seq_len，上面的配置表示模型的三个输入shape支持[1, seq_len]，seq_len取值[50,100, ⋯,450,512]。另外需要注意seq_len不能超过模型支持的最大值，本文中下载的bert模型seq_len最大支持512。关于动态batch配置说明详见：**https://www.mindspore.cn/lite/docs/zh-CN/r2.3.0/use/cloud_infer/converter_tool_ascend.html**

使用如下转换命令：

```
converter_lite --fmk=ONNX --modelFile=bert_model.onnx --outputFile=bert_model_dy --saveType=MINDIR --optimize=ascend_oriented --configFile=convert_config.ini
```

说明

使用converter_lite转换模型时，如果报E10001: Value [linux] for parameter [--host_env_os] is invalid. Reason: os not supported, support setting are the OS types of opp package。

建议在**启动容器镜像**中通过docker run启动容器时，加上--privileged=true参数。

开始推理

执行如下命令开始推理。

```
python infer.py --tokennizer_config_path ./bert-base-chinese/ --mindir_model_path bert_model_dy.mindir --onnx_model_path bert_model.onnx --input_text [MASK]京是中国的[MASK]都。
```

infer.py是NPU上使用MindSpore Lite推理的样例，不同业务场景需根据实际情况做相应修改。infer.py文件预置在AscendCloud-CV-6.3.910-xxx.zip软件包中。

infer.py中包含使用MindSpore Lite在NPU上推理和使用推理onnxruntime在CPU上推理，结果如下图，按顺序展示[MASK]位置最大概率填充的文字。

```
mslite infer start
mslite infer result:  北 首
onnx infer start
onnx infer result:  北 首
```

如果是静态seq_len推理，修改infer脚本中45行max_length 的值为静态seq_len，并屏蔽或者删除25~26行以及46~49行，如下图所示。

```python
def mslite_infer(model, input_data):
    ms_inp = list(input_data.values())
    inputs = model.get_inputs()
    # shapes = [ms_inp[0].shape, ms_inp[1].shape, ms_inp[2].shape]
    # model.resize(inputs, shapes)

    for i, _input in enumerate(inputs):
        _input.set_data_from_numpy(ms_inp[i])

    outputs = model.predict(inputs)
    return outputs[0].get_data_to_numpy()

def main(args):
    #加载model 和 tokenizer
    tokenizer = AutoTokenizer.from_pretrained(args.tokennizer_config_path)
    model = mslite_init_model(args.mindir_model_path)

    input_text = args.input_text

    """len_range取值为使用converter_lite转换mindir模型时指定的动态维度,
       如果为静态seq_len,则直接指定max_length为支持的seq_len,但要保证输入的text长度不能超过max_length
    """
    len_range = [50,100,150,200,250,300,350,400,450,512]
    max_length = 50
    # for i in len_range:
    #     if len(input_text) < i:
    #         max_length = i
    #         break
```

9.2 Yolov8 基于 Lite Server 适配 MindSpore Lite 推理指导（6.3.909）

方案概览

本方案介绍了在ModelArts的Lite Server上使用昇腾Atlas 300I Duo推理卡计算资源，部署Yolov8 Detection模型推理的详细过程。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Atlas 300I Duo。

表 9-3 资源规格要求

名称	版本
资源规格	Atlas 300I Duo
PyTorch	2.1.0
驱动	24.1.RC2.3
Python	3.9
CANN	8.0.RC3
MindSpore Lite	2.3.0

名称	版本
OS	arm
ultralytics	8.2.70

获取软件和镜像

表 9-4 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.909-xxx.zip软件包中的AscendCloud-CV-6.3.909-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2406-aarch64-snt3p-20240906180137-154bd1b	从SWR拉取。

约束限制

本文档适配昇腾云ModelArts 6.3.909版本，请参考**表9-4**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

确保容器可以访问公网。

步骤一：准备环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先释放被挂载的NPU或者联系华为方技术支持。

3. 检查驱动版本。

   运行如下命令查询驱动版本，回显信息中的“Software Version”字段值表示驱动版本。*NPU ID*表示设备编号，可通过**npu-smi info -l**命令查询。

```
npu-smi info -t board -i NPU ID
```

如果Atlas 300I Duo推理卡的驱动版本低于24.1.RC2.3，请参考**升级文档**升级驱动(24.1.RC2.3升级操作和24.1.RC2相同)，24.1.RC2.3驱动软件包获取地址参考**驱动软件包**。

4. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```

5. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

```
sysctl -p | grep net.ipv4.ip_forward
```

如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

步骤二：获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见表2。

```
docker pull {image_url}
```

步骤三：启动容器镜像

启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -it --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。

driver及npu-smi需同时挂载至容器。

不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

步骤四：获取代码并上传

上传推理代码AscendCloud-CV-6.3.909-xxx.zip到宿主机的工作目录中，包获取路径请参见表2。

上传代码到宿主机时使用的是root用户，此处需要在容器中执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

步骤五：准备推理环境

1. 安装ultralytics，用于转换模型。
   ```
   pip install ultralytics==8.2.70 numpy==1.23.0 onnxslim==0.1.34
   ```
2. 获取推理代码。
   ```
   cd ${container_work_dir}
   unzip AscendCloud-CV-6.3.909-xxx.zip
   cd Yolov8/yolov8_infer/mindspore_lite
   ```
3. 获取yolov8 detection pt模型文件。下载地址：**https://github.com/autogyro/yolo-V8**

   图 9-1 下载 yolov8 detection pt 模型文件

   ▼ Detection (COCO)

   See Detection Docs for usage examples with these models trained on COCO, which include 80 pre-trained classes.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

4. pt模型转onnx模型。以转换yolov8n.pt为例，执行如下命令，执行完会在当前目录生成yolov8n.onnx文件。
   ```
   python pt2onnx.py --pt yolov8n.pt
   ```
5. onnx模型转mindir格式，执行如下命令，转换完成后会生成yolov8n.mindir文件。
   ```
   converter_lite --fmk=ONNX --modelFile=yolov8n.onnx --outputFile=yolov8n --
   inputShape='images:1,3,640,640' --saveType=MINDIR --optimize=ascend_oriented
   ```
   如果要使用动态batch，使用如下转换命令


```
converter_lite --fmk=ONNX --modelFile=yolov8n.onnx --outputFile=yolov8n_dy --saveType=MINDIR --optimize=ascend_oriented --configFile=convert_config.ini
```

配置文件convert_config.ini的内容如下：

```
[acl_build_options]
input_format="ND"
input_shape="images:-1,3,640,640"
ge.dynamicDims="1;8;16"
```

其中input_shape中的-1表示设置动态batch，ge.dynamicDims表示支持的batch值，上面的配置表示输入模型shape支持[1,3,640,640]，[8,3,640,640]，[16,3,640,640]这三种。

关于动态batch配置说明详见：**https://www.mindspore.cn/lite/docs/zh-CN/r2.3.0/use/cloud_infer/converter_tool_ascend.html**

说明

使用converter_lite转换模型时，如果报E10001: Value [linux] for parameter [--host_env_os] is invalid. Reason: os not supported, support setting are the OS types of opp package。

建议在**步骤三：启动容器镜像**中通过docker run启动容器时，加上--privileged=true参数。

步骤六：开始推理

执行如下命令开始推理，推理完成后会生产*_result.jpg，即检测结果。

```
python infer.py --model yolov8n.mindir
```

infer.py是NPU上使用MindSpore Lite推理的样例，与GPU推理代码区别主要参考infer函数，不同业务场景需根据实际情况做相应修改。infer.py文件预置在AscendCloud-CV-6.3.909-xxx.zip软件包中。

模型每次推理的图片数量必须是支持的batchsize，比如当前转换的mindir模型batchsize仅支持1，那么模型推理输入的图片数只能是1张；如果当前转换的mindir模型的batchsize支持多个，比如1，2，4，8，那么模型推理输入的图片数可以是1，2，4，8。

如果使用动态batch模型，需要将infer.py中如下图红框中的两行代码取消注释。

图 9-2 修改 infer.py

```
def infer(model_path, input):

    # init context, and set target is ascend
    context = mslite.Context()
    context.target = ["ascend"]
    context.ascend.device_id = 0
    context.cpu.thread_num = 1
    context.cpu.thread_affinity_mode=2

    # build model from file
    model = mslite.Model()
    model.build_from_file(model_path, mslite.ModelType.MINDIR, context)

    #当使用动态batch模型时，需要根据实际输入shape,resize模型输入shape
    #shapes =[input.shape]
    #model.resize(model.get_inputs(), shapes)

    inputs = model.get_inputs()
    inputs[0].set_data_from_numpy(input)

    # execute inference
    outputs = model.predict(inputs)[0].get_data_to_numpy()

    return outputs
```

9.3 Paraformer 基于 Lite Server 适配 PyTorch NPU 推理指导（6.3.911）

方案概览

本方案介绍了在ModelArts Lite Lite Server上使用昇腾计算资源Ascend Snt9B开展Paraformer的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.911版本，请参考获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的Lite Server资源和Ascend Snt9B。

获取软件和镜像

表 9-5 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.911软件包中的AscendCloud-CV-6.3.911-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E**，登录后在此路径中查找下载ModelArts 6.3.911 版本。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc3-py_3.9-hce_2.0.2409-aarch64-snt9b-20241112192643-c45ac6b	SWR上拉取。

表 9-6 资源规格要求

模型	版本
CANN	cann_8.0.rc3
驱动	23.0.6

模型	版本
PyTorch	2.1.0

步骤一：检查环境

1. 请参考**Lite Server资源开通**，购买Lite Server资源，并确保机器已开通，密码已获取，能通过SSH登录，不同机器之间网络互通。

   说明

   购买Lite Server资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

   当容器需要提供服务给多个用户，或者多个用户共享使用该容器时，应限制容器访问Openstack的管理地址（169.254.169.254），以防止容器获取宿主机的元数据。具体操作请参见**禁止容器获取宿主机元数据**。

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

3. 检查是否安装docker。

   ```
   docker -v   #检查docker是否安装
   ```

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。

   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```

   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤二：获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表9-5**。

```
docker pull {image_url}
```

步骤三：启动容器镜像

启动容器镜像，启动前可以根据实际需要增加修改参数。

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
```



```
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

- --device=/dev/davinciX 挂载NPU设备，示例中挂载了8张卡，可根据需要选择挂载卡数
- work_dir：工作目录，目录下存放着训练所需代码、数据等文件
- container_work_dir: 容器工作目录，一般同work_dir
- container_name：自定义容器名
- image_id：镜像ID，通过docker images来查看拉取的镜像ID。

步骤四：进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it  ${container_name} bash
```

修改权限。

```
sudo chown -R ma-user:ma-group ${container_work_dir}
```

此步骤可能需要密码或root权限。

步骤五：下载代码及安装环境

下载华为侧插件代码包AscendCloud-CV-6.3.911-xxx.zip文件，获取路径参见**表9-5**。

```
unzip AscendCloud-CV-6.3.911-*.zip
#解压后，进到指定目录
cd Paraformer/paraformer_infer/torch_npu
#安装三方库
pip install funasr==1.1.12 torchaudio==2.1.0 tqdm==4.66.6
```

步骤六：下载模型参数

下载模型参数，下载全部文件，并根据以下目录结构存放。

```
torch_npu
├── infer.py
├── speech_paraformer-large-vad-punc_asr_nat-zh-cn-16k-common-vocab8404-pytorch
│   ├── example
│   ├── fig
│   ├── .DS_Store
│   ├── .gitattributes
│   ├── am.mvn
│   ├── config.yaml
│   ├── configuration.json
│   ├── model.pt
│   ├── README.md
│   ├── seg_dict
│   └── tokens.json
```

步骤七：下载 Aishell1 数据集

下载Aishell1数据集。

图 9-3 Aishell-1 数据集

解压后，存放的目录结构如下：

```
torch_npu
├── infer.py
├── speech_paraformer-large-vad-punc_asr_nat-zh-cn-16k-common-vocab8404-pytorch
├── speech_asr_aishell1_testsets
│   ├── wav
│   ├──transcript
```

在torch_npu目录下制作aishell.scp文件：

```
find $(pwd)/speech_asr_aishell1_testsets/wav/test -name "*.wav" | awk -F "[/ .]" '{print $(NF-1) " " $0}' >> aishell.scp
```

在torch_npu目录下制作label.txt文件：

```
wget https://www.modelscope.cn/datasets/modelscope/speech_asr_aishell1_testsets/resolve/master/aishell1_test.csv
sed -i '1d' aishell1_test.csv
sed -i 's/,/ /g' aishell1_test.csv
cat aishell1_test.csv | awk -F "/" '{print $5}' | sed 's/.wav/ /g' >> label.txt
```

注意

如果aishell1_test.csv发生变化，请根据实际情况调整命令。

步骤八：执行推理脚本

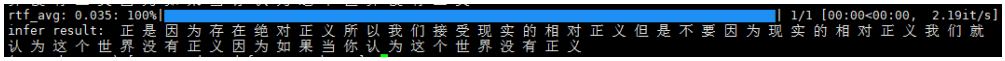
```
python infer.py --model_path 模型文件所在的绝对路径 --input_file 测试音频所在路径
```

参数说明：

- --model_path：为模型所在文件夹的绝对路径
- --input_file：输入音频，相关格式说明参考**文档**。

测试音频speech_paraformer-large-vad-punc_asr_nat-zh-cn-16k-common-vocab8404-pytorch/example/asr_example.wav的识别结果如下：

图 9-4 测试音频识别结果



步骤九：在 Aishell1 测试集上测试

```
python infer.py --model_path 模型文件所在的绝对路径 --input_file aishell.scp
```

执行完生成推理结果文件infer_result.txt。

10 GPU 业务迁移至昇腾训练推理

10.1 ModelArts 昇腾迁移调优工具总览

ModelArts集成了多个昇腾迁移调优工具，方便您在ModelArts平台环境中进行训练推理迁移、精度调试、性能调优等工作，您可在下表中查看当前ModelArts支持的昇腾迁移调优工具及对应指导。

表格中的部分工具已集成到ModelArts基础镜像中（镜像地址详见**基础镜像**章节）。如果您使用的是ModelArts基础镜像，可先尝试直接使用工具命令，如果相关命令不存在则需要参考工具安装指导自行安装。

表 10-1 ModelArts 昇腾迁移调优工具总览表

使用场景	类别	工具名称	工具描述	工具安装	使用指导
PyTorch GPU训练迁移至PyTorch NPU训练	训练迁移	Transfer2NPU	**代码自动迁移工具**，通过简单import命令可将PyTorch训练脚本从GPU平台迁移至NPU平台运行。	包含在torch_npu包中。	• **自动迁移工具使用指导** • **训练业务代码适配昇腾PyTorch代码适配**
		PyTorch Analyse	**迁移分析工具**，可以使用工具扫描用户的训练脚本，识别出源码中不支持的torch API和cuda API信息。	包含在cann toolkit中。	**分析工具使用指导**

使用场景	类别	工具名称	工具描述	工具安装	使用指导
	精度调优	msprobe	msprobe是MindStudio Training Tools工具链下精度调试部分的工具包。主要包括精度预检、溢出检测和精度比对等功能，目前适配PyTorch和MindSpore框架。这些子工具侧重不同的训练场景，可以定位模型训练中的精度问题。 • 支持精度预检，可扫描训练模型中的所有API进行API复现，给出精度情况的诊断和分析。 • 精度比对，对PyTorch整网API粒度的数据dump、精度比对，进而定位训练场景下的精度问题 • 支持溢出检测功能，判断是否存在输入正常但输出存在溢出的API，从而判断是否为正常溢出。 • 梯度状态监控，用于采集梯度数据并进行梯度相似度比对，可以精准定位出现问题的step。	执行pip install mindstudio-probe	msprobe使用手册
	性能调优	PyTorch Profiler	**性能采集工具**，在训练脚本中调用Ascend PyTorch Profiler接口，可在训练过程中采集性能数据文件，包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等。	包含在torch_npu包中。	Ascend PyTorch Profiler数据采集与分析

使用场景	类别	工具名称	工具描述	工具安装	使用指导
		MA-Advisor	**性能自动诊断工具**，采集好的Profiling数据通过该工具进行自动扫描分析，可给出性能瓶颈的诊断和修改建议。当迁移开箱性能较低时，通过该工具给出的建议修改代码后，通常可提升10%~30%。	执行pip install msprof-analyze	**昇腾性能自动诊断工具使用说明**
		compare_tools	**性能比对工具**，将在GPU和NPU采集的Profiling数据进行性能拆解和分类比对，展示算子、通信、内存等类别的性能比对数据。	下载工具**源码**使用。	**性能比对工具**
		cluster_analyse	**集群性能分析工具**，采集好的多机Profiling数据可通过该工具分析集群通信耗时、通信带宽矩阵等内容，从而辅助定位慢卡、慢节点等问题。工具的输出数据为csv格式，可直接拖入Ascend Insight进行可视化查看。	下载工具**源码**使用。	**集群分析工具**
		MindStudio-Insight	**性能可视化工具**，采集好的profiling数据可通过该工具进行可视化展示，辅助人工进行profiling数据查看和分析。	windows版本工具，下载链接见教程内。	**MindStudio-Insight用户指南**
PyTorch GPU推理迁移至MindSpore Lite NPU推理	模型迁移	Tailor	Mindspore-lite模型转换、精度误差分析、性能分析。	whl包，地址见教程中下载链接。	**Tailor使用指导**
	性能调优	msprof	msprof命令行工具提供了AI任务运行性能数据、昇腾AI处理器系统数据等性能数据的采集和解析能力。	包含在cann toolkit中。	**msprof**

使用场景	类别	工具名称	工具描述	工具安装	使用指导
		AOE	自动调优工具，提供子图调优和算子调优功能，在静态shape场景下有较好的调优效果。推荐在mindspore-lite离线推理场景下使用。	包含在cann toolkit中。	**AOE性能自动调优**
		AKG	MindSpore自动调优工具，提供算子自动优化和算子自动融合的功能，推荐在mindspore-lite离线推理场景下使用。	下载工具**源码**使用。	**AKG**
PyTorch GPU推理迁移至PyTorch ascend-vllm /atb/ torchair推理	模型迁移	-	需要用户自行代码适配，或者使用ModelArts迁移好的模型。	-	ModelArts迁移好的模型可参考最佳实践中的案例，使用AscendCloud软件包中的模型，例如：**主流开源大模型基于DevServer适配PyTorch NPU推理指导**。
	模型量化	modelslim	**模型量化工具**，通过量化提升模型的推理性能。	包含在cann toolkit中。	**ModelSlim**
	精度调试	msitllm	**大模型精度调试工具，**支持加速库（atb）和torchair的大模型推理的精度数据dump及比对功能，辅助大模型推理精度问题定位。	下载工具**whl包安装**使用，推荐使用最新版本。	**大模型推理精度工具**

10.2 GPU 训练业务迁移至昇腾的通用指导

10.2.1 训练业务迁移到昇腾设备场景介绍

场景介绍

本文介绍如何将客户已有的PyTorch训练业务迁移到昇腾设备上运行并获得较好的模型训练效果。华为云ModelArts针对该场景提供了系统化的迁移指导，包括迁移原理、迁移流程以及迁移后的精度调试及性能调优方法介绍。此外，ModelArts提供了即开即用

的云上集成开发环境，包含迁移所需要的算力资源、AI框架、昇腾开发套件以及迁移调优工具链，最大程度减少客户自行配置环境的复杂度。

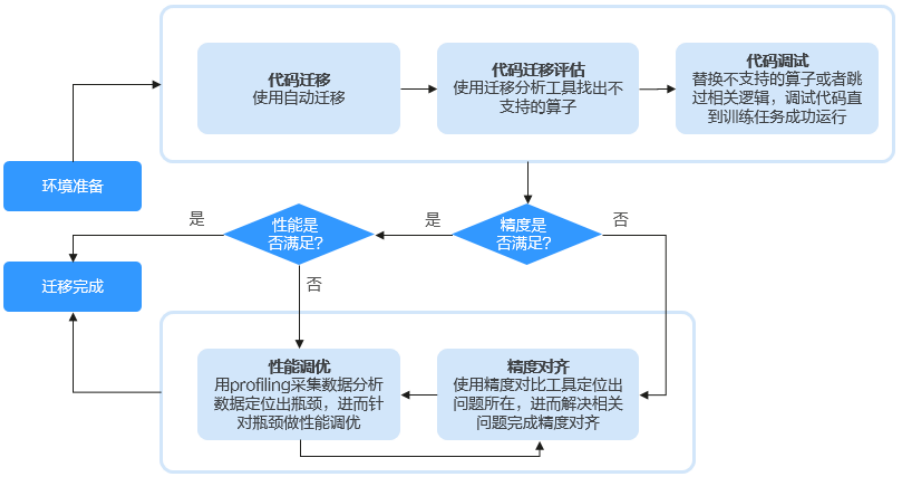
范围

本文涉及PyTorch训练的单卡和分布式业务迁移到昇腾的业务范围。当前针对常见的开源LLM/AIGC等领域的开源模型，ModelArts已经提供了迁移好的开箱即用模型，且保证了较优的精度和性能。如果用户业务同样使用这些开源模型，建议直接使用ModelArts提供的**模型运行指导**，其余场景再考虑使用本指导自行迁移和调优。

迁移流程

模型迁移主要指将开源社区中实现过的模型或客户自研模型迁移到昇腾AI处理器上，需要保证模型已经在CPU/GPU上运行成功。迁移到昇腾AI处理器的主要流程如下图所示。

图 10-1 迁移流程



10.2.2 训练迁移快速入门案例

本篇指导是迁移的总体思路介绍，便于用户对迁移过程有一个整体的认识。如果您希望通过具体案例直接实操，请参考《**主流开源大模型基于DevServer适配PyTorch NPU训练指导**》。该案例以ChatGLM-6B为例，介绍如何将模型迁移至昇腾设备上训练、模型精度对齐以及性能调优。

迁移环境准备

本文以弹性裸金属作为开发环境。弹性裸金属支持深度自定义环境安装，可以方便地替换驱动、固件和上层开发包，具有root权限，结合配置指导、初始化工具及容器镜像可以快速搭建昇腾开发环境。

开通裸金属服务器资源请参见**DevServer资源开通**，在裸金属服务器上搭建迁移环境请参见**裸金属服务器环境配置指导**，使用ModelArts提供的基础容器镜像请参见**容器环境搭建**。

训练代码迁移

前提条件

- 要迁移的训练任务代码在GPU上多次训练稳定可收敛。训练业务代码和数据，应该确保在GPU环境中能够运行，并且训练任务有稳定的收敛效果。
- 本文只针对基于PyTorch的训练代码迁移。此处假设用户使用基于PyTorch的训练代码进行迁移。其他的AI引擎如TensorFlow、Caffe等不在本指导的讨论范围中。
- 已完成**迁移环境准备**，且代码、预训练模型、数据等训练必需内容已经上传到环境中。

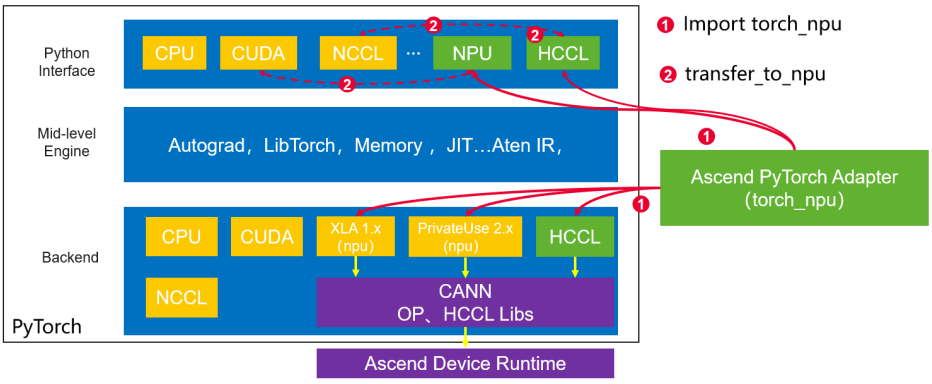
约束和限制

- 安装插件后，大部分能力能够对标在GPU上的使用，但并不是所有行为和GPU上是一一对应的。例如在torch_npu下，当PyTorch版本低于2.1.0时，一个进程只能操作一张昇腾卡，不支持一个进程操作多卡的能力；在PyTorch2.1.0及以上版本中torch_npu才支持一个进程中使用多张昇腾卡。
- 基于PyTorch上的第三方开发库非常多，例如transformers、accelerate、deepspeed以及Megatron-LM等，这些三方库昇腾也做了类似PyTorch Adapter的适配插件库。您可以在Gitee的昇腾**官方仓库**按需使用插件库。部分三方库例如最新版本deepspeed已原生支持NPU，可以直接在昇腾设备上运行。

代码迁移基础知识

- PyTorch 2.1以下版本时，PyTorch官方并不直接支持昇腾的后端，仅直接支持CUDA和AMD ROCm，因此PyTorch在GPU上的训练代码无法直接在昇腾设备运行。PyTorch 2.1版本提供了新硬件适配的插件机制，通过昇腾提供的Ascend Extension for PyTorch插件，NPU可以成为PyTorch支持的硬件直接使用。
- **Ascend Extension for PyTorch**作为一个PyTorch插件，支持在不改变PyTorch表达层的基础上，动态添加昇腾后端适配，包含增加了NPU设备、hccl等一系列能力的支持。安装后可以直接使用PyTorch的表达层来运行在NPU设备上。
- 当前提供了自动迁移工具进行GPU到昇腾适配，原理是通过**monkey-patch**的方式将torch下的CUDA、nccl等操作映射为NPU和hccl对应的操作。如果没有用到GPU的高阶能力，例如自定义算子、直接操作GPU显存等操作，简单场景下可以直接使用自动迁移。

图 10-2 torch_npu 工作原理示意图



- NPU（Neural Network Processing Unit）和GPU在构造结构上存在差异，因此迁移过程并不是完全平替的关系。昇腾训练芯片属于NPU的范畴，虽然在表达层可以通过torch.cuda和torch.npu的形式来替代，但是真实的算子下发、显存管理、

集合通信等存在差异，用户需要了解NPU的运行机制才能更好的使用NPU设备，同时在遇到问题时快速找到原因。

代码迁移操作步骤

步骤1 在训练任务启动的Python脚本入口初始化Ascend Extension for PyTorch（torch_npu）。

在torch_npu安装后，该部分并没有直接植入到PyTorch中生效，需要用户显式调用。

```
# torch npu初始化。
import torch_npu
```

调用后，前端会通过monkey-patch的方式注入到torch对象中，后端会注册NPU设备以及HCCL的参数面通信能力，这样就可以运行torch.npu相关接口。

图 10-3 torch_npu 导入

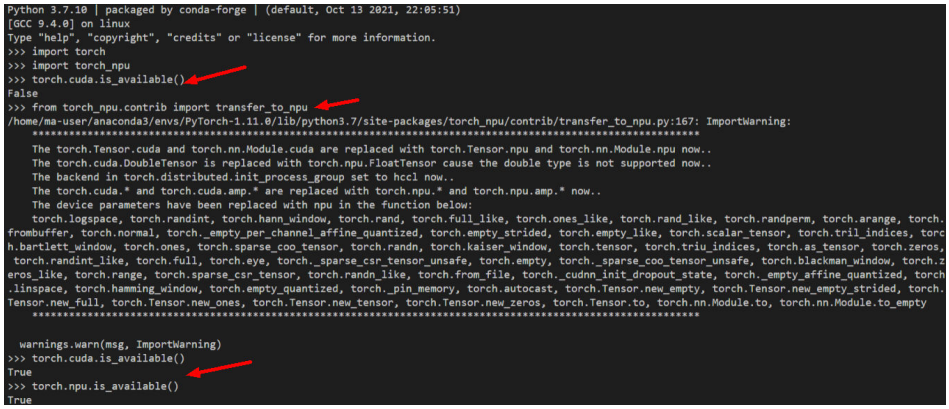
步骤2 自动迁移完成GPU代码到昇腾的快速适配。

torch_npu初始化后，原则上需要用户将原来代码中CUDA相关的内容迁移到NPU相关的接口上，包含算子API、显存操作、数据集操作、分布式训练的参数面通信nccl等，手动操作修改点较多且较为分散，因此昇腾提供了自动迁移工具transfer_to_npu帮助用户快速迁移。

自动迁移的原理是：通过注入的方式将当前Python运行环境中，运行时的torch.cuda等需要适配的接口和操作都映射成为torch.npu对应的接口。所以理论上常见场景下的代码不需要额外手工适配就可以运行到昇腾设备上。

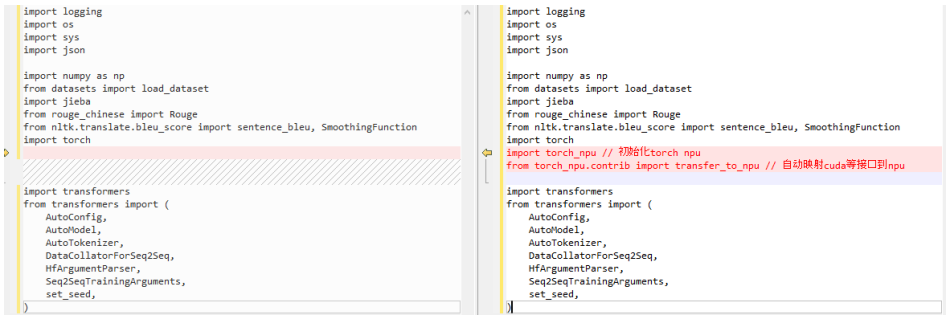
```
# 自动映射cuda API到NPU的代码。
from torch_npu.contrib import transfer_to_npu
```

图 10-4 自动迁移后 cuda 映射为 NPU 相关的 API



以**chatGLM-6b**为例，在使用自动迁移时，在开发环境中克隆对应的代码。假设数据和预训练权重已经配置好，可以直接在ptuning目录下，训练入口代码main.py中添加两行代码来完成昇腾适配。请注意添加位置为导入torch之后。启动训练脚本可以观察运行效果。

图 10-5 chatGLM-6b pTuning 训练入口导入自动迁移工具



说明

自动迁移适合没有使用CUDA高阶能力的简单场景，如果涉及自定义算子、主动申请GPU显存等操作，则需要额外进行手动迁移适配。

步骤3 手动迁移解决报错问题。

在完成代码自动迁移后，如果训练代码运行时还出现错误，则代表需要手动迁移适配。针对代码报错处，需要用户分析定位后将自动迁移未能迁移的GPU相关的代码调用修改为NPU对应的接口，请参考昇腾**手工迁移**文档进行操作。

----结束

常见问题

1. 如何检测当前的torch_npu是否正确安装?

   您可以使用如下的python命令在对应的运行环境中初步校验torch_npu是否正常安装。

   ```
   python3 -c "import torch;import torch_npu;print(torch_npu.npu.is_available())"
   ```

2. torch_npu使用报错看不懂怎么办？应该怎么求助?

   如果报错可以首先在**昇腾社区论坛**以及**Gitee的PyTorch** Issues中查看是否有类似的问题找到相关线索。如果还无法解决，可以通过提交工单的形式从华为云ModelArts入口进行咨询以及求助对应的专业服务。

3. 自动迁移似乎还要改很多脚本才能运行起来?

   因为自动迁移其实是对于torch运行环境中常用的GPU上的接口进行和昇腾设备的映射。原有的训练任务代码逻辑中例如数据集导入、预训练权重、GPU自定义算子的内容，以及对应的环境的超参数等内容都需要在实际的昇腾环境中进行调整。

10.2.3 PyTorch 迁移精度调优

10.2.3.1 精度问题概述

随着ChatGPT的推出，大模型迅速成为AI界热点。大模型训练需要强大的算力支撑，涉及数据、模型、框架、算子、硬件等诸多环节。由于规模巨大，训练过程复杂，经常出现loss不收敛的情况（模型精度问题），主要表现为loss曲线起飞或者毛刺等，且模型的下游任务评测效果变差。影响大模型loss收敛的原因是多方面的：首先，数据问题可能导致不收敛，比如数据预处理不完善；其次，模型的训练超参数也同样会导致类似的情况；再者，模型本身的算法设计过程也可能会引入不收敛情况；最后，则是由计算过程导致的模型收敛问题。

模型精度（以模型评测结果衡量的各种指标，广义的Model Accuracy），是多种因素共同作用的结果，出现问题的主要表现是训练过程的Loss不收敛或者收敛出问题或者loss收敛却评测集上表现不佳。而计算的数值精度问题，（Computational Precision，Floating-Point Arithmetic Precision等）,则是由于浮点数计算过程的有限字长效应及计算序所带来的近似误差，包括各种计算的数学表达，都会带来结果的近似性。二者是完全不同的两个问题， 不能混为一谈。计算数值的近似性一定概率上会影响模型的收敛性，但是影响大模型收敛的原因是复杂且多样的，大模型本身也对计算差异有一定韧性，所以，不能简认地为计算过程的差异一定会导致模型收敛出现问题。算子的数值精度是计算过程的基础，通常认为算子精度问题是大模型精度问题的来源之一，从实际经验看，算子数值精度不足（除去计算错误等BUG问题）所导致的模型收敛问题在整个模型收敛比例里面较低，但其影响会较大，所以，该问题需要引起重视。

而且，由于实现过程差异，不同硬件对于同样的计算过程，数值计算结果通常会有差异，比如GPU和CPU之间，GPU各版本之间，数值计算结果都有一定差异，在特定的容限范围内，不会影响模型的最终收敛。所以，计算的数值差异是很常规的现象，并非错误。

为了更好地了解这种计算差异，并且能够正确区分正常计算差异和引起模型精度问题的异常差异，本指南提供了算子问题定位工具集详细的使用场景和使用步骤，方便用户自行或在支持下排查可能的数值计算精度问题。

当用户将大语言模型或者其他类型深度神经网络的训练从GPU迁移到昇腾AI处理器时，可能出现以下不同现象的模型精度问题。一般包括：

- Loss曲线与CPU/GPU差异不符合预期。
- 验证准确度与CPU/GPU差异不符合预期。

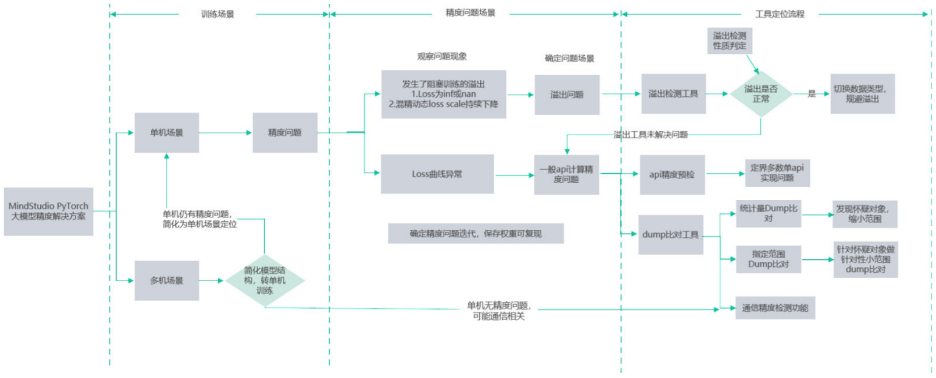
在迁移到NPU环境下训练发现以上问题时，说明精度可能存在偏差，需要进一步做精度调优。下文将分别阐述精度诊断的整体思路和如何借助精度工具进行精度问题的定位。

10.2.3.2 精度调优总体思路

PyTorch大模型训练的精度问题的分析、定位可以参考如下思路：

1. 大模型训练通常使用多机训练，鉴于多机训练复现问题的成本较高，且影响因子较多，建议用户先减少模型层数，使模型能够单机训练，确认单机训练是否也存在精度问题，若存在，则使用下述手段定位精度问题，使得单机精度达标，然后再恢复层数拉起多机训练。
2. 若单机精度正常但多机精度异常，有可能是多机通信造成的精度问题，此时可以用精度工具的通信精度检测功能进行定位。部分集合通信算子要求通信域内各rank结果一致，如AllReduce、AllGather等，利用这一特性，工具将多机模型训练中产生的通信输出存盘，并传输到同一节点来比较其一致性，从而确定模型中通信算子的精度是否存在问题。若已排除通信算子异常，则可能是由于网络层数增加放大了累积误差，需要使用精度比对等工具进一步分析。

图 10-6 精度调优流程



10.2.3.3 精度调优前准备工作

在定位精度问题之前，首先需要排除训练脚本及参数配置等差异的干扰。目前大部分精度无法对齐的问题都是由于模型超参数、Python三方库版本、模型源码等与标杆环境（GPU/CPU）设置的不一致导致，为了在定位过程中少走弯路，需要在定位前先对训练环境及代码做有效排查。此外，问题定位主要基于GPU环境和NPU环境上运行的过程数据做对比，所以需要分别准备GPU和NPU训练环境，大部分场景需要规模相同的训练环境。如果已经将模型缩减到单机可运行，则只是单台GPU设备即可。

定位前的排查当前主要包含如下几个方面：

1. 训练超参数。常见的超参如下图所示：

图 10-7 训练超参数

```
python -m torch.distributed.launch $DISTRIBUTED_ARGS \
       pretrain_llama.py \
       --DDP-impl local \
       --tensor-model-parallel-size 8 \
       --pipeline-model-parallel-size 1 \
       --sequence-parallel \
       --num-layers 32 \
       --hidden-size 4096 \
       --ffn-hidden-size 11008 \
       --num-attention-heads 32 \
       --attention-dropout 0.0 \
       --hidden-dropout 0.0 \
       --init-method-std 0.01 \
       --micro-batch-size 4 \
       --global-batch-size 16 \
       --seq-length 4096 \
       --max-position-embeddings 4096 \
       --data-path $DATA_PATH \
       --tokenizer-name-or-path $TOKENIZER_PATH \
       --tokenizer-not-use-fast \
       --split 100,0,0 \
       --distributed-backend nccl \
       --lr 1.25e-5 \
       --min-lr 1.25e-6 \
       --lr-decay-style cosine \
       --weight-decay 1e-1 \
       --clip-grad 1.0 \
       --initial-loss-scale 65536.0 \
       --adam-beta1 0.9 \
       --adam-beta2 0.95 \
       --log-interval 1 \
       --load ${LOAD_CHECKPOINT_PATH} \
       --save ${SAVE_CHECKPOINT_PATH} \
       --save-interval 10000 \
       --eval-interval 10000 \
       --eval-iters 0 \
       --use-fused-rotary-pos-emb \
       --no-query-key-layer-scaling \
       --attention-softmax-in-fp32 \
       --no-masked-softmax-fusion \
       --train-iters 50000 \
       --lr-warmup-fraction 0.01 \
       --no-load-optim \
       --no-load-rng \
       --mlp-layer-fusion \
       --use-flash-attn \
       --print-input-ids \
       --bf16 | tee ./logs/ascendspeed_npu_7b-bf16_tp8pp1mbs4_16_fa_sp_cann1107cmc3${logfile}.log
```

模型的超参通常可能调整的主要有学习率、batch size、并行切分策略、学习率warm-up、模型参数、FA配置等。**用户在进行NPU精度和GPU精度比对前，需要保证两边的配置一致。**

表 10-2 超参说明

超参	说明
学习率	影响模型收敛程度，决定了模型在每次更新权重时所采用的步长。学习率过高，模型可能会过度调整权重，导致不稳定的训练过程；如果学习率过低，模型训练速度会变慢，甚至陷入局部最优。
batch size	影响训练速度，有时候也会影响模型精度。
micro batch size	影响流水线并行中设备的计算效率。

超参	说明
切分策略	包括DP（Data Parallel）、TP（Tensor Parallel）、PP（Pipeline Parallel）。 • DP：数据并行（Data Parallelism）是大规模深度学习训练中常用的并行模式，它会在每个进程(设备)或模型并行组中维护完整的模型和参数，但在每个进程上或模型并行组中处理不同的数据。因此，数据并行非常适合大数据量的训练任务。 • TP：张量并行也叫层内并行，通过将网络中的权重切分到不同的设备，从而降低单个设备的显存消耗，使得超大规模模型训练成为可能。张量并行不会增加设备等待时间，除了通信代价外，没有额外代价。 • PP：流水线并行将模型的不同层放置到不同的计算设备，降低单个计算设备的显存消耗，从而实现超大规模模型训练。流水线并行也叫层间并行，层输入输出的依赖性使得设备需要等待前一步的输出，通过batch进一步切分成微batch， 网络层在多个设备上的特殊安排和巧妙的前向后向计算调度，可以最大程度减小设备等待（计算空泡），从而提高训练效率。
学习率预热	不同的学习率调度器（决定什么阶段用多大的学习率）有不同的学习率调度相关超参，例如线性调度可以选择从一个初始学习率lr-warmup-init开始预热。您可以选择多少比例的训练迭代步使用预热阶段的学习率。不同的训练框架有不同的参数命名，需要结合代码实现设置对应的参数。
模型结构	配置模型结构的超参主要有num-layer、hidden-size、seq-length等。
FA配置	超参数为use-flash-attn，决定训练过程中的Attention模块是否使用融合flash attention算子（性能较优）或者使用小算子。

2. 训练脚本

   由算法迁移人员排查迁移后的NPU脚本是否存在问题，可以通过Beyond Compare工具比对GPU训练脚本和NPU训练脚本之间是否存在差异。例如是否GPU环境下开启了FA但是NPU上未开启FA。

3. 三方库版本比对

   大模型训练通常会使用Deepspeed、Megatron等三方库，需要确保这些三方库的版本一致。

4. 环境版本更新

   这一项仅在条件允许的情况下进行，根据精度问题定位经验，部分问题是由于使用了较早版本的昇腾软件版本或者非商用发布的昇腾软件版本，所以推荐在条件允许的前提下配套安装最新商发版本的昇腾开发套件CANN Toolkit、昇腾驱动以及torch_npu包。具体操作，请参考**昇腾商用版资源下载指导**。

5. 数据集。

   需要排查是否使用的训练数据集存在差异。

6. 初始权重。

需要排查是否加载的初始权重有差异，建议加载相同的初始权重。

10.2.3.4 msprobe 精度分析工具使用指导

msprobe是MindStudio Training Tools工具链下精度调试部分的工具包，其通过采集和对比标杆（GPU/CPU）环境和昇腾环境上运行训练时的差异点来判断问题所在，主要包括精度预检、精度比对和梯度监控等功能。更多内容请参考**msprobe工具介绍**。

一般场景的训练模型都是包括随机种子、数据集Shuffle、网络结构Dropout等操作的，目的是在网络阶段引入一定的随机性使得训练结果更加具有鲁棒性。然而在精度对齐阶段，这些随机性会导致训练运行结果每次表现不一致，无法进行和标杆的比对。因此在训练模型复现问题时，需要固定存在随机性的步骤，保证实验可重复性。存在随机性的步骤包括模型参数初始化，数据Batch加载顺序，Dropout层等。部分算子的计算结果也存在不确定性，需要固定。

当前固定随机性操作可分为**工具固定**和**人工固定**两种。

1. 工具固定Seed

   对于网络中随机性的固定，msprobe提供了固定Seed的方式，只需要在config.json文件中添加对应seed配置即可。

   msprobe工具提供了seed_all接口用于固定网络中的随机数。如果客户使用了工具但取用了其他随机种子，则必须使用客户的随机种子固定随机性。

   函数原型

   ```
   from msprobe.pytorch.common import seed_all
   seed_all(seed=1234, mode=False)
   ```

   表 10-3 参数说明

参数名	说明	是否必选
seed	随机数种子。参数示例：seed=1000。默认值：1234。	否
mode	确定性计算模式。可配置True或False。参数示例：mode=True。默认值：False。 即使在相同的硬件和输入下，API多次执行的结果也可能不同，开启确定性计算是为了保证在相同的硬件和输入下，API多次执行的结果相同。 确定性计算会导致API执行性能降低，通常不需要在精度问题刚开始定位时就开启，而是建议在发现模型多次执行结果不同的情况下时再开启。 rnn类算子、ReduceSum、ReduceMean等算子可能与确定性计算存在冲突，如果开启确定性计算后多次执行的结果不相同，则考虑存在这些算子。	否

   函数示例

   seed_all函数的随机数种子，取默认值即可，无须配置；第二个参数默认关闭，不开启确定性计算时也无须配置。

   确定性计算是NPU的一套机制，用于保证算子的计算确定性。之所以要有这个机制，是为了在Debug过程中，让所有的算子计算结果前后完全一致可复现，这是大多数精度问题分析的重要前提。因此，在精度问题定位过程中，确定性计算**不**

是目的，而是手段。很多场景下需要在确定性计算使能的情况下，进行下一步的精度问题分析定位。Cuda对部分算子实现了确定性计算，但仍有部分算子无法固定。通常需要依赖确定性计算的场景是长稳问题，因为长稳问题需要通过多次长跑来分析Loss情况，这时候如果NPU本身计算结果不确定，就难以支撑和GPU结果的多次对比。

示例1：仅固定随机数，不开启确定性计算。

```
seed_all()
```

示例2：固定随机数，开启确定性计算。

```
seed_all(mode=True)
```

在多卡训练场景下由于通信算子计算累加计算顺序不确定，需要添加以下环境变量，固定通信算子计算的确定性：

```
export HCCL_DETERMINISTIC=TRUE
```

固定随机数范围

seed_all函数可固定随机数的范围如下表所示。

API	固定随机数
os.environ['PYTHONHASHSEED'] = str(seed)	禁止Python中的hash随机化。
random.seed(seed)	设置random随机生成器的种子。
np.random.seed(seed)	设置numpy中随机生成器的种子。
torch.manual_seed(seed)	设置当前CPU的随机种子。
torch.cuda.manual_seed(seed)	设置当前GPU的随机种子。
torch.cuda.manual_seed_all(seed)	设置所有GPU的随机种子。
torch_npu.npu.manual_seed(seed)	设置当前NPU的随机种子。
torch_npu.npu.manual_seed_all(seed)	设置所有NPU的随机种子。
torch.backends.cudnn.enable=False	关闭cuDNN。
torch.backends.cudnn.benchmark=False	cuDNN确定性地选择算法。
torch.backends.cudnn.deterministic=True	cuDNN仅使用确定性的卷积算法。

2. 工具固定（Dropout）

   Dropout的实质是以一定概率使得输入网络的数据某些位置元素的数值变为0，这样可以使得模型训练更加有效。但在精度问题的定位过程之中，需要避免产生这种问题，因此需要关闭Dropout。

   在导入PrecisionDebugger后，工具会自动将如下接口参数p（丢弃概率）置为0。

   ```
   torch.nn.functional.dropout
   torch.nn.functional.dropout2d
   torch.nn.functional.dropout3d
   torch.nn.Dropout
   torch.nn.Dropout2d
   torch.nn.Dropout3d
   ```

3. 人工固定（硬件随机差异）

   工具内部对于随机的控制，是通过设定统一的随机种子进行随机性固定的。但是由于硬件的差异，会导致同样的随机种子在不同硬件上生成的随机数不同。具体示例如下：

```
>>> from msprobe.pytorch.common import seed_all
>>> seed_all()
>>> torch.rand((2, 3), device="cuda")
tensor([[0.1272, 0.8167, 0.5440],
        [0.6601, 0.2721, 0.9737]], device='cuda:0')
```

```
>>> import torch, torch_npu
>>> from msprobe.pytorch.common import seed_all
>>> torch.rand((2, 3), device="npu")
tensor([[0.0102, 0.5372, 0.6823],
        [0.6912, 0.3431, 0.5492]], device='npu:0')
```

   由上图可见，torch.randn在GPU和NPU上固定随机种子后，仍然生成不同的随机张量。

   对于上述场景，用户需要将网络中的randn在CPU上完成后再转到对应device。例如，StableDiffusion中需要在forward过程中逐步生成随机噪声。

```
>>> import torch
>>> from msprobe.pytorch.common import seed_all
>>> seed_all()
>>> tensor = torch.rand((2, 3), device="cpu")
>>> tensor
tensor([[0.0290, 0.4019, 0.2598],
        [0.3666, 0.0583, 0.7006]])
>>> tensor_cuda = tensor.cuda()
>>> tensor_cuda
tensor([[0.0290, 0.4019, 0.2598],
        [0.3666, 0.0583, 0.7006]], device='cuda:0')
```

```
>>> import torch, torch_npu
>>> from msprobe.pytorch.common import seed_all
>>> seed_all()
>>> tensor = torch.rand((2, 3), device="cpu")
>>> tensor
tensor([[0.0290, 0.4019, 0.2598],
        [0.3666, 0.0583, 0.7006]])
>>> tensor_npu = tensor.npu()
>>> tensor_npu
tensor([[0.0290, 0.4019, 0.2598],
        [0.3666, 0.0583, 0.7006]], device='npu:0')
```

   这样在Host侧生成的随机张量能够保证一样，搬移到NPU或者GPU设备上仍然一样。

   固定随机性完成后，可以使用缩小的模型在单机环境进行问题复现。复现后使用**msprobe工具**进行问题定位。需要注意的是，部分模型算法本身存在固有的随机性，在使用上述方法固定随机性后，如果使用工具也未能找到出问题的API，需要分析是否由算法本身的随机性导致。

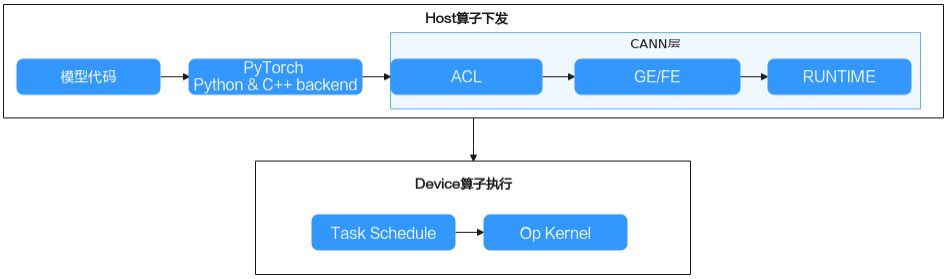
10.2.4 PyTorch 迁移性能调优

10.2.4.1 性能调优总体原则和思路

PyTorch在昇腾AI处理器的加速实现方式是以算子为粒度进行调用（OP-based），即通过Python与C++调用CANN层接口Ascend Computing Language（AscendCL）调用一个或几个亲和算子组合的形式，代替原有GPU的实现方式，具体逻辑模型请参考**PyTorch自动迁移**。

在PyTorch模型迁移后进行训练的过程中，CPU只负责算子的下发，而NPU负责算子的执行，算子下发和执行异步发生，性能瓶颈在此过程中体现。在PyTorch的动态图机制下，算子被CPU逐个下发到NPU上执行。一方面，理想情况下CPU侧算子下发会明显比NPU侧算子执行更快，此时性能瓶颈主要集中在NPU侧；另一方面，理想情况下NPU侧算子计算流水线一直执行，不会出现NPU等待CPU算子下发即NPU空转的场景，如果存在，则CPU侧算子下发存在瓶颈。

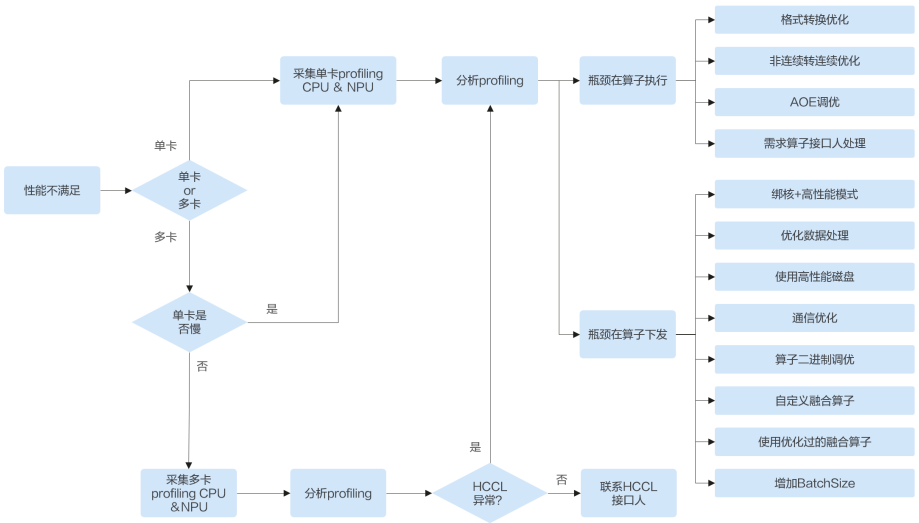
图 10-8 Host 算子下发和 Device 算子执行



综上所述，性能优化的总体原则为：减少Host算子下发时间、减少Device算子执行时间。

训练代码迁移完成后，如存在性能不达标的问题，可参考下图所示流程进行优化。建议按照单卡、单机多卡、多机多卡的流程逐步做性能调优。

图 10-9 性能调优总体思路



为了便于用户快速进行迁移调优，降低调优门槛，ModelArts提供了MA-Adivisor性能自动诊断工具。用户采集性能profiling数据后，可通过该工具自动扫描profiling数据，工具分析完数据后会给出可能的性能问题点及调优建议，用户可以根据调优建议做相应的修改适配。目前该工具对CV类模型给出的调优建议较多，LLM类建议稍少，但是

总体都有性能提升，实测大约可提升10%~30%的性能，并且已经在多个迁移性能调优项目中实际应用。

10.2.4.2 MA-Advisor 性能调优建议工具使用指导

MA-Advisor是一款迁移性能问题自动诊断工具，其集成了昇腾自动诊断工具msprof-analyze，并在ModelArts Standard的Jupyter lab平台进行了插件化，能快速分析和诊断昇腾场景下PyTorch性能劣化问题并给出相关调优建议。

在过往性能调优场景中，如果性能profiling数据在OBS上，通常需要将TB或者GB级别的profiling数据下载至本地后才能使用msprof-analyze进行分析，大量数据的下载耗时以及对本地大规格存储盘的要求容易导致分析受阻。为了能自动串联高性能挂载OBS至ModelArts环境和msprof-analyze的分析能力，ModelArts Standard 场景下对外提供一种插件化的 advisor 分析能力，详细的操作方式请参见**基于advisor的昇腾训练性能自助调优指导**。

对于GPU和NPU性能比对、NPU多次训练之间性能比对的场景，昇腾提供了性能比对工具**compare_tools**，通过对训练耗时和内存占用的比对分析，定位到具体劣化的算子，帮助用户提升性能调优的效率。工具将训练耗时拆分为计算、通信、调度三大维度，并针对计算和通信分别进行算子级别的比对；将训练占用的总内存，拆分成算子级别的内存占用进行比对。

对于集群训练场景，昇腾提供了集群分析工具**cluster_analysis**，当前主要对基于通信域的迭代内耗时分析、通信时间分析以及通信矩阵分析为主，从而定位慢卡、慢节点以及慢链路问题。

10.2.4.3 MindStudio-Insight 性能可视化工具使用指导

对于高阶的调优用户，可以使用可视化工具MindStudio Insight查看profiling数据详情并分析可优化点，其提供了丰富的调优分析手段，可视化呈现真实软硬件运行数据，多维度分析性能瓶颈点，支持百卡、千卡及以上规模的可视化集群性能分析，助力开发者天级完成性能调优。

MindStudio-Insight提供时间线视图、内存、算子耗时、通信瓶颈分析等功能，借助于数据库支持超大性能数据处理，可以支持20GB的集群性能文件分析，并且能够支持大模型场景下的性能调优，相比于Chrometrace、tensorboard等工具提供了更优的功能和性能。

更多详细信息，请参见昇腾**MindStudio-Insight用户指南**。

10.2.5 训练网络迁移总结

- 确保算法在GPU训练时，持续稳定可收敛。避免在迁移过程中排查可能的算法问题，并且要有好的对比标杆。如果是NPU上全新开发的网络，请参考**PyTorch迁移精度调优**排查溢出和精度问题。
- 理解GPU和NPU的构造以及运行的差别，有助于在迁移过程中分析问题并发挥NPU的优势。由于构造和运行机制的差别，整个迁移过程并非是完全平替，GPU在灵活性上有其独特的优势，而NPU上的执行目前还是依赖于算子的下发，对于NPU构造的理解是昇腾训练迁移中必备的知识，只有对于昇腾有基础理解，配合一些诊断工具，面对复杂问题时，才能进行进一步诊断与定位，进而发挥NPU的能力。
- 性能调优可以先将重点放在NPU不亲和的问题处理上，确保一些已知的性能问题和优化方法得到较好的应用。通用的训练任务调优、参数调优可以通过可观测数据来进行分析与优化，一般来说分段对比GPU的运行性能会有比较好的参考。算

子级的调优某些情况下如果是明显的瓶颈或者性能攻坚阶段，考虑到门槛较高，可以联系华为工程师获得帮助。

- 精度问题根因和表现种类很多，会导致问题定位较为复杂，一般还是需要GPU上充分稳定的网络（包含混合精度）再到NPU上排查精度问题。常见的精度调测手段，包含使用全精度FP32，或者关闭算子融合开关等，先进行排查。对于精度问题，系统工程人员需要对算法原理有较深入的理解，仅从工程角度分析有时候会非常受限，同时也可联系华为工程师进行诊断与优化。

10.3 基于 AIGC 模型的 GPU 推理业务迁移至昇腾指导

10.3.1 场景介绍

阅读本文前建议您先了解以下内容：

- Stable Diffusion的基础知识，可参考**Stable Diffusion github**、**Stable Diffusion wikipedia**、**diffusers github**、**Stable Diffusion with diffusers**。
- 推理业务迁移到昇腾的通用流程，可参考**GPU推理业务迁移至昇腾的通用指导**。

  说明

  由于Huggingface网站的限制，访问Stable Diffusion链接时需使用代理服务器，否则可能无法访问网站。

在Stable Diffusion迁移适配时，更多的时候是在适配Diffusers和Stable Diffusion WebUI，使其能够在昇腾的设备上运行。其中，Diffusers遵循了Huggingface的“**single-file policy**”的设计原则，它的三个主要模块Pipeline、Schedulers和预训练模型中，Pipeline和Schedulers都完全遵循了“single-file policy”原则。该设计原则更推荐直接复制粘贴代码，而不是进行抽象处理。因此，与模型前向运算相关的所有源代码都被直接复制粘贴到同一个文件中，而不是调用某些抽象提取出的模块化库。Diffusers的这种设计原则的好处是代码简单易用、对代码贡献者友好。然而，这种反软件结构化的设计也有明显的缺点。由于缺乏统一的模块化库，对于昇腾适配而言变得更加复杂，必须针对每个不同业务的Pipeline进行单独适配。

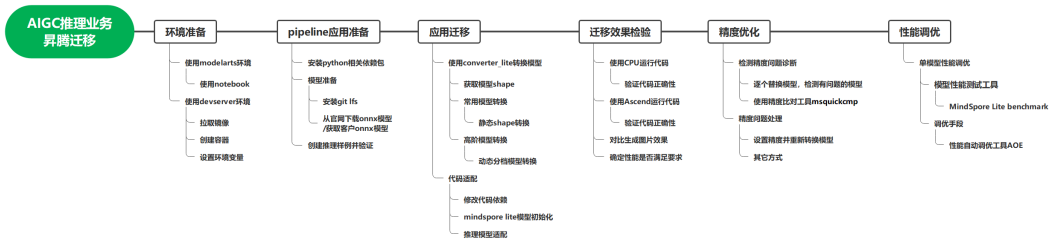
本文以**Stable Diffusion v1.5**的图生图为例，通过可以直接执行的样例代码介绍Diffusers的昇腾迁移过程。对于其他pipeline的迁移，可以在充分理解其代码的基础上，参考本文的思路进行举一反三。Stable Diffusion WebUI的迁移不包含在本文中，具体原因详见**Stable Diffusion WebUI如何适配**。

AI推理应用运行在昇腾设备上一般有两种方式：

- 方式1：通过Ascend PyTorch，后端执行推理，又称在线推理。
- 方式2：通过模型静态转换后，执行推理，又称离线推理。

通常为了获取更好的推理性能，推荐使用方式2的离线推理。下文将以Diffusers img2img onnx pipeline为示例来讲解如何进行离线推理模式下的昇腾迁移。迁移的整体流程如下图所示：

图 10-10 迁移流程图



10.3.2 迁移环境准备

迁移环境准备有以下两种方式：

表 10-4 方式说明

序号	名称	说明
方式一	ModelArts Notebook	该环境为在线调试环境，主要面向演示、体验和快速原型调试场景。 • 优点：可快速、低成本地搭建环境，使用标准化容器镜像，官方Notebook示例可直接运行。 • 缺点：由于是容器化环境因此不如裸机方式灵活，例如不支持root权限操作、驱动更新等。 环境开通指导请参考**Notebook环境创建**；样例演示请参考**Notebook样例：Stable Diffusion模型迁移到Ascend上进行推理**。
方式二	ModelArts Lite DevServer	该环境为裸机开发环境，主要面向深度定制化开发场景。 • 优点：支持深度自定义环境安装，可以方便的替换驱动、固件和上层开发包，具有root权限，结合配置指导、初始化工具及容器镜像可以快速搭建昇腾开发环境。 • 缺点：资源申请周期长，购买成本高，管理视角下资源使用效率较低。 环境开通指导请参考**DevServer资源开通**；环境配置指导请参考**Snt9B裸金属服务器环境配置指南**。

本文基于方式二的环境进行操作，请参考方式二中的环境开通和配置指导完成裸机和容器开发初始化配置。注意业务基础镜像选择**Ascend+PyTorch**镜像。

配置好的容器环境如下图所示：

图 10-11 环境配置完成

```
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker run -itd --cap-add=SYS_PTRACE -e ASCEND_VISIBLE_DEVICES=3 -v
/home:/host_home -u=0 --name pytorch_test swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.
11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-d910b-20230815141604-3685231 /bin/bash
0292be41ac1ef03a37b7c78adcff4fc999a967e1163e5f6e565edbe6a638c69b
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker exec -ti 0292be41a bash
The environment has been set
[root@0292be41ac1e ma-user]# source .bashrc
The environment has been set
The environment has been set
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]# python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu();
 print(a + a);"
tensor([[-1.0911, -0.4146,  1.6027,  1.8585],
        [ 3.2549,  0.7026,  2.9356,  0.9544],
        [ 5.1409, -0.8820, -0.3400,  0.0257]], device='npu:0')
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]#
```

10.3.3 pipeline 应用准备

当前迁移路径是从ONNX模型转换到MindIR模型，再用MindSpore Lite做推理， 所以迁移前需要用户先准备好自己的ONNX pipeline。下文以官方开源的图生图的Stable Diffusion v1.5的onnx pipeline代码为例进行说明。

步骤1 进入容器环境，创建自己的工作目录。

由于在**Snt9B裸金属服务器环境配置指南**的配置环境步骤中，在启动容器时将物理机的home目录挂载到容器的“/home_host”目录下，该目录可以直接使用上传到物理机“home”目录下的文件。本文中，将基于容器的“/home_host”目录创建工作目录。

```
mkdir -p /home_host/work
cd /home_host/work
```

步骤2 在迁移onnx pipeline前，首先需要确保原始的onnx pipeline能在昇腾机器的ARM CPU上正常执行。进入容器环境后，安装依赖包。

```
pip install torch==1.11.0 onnx transformers==4.27.4 accelerate onnxruntime diffusers==0.11.1
```

步骤3 下载git lfs，用于下载git仓中的大文件。

由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到服务器的/home目录。

```
https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
```

步骤4 安装git lfs。

```
tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
cd git-lfs-3.2.0
sh install.sh
rm -rf git-lfs-linux-arm64-v3.2.0.tar.gz git-lfs-3.2.0
```

步骤5 通过git下载sd PyTorch模型。

该模型用于获取模型shape，也可以转换生成onnx模型。后文中的modelarts-ascend仓库已经给出了模型shape，可以直接使用，onnx模型也可以单独下载。

```
# git clone sd模型。
git lfs install
mkdir -p /home_host/work/runwayml
cd /home_host/work/runwayml
git clone https://huggingface.co/runwayml/stable-diffusion-v1-5/ -b main
# 将下载的文件夹重命名，以便后续脚本中引用。
mv stable-diffusion-v1-5 pytorch_models
```

说明

此处由于Huggingface网站的限制以及模型文件的大小原因，很可能会下载失败。您可以登录**Huggingface网站**，从浏览器下载模型后，再手动上传到物理机/home/pytorch_models目录下。

步骤6 通过git下载sd onnx模型。

```
# git clone sd模型。
git lfs install
cd /home_host/work/runwayml
git clone https://huggingface.co/runwayml/stable-diffusion-v1-5 -b onnx
# 将下载的文件夹重命名，以便后续脚本中引用。
mv stable-diffusion-v1-5 onnx_models
```

说明

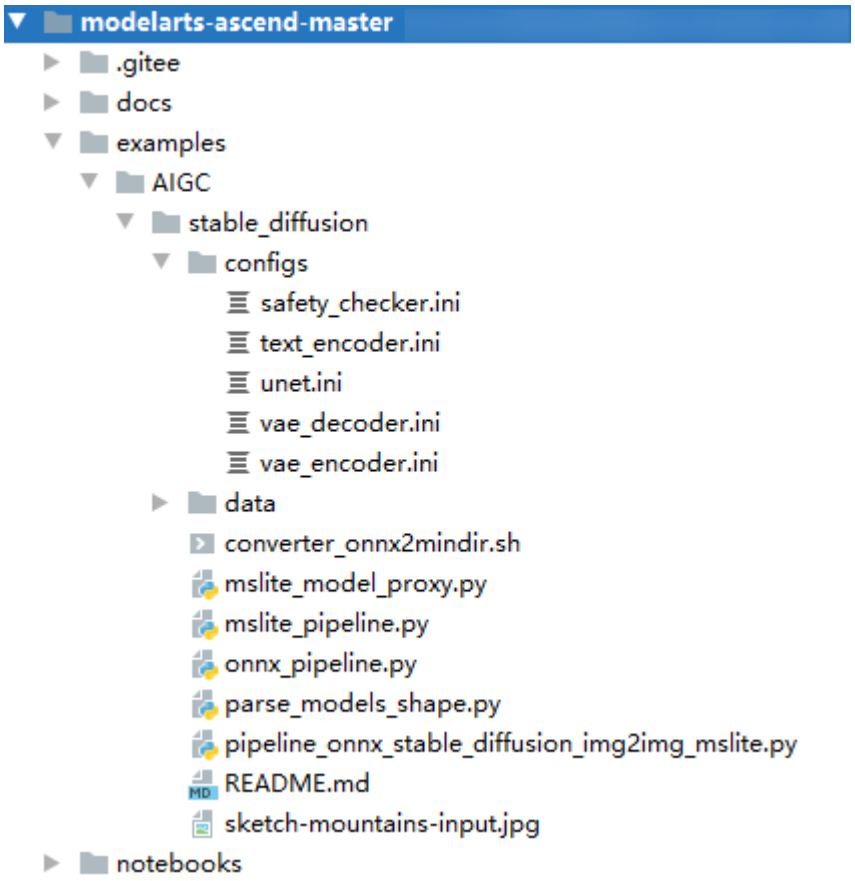
此处由于Huggingface网站的限制以及模型文件的大小原因，很可能会下载失败。您可以登录**Huggingface网站**，从浏览器下载模型后，再手动上传到物理机/home/onnx_models目录下。

步骤7 下载好模型后，需要编写推理脚本。为了便于操作，本指导中所需的代码已发布在ModelArts代码仓，可以使用如下命令下载推理脚本样例代码：

```
cd /home_host/work
git clone https://gitee.com/ModelArts/modelarts-ascend.git
ll modelarts-ascend/examples/AIGC/stable_diffusion
```

代码目录如下图所示，onnx_pipeline.py是图生图推理脚本。mslite_pipeline.py、mslite_model_proxy.py、pipeline_onnx_stable_diffusion_img2img_mslite.py是迁移后的文件，其中mslite_model_proxy.py是代理模型类，pipeline_onnx_stable_diffusion_img2img_mslite.py是从Stable Diffusion源码中的pipeline复制并修改的，这些文件在后续的章节中会使用并进一步介绍。

图 10-12 代码目录



步骤8 将“modelarts-ascend/examples/AIGC/stable_diffusion/onnx_pipeline.py”文件中的“onnx_model_path”改为**步骤6**中下载的onnx_models地址“/home_host/work/runwayml/onnx_models”。执行推理脚本进行测试，此处使用的推理硬件是CPU。由于CPU执行较慢，验证待迁移的代码可能需要大约15分钟左右才能完成。

```
cd modelarts-ascend/examples/AIGC/stable_diffusion # 必须执行该命令，否则会报错找不到sketch-mountains-input.jpg
python onnx_pipeline.py
```

生成的图片fantasy_landscape.png会保存在当前路径下，该图片也可以作为后期精度校验的一个对比。

图 10-13 生成图片

----结束

10.3.4 应用迁移

10.3.4.1 模型适配

MindSpore Lite是华为自研的推理引擎，能够最大化地利用昇腾芯片的性能。在使用MindSpore Lite进行离线推理时，需要先将模型转换为mindir模型，再利用MindSpore Lite作为推理引擎，将转换后的模型直接运行在昇腾设备上。模型转换需要使用converter_lite工具。

Huggingface提供的onnx模型文件的输入是动态shape，而mindir不支持动态shape，只能使用静态shape或者几个固定档位的分档shape代替。使用converter_lite转换模型时，也分为静态shape和分档shape两种方式，需要根据具体的业务需求使用对应的转换方式。本次迁移使用的是静态shape方式进行模型转换。

获取模型 shape

由于在后续模型转换时需要知道待转换模型的shape信息，此处指导如何通过训练好的stable diffusion PyTorch模型获取模型shape，主要有如下两种方式获取：

- 方式一：通过stable diffusion的PyTorch模型获取模型shape。

- 方式二：通过查看**ModelArts-Ascend代码仓库**，根据每个模型的configs文件获取已知的shape大小。

下文主要介绍如何通过方式一获取模型shape。

步骤1 在**pipeline应用准备**章节，已经下载到sd的PyTorch模型（ /home_host/work/runwayml/pytorch_models ）。进入工作目录：

```
cd /home_host/work
```

步骤2 新建Python脚本文件“parse_models_shape.py”用于获取shape。其中，model_path是指上面下载的pytorch_models的路径。

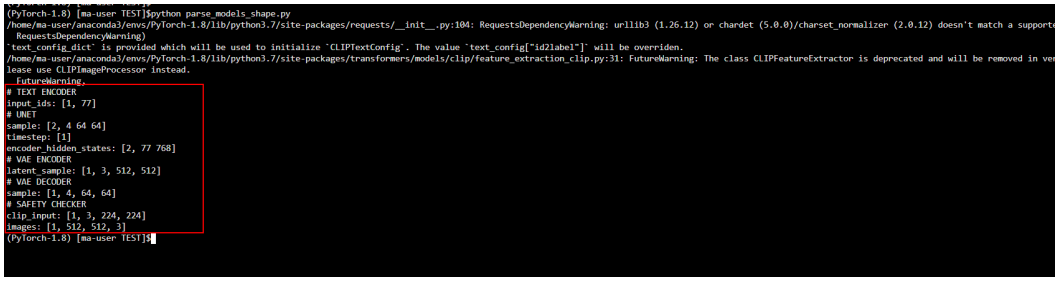
```
# parse_models_shape.py
import torch
import numpy as np
from diffusers import StableDiffusionPipeline

model_path = '/home_host/work/runwayml/pytorch_models'
pipeline = StableDiffusionPipeline.from_pretrained(model_path, torch_dtype=torch.float32)

# TEXT ENCODER
num_tokens = pipeline.text_encoder.config.max_position_embeddings
text_hidden_size = pipeline.text_encoder.config.hidden_size
text_input = pipeline.tokenizer(
    "A sample prompt",
    padding="max_length",
    max_length=pipeline.tokenizer.model_max_length,
    truncation=True,
    return_tensors="pt",
)
print("# TEXT ENCODER")
print(f"input_ids: {np.array(text_input.input_ids.shape).tolist()}")

# UNET
unet_in_channels = pipeline.unet.config.in_channels
unet_sample_size = pipeline.unet.config.sample_size
print("# UNET")
print(f"sample: [{2}, {unet_in_channels} {unet_sample_size} {unet_sample_size}]")
print(f"timestep: [{1}]")  # 此处应该是1，否则和后续的推理脚本不一致。
print(f"encoder_hidden_states: [{2}, {num_tokens} {text_hidden_size}]")

# VAE ENCODER
vae_encoder = pipeline.vae
vae_in_channels = vae_encoder.config.in_channels
vae_sample_size = vae_encoder.config.sample_size
print("# VAE ENCODER")
print(f"sample: [{1}, {vae_in_channels}, {vae_sample_size}, {vae_sample_size}]")

# VAE DECODER
vae_decoder = pipeline.vae
vae_latent_channels = vae_decoder.config.latent_channels
vae_out_channels = vae_decoder.config.out_channels
print("# VAE DECODER")
print(f"latent_sample: [{1}, {vae_latent_channels}, {unet_sample_size}, {unet_sample_size}]")

# SAFETY CHECKER
safety_checker = pipeline.safety_checker
clip_num_channels = safety_checker.config.vision_config.num_channels
clip_image_size = safety_checker.config.vision_config.image_size
print("# SAFETY CHECKER")
print(f"clip_input: [{1}, {clip_num_channels}, {clip_image_size}, {clip_image_size}]")
print(f"images: [{1}, {vae_sample_size}, {vae_sample_size}, {vae_out_channels}]")
```

步骤3 执行以下命令获取shape信息。

```
python parse_models_shape.py
```

可以看到获取的shape信息如下图所示。

图 10-14 shape 信息



----结束

PyTorch 模型转换为 Onnx 模型（可选）

获取onnx模型有以下两种方式。下文介绍如何通过方式一进行操作。如果采用方式二，可以跳过此步骤。

方式一：使用官方提供的模型转换脚本将PyTorch模型转换为onnx模型。

方式二：对于提供了onnx模型的仓库，可以直接下载onnx模型。

步骤1 通过git下载diffusers对应版本的源码。

```
git clone https://github.com/huggingface/diffusers.git -b v0.11.1
```

步骤2 在diffusers的script/convert_stable_diffusion_checkpoint_to_onnx.py脚本中，可以通过执行以下命令生成onnx模型。其中，model_path指定PyTorch的模型根目录，output_path指定生成的onnx模型目录。

```
cd /home_host/work
python diffusers/scripts/convert_stable_diffusion_checkpoint_to_onnx.py --model_path "./runwayml/
pytorch_models"  --output_path "./pytorch_to_onnx_models"
```

----结束

静态 shape 模型转换

转换静态shape模型需要在模型转换阶段固定模型的输入shape，也就是说每个输入shape是唯一的。静态shape转换主要包括两种场景：

- 第一种是待转换onnx模型的输入本身已经是静态shape，此时不需要在转换时指定输入shape也能够正常转换为和onnx模型输入shape一致的mindir模型。
- 第二种是待转换onnx模型的输入是动态shape（导出onnx模型时指定了dynamic_axes参数），此时需要在转换时明确指定输入的shape。

转换时指定输入shape可以在命令行中指定，也可以通过配置文件的形式进行指定。

- 在命令行中指定输入shape。

  命令行可以直接通过--inputShape参数指定输入的shape，格式为“input_name:input_shape”，如果有多个输入，需要使用“;”隔开，例如“input1_name:input1_shape;input2_name:input2_shape”。

  ```
  converter_lite --modelFile=./text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR --
  optimize=ascend_oriented --outputFile=./text_encoder --inputShape="input_ids:1,77"
  ```

- 在配置文件中指定输入shape。

  配置文件中通过“[ascend_context]”配置项指定input_shape，格式与命令行一致，多个输入，需要使用“;”隔开。然后在命令行中通过--configFile指定对应的配置文件路径即可。

```
# text_encoder.ini

[ascend_context]
input_shape=input_ids:[1,77]
```

转换命令如下：

```
converter_lite --modelFile=./text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR --
optimize=ascend_oriented --outputFile=./text_encoder --configFile=./text_encoder.ini
```

在使用converter_lite工具转换时，默认是将所有算子的精度转换为fp16。如果想要将固定shape的模型精度修改为fp32进行转换，需要在配置文件中指定算子的精度模式为precision_mode，配置文件的写法如下（更多精度模式请参考**precision_mode**）：

```
# text_encoder.ini
[ascend_context]
input_shape=input_ids:[1,77]
precision_mode=enforce_fp32
```

对于本次AIGC迁移，为了方便对多个模型进行转换，可以通过批量模型转换脚本自动完成所有模型的转换。

步骤1 执行以下命令，创建并进入static_shape_convert目录。

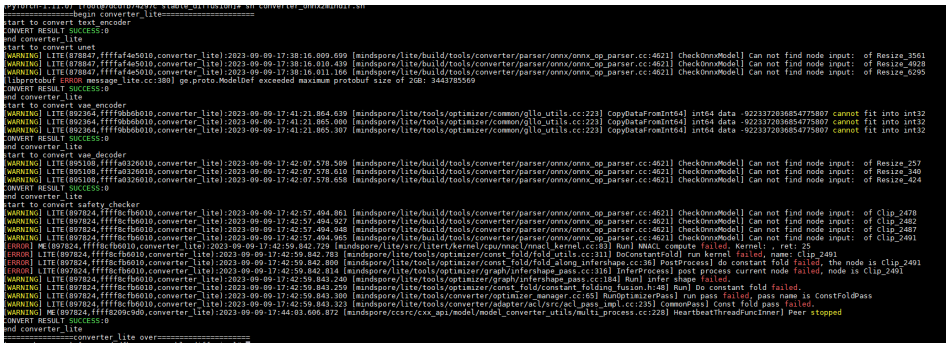
```
mkdir -p /home_host/work/static_shape_convert
cd /home_host/work/static_shape_convert
```

步骤2 在static_shape_convert目录下新建converter_onnx2mindir.sh文件并复制下面内容。其中，onnx_dir表示onnx模型的目录，mindir_dir指定要生成的mindir模型的保存目录。

```
# converter_onnx2mindir.sh
# 设置onnx模型和mindir模型目录。
onnx_dir=/home_host/work/runwayml/onnx_models
mindir_dir=./mindir_models

# 指定配置文件路径。
config_dir=/home_host/work/modelarts-ascend/examples/AIGC/stable_diffusion/configs

echo "================begin converter_lite====================="

sub_cmd='--fmk=ONNX --optimize=ascend_oriented --saveType=MINDIR'
mkdir -p $mindir_dir
# rm缓存,慎改。
atc_data_dir=/root/atc_data/
# 通用转换方法。
common_converter_model() {
    model_name=$1
    echo "start to convert $model_name"
    rm -rf $atc_data_dir
    converter_lite --modelFile="$onnx_dir/$model_name/model.onnx" \
                --outputFile="$mindir_dir/$model_name" \
                --configFile="$config_dir/$model_name.ini" \
                $sub_cmd
    printf "end converter_lite\n"
}
common_converter_model "text_encoder"
common_converter_model "unet"
common_converter_model "vae_encoder"
common_converter_model "vae_decoder"
common_converter_model "safety_checker"

echo "================converter_lite over====================="
```

转换结果如下，其中safety_checker模型转换成功，但中间有ERROR日志，该ERROR属于常量折叠失败，不影响结果。

图 10-15 转换结果



----结束

动态分档模型转换（可选）

说明

如果迁移的模型有多个shape档位的需求，可以通过如下方式对模型进行分档转换。

动态分档是指将模型输入的某一维或者某几维设置为“动态”可变，但是需要提前设置可变维度的“档位”范围。即转换得到的模型能够在指定的动态轴上使用预设的几种shape（保证模型支持的shape），相比于静态shape更加灵活，且性能不会有劣化。

动态分档模型转换需要使用配置文件，指定输入格式为“ND”，并在config文件中配置ge.dynamicDims和input_shape使用，在input_shape中将输入shape的动态维度设为-1，并在ge.dynamicDims中指定动态维度的档位，更多配置项可以参考**官方文档**。

- 如果网络模型只有一个输入：每个档位的dim值与input_shape参数中的-1标识的参数依次对应，input_shape参数中有几个-1，则每档必须设置几个维度。

  以text_encoder模型为例，修改配置文件text_encoder.ini如下所示：

  ```
  # text_encoder.ini

  [acl_build_options]
  input_format="ND"
  input_shape="input_ids:1,-1"
  ge.dynamicDims="77;33"
  ```

  使用上述配置文件转换得到的模型，支持的输入shape为(1,77)和(1,33)。

  然后使用converter lite执行模型转换，转换命令如下：

  ```
  converter_lite --modelFile=./onnx_models/text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR
  --optimize=ascend_oriented --outputFile=./mindirs --configFile=./configs/text_encoder.ini
  ```

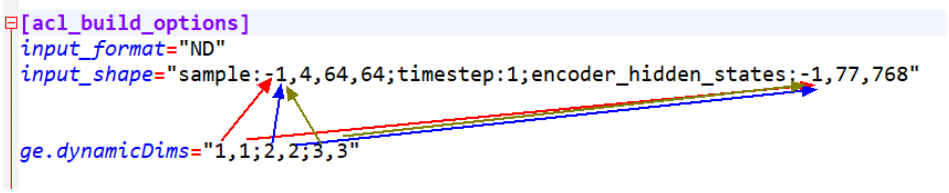
- 如果网络模型有多个输入：档位的dim值与网络模型输入参数中的-1标识的参数依次对应，网络模型输入参数中有几个-1，则每档必须设置几个维度。

  以unet模型为例，该网络模型有三个输入，分别为“sample(1,4,64,64)”、“timestep(1)”、“encoder_hidden_states(1,77,768)”，修改unet.ini配置文件如下所示：

  ```
  # unet.ini

  [acl_build_options]
  input_format="ND"
  input_shape="sample:-1,4,64,64;timestep:1;encoder_hidden_states:-1,77,768"
  ge.dynamicDims="1,1;2,2;3,3"
  ```

  转换得到的模型支持的输入dims组合档数分别为：

图 10-16 组合档数



第0档：sample(1,4,64,64) + timestep(1) + encoder_hidden_states(1,77,768)

第1档：sample(2,4,64,64) + timestep(1) + encoder_hidden_states(2,77,768)

第2档：sample(3,4,64,64) + timestep(1) + encoder_hidden_states(3,77,768)

然后使用converter lite执行模型转换，转换命令如下：

```
converter_lite --modelFile=./onnx_models/unet/model.onnx --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented --outputFile=./mindirs --configFile=./configs/unet.ini
```

说明

- 最多支持100档配置，每一档通过英文逗号分隔。
- 如果用户设置的dim数值过大或档位过多，可能会导致模型编译失败，此时建议用户减少档位或调低档位数值。
- 如果用户设置了动态维度，实际推理时，使用的输入数据的shape需要与设置的档位相匹配。

10.3.4.2 pipeline 代码适配

onnx pipeline的主要作用是将onnx模型进行一系列编排，并在onnx Runtime上按照编排顺序执行。因此，需要将转换得到的mindir模型按照相同的逻辑进行编排，并在MindSpore Lite上执行。只需要将原始onnx的pipeline中涉及到onnx模型初始化及推理的接口替换为MindSpore Lite的接口即可。

MindSpore Lite提供了Python、C++以及JAVA三种应用开发接口。此处以Python接口为例，介绍如何使用MindSpore Lite Python API构建并推理Stable Diffusion模型，更多信息请参考**MindSpore Lite应用开发**。

以官方onnx pipeline代码为例，其提供的onnx pipeline代码路径在“**${diffusers}/pipelines/stable_diffusion/pipeline_onnx_stable_diffusion_img2img.py**”，其中${diffusers}表示diffusers包的安装路径，可以通过pip进行查看。

```
# shell
pip show diffusers
```

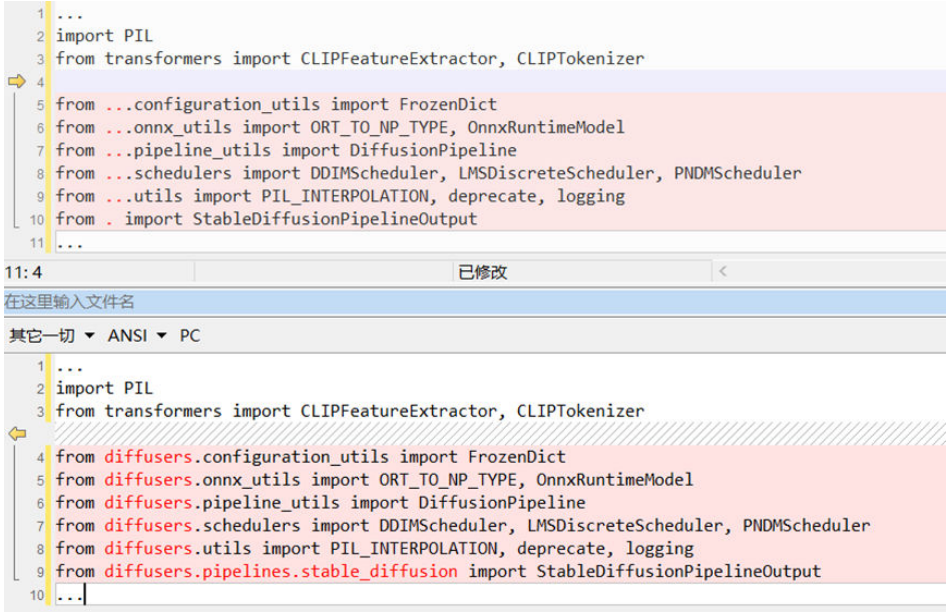
修改代码依赖

新建并进入/home_host/work/pipeline目录。

```
mkdir -p /home_host/work/pipeline
cd /home_host/work/pipeline
```

将onnx pipeline依赖的图生图源码“pipeline_onnx_stable_diffusion_img2img.py”复制到该目录下，名称改为“pipeline_onnx_stable_diffusion_img2img_mslite.py”，以便与源文件名称区分。但是这样也会导致无法正确找到源码中相对路径下的依赖，需要将对于diffusers包内的相对路径修改为绝对路径的形式。

图 10-17 代码依赖修改前与修改后



将推理代码“modelarts-ascend/examples/AIGC/stable_diffusion/onnx_pipeline.py”也复制一份到该目录，名称改为“mslite_pipeline.py”，迁移后的推理代码中的pipeline需要修改为从复制的onnx pipeline文件导入：

```
# onnx_pipeline.py
from pipeline_onnx_stable_diffusion_img2img_mslite import OnnxStableDiffusionImg2ImgPipeline
```

模型初始化

使用MindSpore Lite进行推理时一般需要先设置目标设备的上下文信息，然后构建推理模型，获取输入数据，模型预测并得到最终的结果。一个基础的推理框架写法如下所示：

```
# base_mslite_demo.py
import mindspore_lite as mslite

# 设置目标设备上下文为Ascend，指定device_id为0。
context = mslite.Context()
context.target = ["ascend"]
context.ascend.device_id = 0
# 构建模型。
model = mslite.Model()
model.build_from_file("./model.mindir", mslite.ModelType.MINDIR, context)

# 输入数据到Device侧，针对于多输入场景可以通过list来指定输入。
in_data = [np.array(data1), np.array(data2)]
inputs = model.get_inputs()
for i, _inputs in enumerate(inputs):
    _input.set_data_from_numpy(in_data[i])
# 前向推理，并将结果从device侧传到host侧。
outputs = model.predict(inputs)
outputs = [output.get_data_to_numpy() for output in outputs]
# 后处理...
```

为了同时兼容onnx模型和mindir模型都能够在适配后的pipeline中运行，需要对于Model进行封装。MsliteModel各参数模型说明已给出，根据模型初始化参数设置当前模型使用onnx模型（运行在CPU上）或mindir模型（运行在昇腾设备上），也能够方便进行精度的校验。

```
# mslite_model_proxy.py
import os
import mindspore_lite as mslite
class MsliteModel:
    def __init__(self, model_path, model_name='ms model', device_type='ascend', use_ascend=True,
                 onnx_runtime_model=None, get_shape=False, resize_shape=False) -> None:
        """
        mindir模型代理类
        Args:
            model_path: mindir文件路径
            model_name: 模型名称
            device_type: 设备类型
            use_ascend: 是否使用Ascend
            onnx_runtime_model: onnx模型对象
            get_shape: 是否获取模型shape信息、输入数据shape信息
            resize_shape: resize shape开关，分档模型需开启
        """
        print('model_path:{}'.format(model_path))
        self.model_name = model_name
        self.context = MsliteModel.init_context(device_type)
        self.model = mslite.Model()
        self.model.build_from_file(model_path, mslite.ModelType.MINDIR, self.context)
        self.ms_inputs = self.model.get_inputs()
        self.use_ascend = use_ascend
        self.onnx_runtime_model = onnx_runtime_model
        self.get_shape = get_shape
        self.resize_shape = resize_shape
    @staticmethod
    def init_context(device_type='ascend'):
        context = mslite.Context()
        context.target = [device_type]
        context.ascend.device_id = int(os.getenv('DEVICE_ID') or 0)
        context.cpu.thread_num = 1 if device_type == 'ascend' else 32
        context.cpu.thread_affinity_mode = 2
        return context
    def __call__(self, **kwargs):
        if not self.use_ascend:
            return self.onnx_runtime_model(**kwargs)
        inputs = list(kwargs.values())
        if len(inputs) <= 0:
            raise Exception('get tensor input info failed')
        ms_input = self.model.get_inputs()
        if self.get_shape:
            print(f'{self.model_name} shape info:')
            for index, val in enumerate(self.model.get_inputs()):
                print(f'{self.model_name}: param{index} shape -> {val.shape}')
            shapes = [list(input.shape) for input in inputs]
            print(f"inputs: input_shape -> {shapes}")
        if self.resize_shape:
            self.model.resize(ms_input, [list(input.shape) for input in inputs])
        for index, val in enumerate(ms_input):
            val.set_data_from_numpy(inputs[index])
        outputs = self.model.predict(ms_input)
        outputs = [output.get_data_to_numpy() for output in outputs]
        return outputs
```

适配MindSpore Lite Runtime到onnx pipeline，首先需要初始化MindSpore LiteModel对象，通过在OnnxStableDiffusionImg2ImgPipeline中增加mindir模型初始化函数，然后在pipeline类的__init__方法调用该函数，在pipeline初始化的时候直接初始化模型。您可以参照如下样例，通过修改use_ascend修改该模型是否使用mindir运行，也可以编写代码通过环境变量指定。

```
# pipeline_onnx_stable_diffusion_img2img_mslite.py

class OnnxStableDiffusionImg2ImgPipeline(DiffusionPipeline):
    ...
    def mslite_modules_init(self):
        self.text_encoder_ms = MsliteModel(
            model_path=os.environ['TEXT_ENCODER_PATH'],
```

```
        model_name='text_encoder', use_ascend=True,
        onnx_runtime_model=self.text_encoder)
    self.vae_encoder_ms = MsliteModel(
        model_path=os.environ['VAE_ENCODER_PATH'],
        model_name='vae_encoder', use_ascend=True,
        onnx_runtime_model=self.vae_encoder)
    self.unet_ms = MsliteModel(model_path=os.environ['UNET_PATH'],
                               model_name='unet', use_ascend=True,
                               onnx_runtime_model=self.unet)
    self.vae_decoder_ms = MsliteModel(model_path=os.environ['VAE_DECODER_PATH'],
                                      model_name='vae_decoder', use_ascend=True,
                                      onnx_runtime_model=self.vae_decoder)
    self.safety_checker_ms = MsliteModel(model_path=os.environ['SAFETY_CHECKER_PATH'],
                                         model_name='safety_checker', use_ascend=True,
                                         onnx_runtime_model=self.safety_checker)
...
```

模型推理适配

完成模型初始化后，您需要将onnx模型推理的代码等价替换为对应的mindir模型推理接口。以vae_encoder模型为例，在pipeline代码中查找vae_encoder推理调用的地方，然后修改为对应的MindSpore Lite版本的推理接口模型。

- 使用MindSpore Lite Runtime接口替换onnx Runtime接口。

  ```
  # pipeline_onnx_stable_diffusion_img2img_mslite.py
  ...
  # onnx模型
  # init_latents = self.vae_encoder(sample=image)[0]
  # ----------------修改点-----------------
  # mslite模型
  init_latents = self.vae_encoder_ms(sample=image)[0]
  ...
  ```

- 替换内嵌模型。

  ```
  # pipeline_onnx_stable_diffusion_img2img_mslite.py
  ...
  # onnx模型
  # image = np.concatenate([self.vae_decoder(latent_sample=latents[i : i + 1])[0] for i in
  range(latents.shape[0])])
  # ----------------修改点-----------------
  # mslite模型
  image = np.concatenate([self.vae_decoder_ms(latent_sample=latents[i : i + 1])[0] for i in
  range(latents.shape[0])])
  ...
  ```

修改后的文件参考**Gitee代码库**中的如下两个文件：

- pipeline_onnx_stable_diffusion_img2img_mslite.py
- mslite_model_proxy.py

运行 pipeline 代码

pipeline代码如下：

```
# mslite_pipeline.py
import os

import requests
import torch
import numpy as np
from PIL import Image
from io import BytesIO

from pipeline_onnx_stable_diffusion_img2img_mslite import OnnxStableDiffusionImg2ImgPipeline

def setup_seed(seed):
```

```
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    torch.backends.cudnn.deterministic = True

setup_seed(0)

# 指定mindir和onnx模型路径。
mindir_dir = "/home_host/work/static_shape_convert/mindir_models"
onnx_model_path = "/home_host/work/runwayml/onnx_models"

os.environ['DEVICE_ID'] = "0"
os.environ['TEXT_ENCODER_PATH'] = f"{mindir_dir}/text_encoder.mindir"
os.environ['VAE_ENCODER_PATH'] = f"{mindir_dir}/vae_encoder.mindir"
os.environ['UNET_PATH'] = f"{mindir_dir}/unet_graph.mindir"
os.environ['VAE_DECODER_PATH'] = f"{mindir_dir}/vae_decoder.mindir"
os.environ['SAFETY_CHECKER_PATH'] = f"{mindir_dir}/safety_checker.mindir"
pipe = OnnxStableDiffusionImg2ImgPipeline.from_pretrained(onnx_model_path,
      torch_dtype=torch.float32).to("cpu")
url = "https://raw.githubusercontent.com/CompVis/stable-diffusion/main/assets/stable-samples/img2img/
sketch-mountains-input.jpg"
response = requests.get(url, verify=False)
init_image = Image.open(BytesIO(response.content)).convert("RGB")
init_image = init_image.resize((512, 512))

prompt = "A fantasy landscape, trending on artstation"
images = pipe(prompt=prompt, image=init_image, strength=0.75, guidance_scale=7.5).images
images[0].save("fantasy_landscape_npu.png")
```

在运行pipeline时，默认的加速卡为0号卡，当机器有多人使用时，可能存在资源占用而无法正常运行的情况，可以通过环境变量指定加速卡ID，如指定5号卡进行执行。

```
# mslite_pipeline.py
...
os.environ['DEVICE_ID'] = "5"
...
```

最后执行python脚本进行推理：

```
# shell
python mslite_pipeline.py
```

图 10-18 执行推理脚本

图 10-19 MindSpore Lite pipeline 输出的结果图片

10.3.5 迁移效果校验

在pipeline适配完成后，需要验证适配后的效果是否满足要求，通过对比原始onnx pipeline的最终输出结果确认迁移效果。如果精度和性能都没有问题，则代表迁移完成。

对比图片生成效果

在CPU上推理onnx，将原始onnx和适配完成的MindSpore Lite pipeline输出的结果图片进行对比，在这里保证输入图片及文本提示词一致。如果差异较为明显可以进行**模型精度调优**。

确认性能是否满足要求

在推理代码开始结尾处加入时间记录，并打印出推理执行耗时。根据用户需求判断性能是否满足要求，如果不满足可以进行**性能调优**。

```
import time
start_time = time.time()
# 推理代码
end_time = time.time()
print(f"infer time {end_time - start_time}")
```

10.3.6 模型精度调优

10.3.6.1 场景介绍

本小节通过一个具体问题案例，介绍模型精度调优的过程。

如下图所示，使用MindSpore Lite生成的图像和onnx模型的输出结果有明显的差异，因此需要对MindSpore Lite pipeline进行精度诊断。

图 10-20 结果对比

onnx pipeline　　MindSpore Lite pipeline

说明

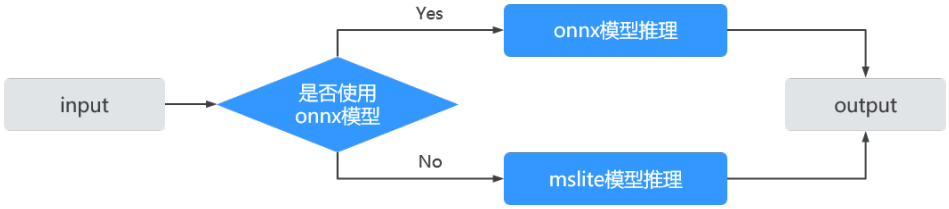
在MindSpore Lite 2.0.0版本中，Stable Diffusion的五个模型的精度都能够保证一致性，但是在最新的2.1.0版本中，会出现text_encoder模型精度不一致的情况。该问题后续会发布补丁进行修复。

10.3.6.2 精度问题诊断

逐个替换模型，检测有问题的模型

该方式主要是通过模型替换，先定位出具体哪个模型引入的误差，进一步诊断具体的模型中哪个算子或者操作导致效果问题，模型替换原理如下图所示。通过设置开关选项（是否使用onnx模型），控制模型推理时，模型使用的是onnx模型或是mindir的模型。

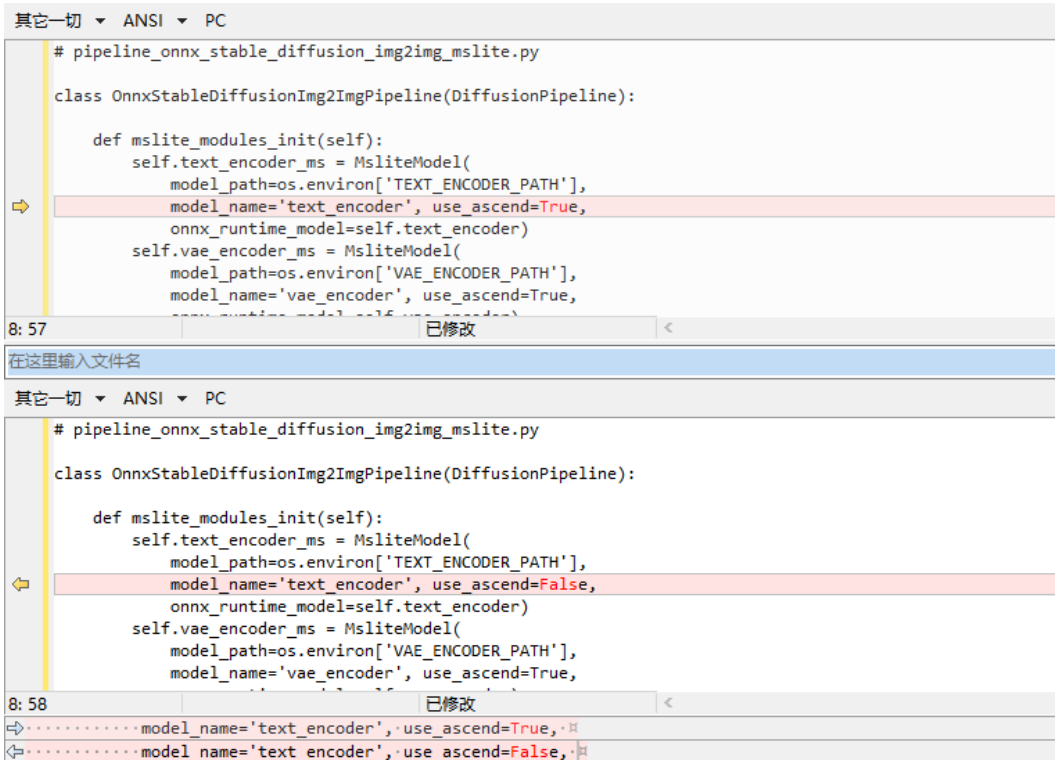
图 10-21 精度诊断流程



一般情况下，onnx模型推理的结果可以认为是标杆数据，单独替换某个onnx模型为MindSpore Lite模型，运行得到的结果再与标杆数据做对比，如果没有差异则说明pipeline的差异不是由当前替换的MindSpore Lite模型引入。

如果有差异，则说明当前模型与原始onnx的结果存在差异。依次单独替换onnx模型为对应的MindSpore Lite模型，从而定位出有差异的模型。在模型初始化的代码块已经添加了use_ascend参数，修改参考如下：

图 10-22 代码修改



以上述现象为例，通过修改use_ascend参数值对模型替换，可以发现：当text_encoder模型为onnx模型，其余模型为mindir模型时，能够得到和标杆数据相同的输出，因此可以判断出转换得到的text_encoder模型是产生pipeline精度误差的根因。通过下一小节可以进一步确认模型精度的差异。

10.3.6.3 精度问题处理

设置高精度并重新转换模型

在转换模型时，默认采用的精度模式是fp16，如果转换得到的模型和标杆数据的精度差异比较大，可以使用fp32精度模式提升模型的精度（精度模式并不总是需要使用fp32，因为相对于fp16，fp32的性能较差。因此，通常只在检测到某个模型精度存在问题时，才会考虑是否使用fp32进行尝试）。使用fp32精度模式的配置文件如下：

配置文件：

```
# config.ini
[ascend_context]
precision_mode=enforce_fp32 # 使用fp32。
```

其他方式

需要实际分析算子层面的差异，需要联系华为工程师进行具体分析。

10.3.7 性能调优

10.3.7.1 单模型性能测试工具 Mindspore lite benchmark

在模型精度对齐后，针对Stable Diffusion模型性能调优，您可以通过AOE工具进行自助性能调优，进一步可以通过profiling工具对于性能瓶颈进行分析，并针对性地做一些调优操作。

您可以直接使用benchmark命令测试mindir模型性能，用来对比调优前后性能是否有所提升。

```
# shell
cd /home_host/work
benchmark --modelFile=diffusers/scripts/mindir_models/text_encoder.mindir --device=Ascend
```

上述命令中：modelFile指定生成的mindir模型文件；device指定运行推理的设备。其他用法请参考**benchmark文档**。

测试结果如下所示：

图 10-23 测试结果

```
[root@6861cf0c12ba work]# benchmark --modelFile=diffusers/scripts/mindir_models/text_encoder.mindir --device=Ascend
ModelPath = diffusers/scripts/mindir_models/text_encoder.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 10
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 3
NumThreads = 2
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 2357.9 ms
Running warm up loops...
Running benchmark loops...
Model = text_encoder.mindir, NumThreads = 2, MinRunTime = 3.974000 ms, MaxRuntime = 3.995000 ms, AvgRunTime = 3.982000 ms
Run Benchmark text_encoder.mindir Success.
```

10.3.7.2 单模型性能调优 AOE

使用**AOE**工具可以在模型转换阶段对于模型运行和后端编译过程进行执行调优。请注意AOE只适合静态shape的模型调优。在AOE调优时，容易受当前缓存的一些影响，建议分两次进行操作，以达到较好的优化效果（第一次执行生成AOE的知识库，在第二次使用时可以复用）。在该场景中，AOE对text_encoder等模型提升效果不大，性能主要瓶颈点在unet模型中，主要对unet模型做调优，整体的操作步骤如下：

步骤1 转换前先清理缓存，避免转换时的影响。

```
# shell
# 删除已有的aoe知识库，或者备份一份。
rm -rf /root/Ascend/latest/data/aoe
# 删除编译缓存。
rm -rf /root/atc_data/*
```

步骤2 新建并进入AOE工作目录。

```
mkdir -p /home_host/work/aoe
cd /home_host/work/aoe
```

步骤3 在配置文件中启用AOE自动调优。

配置unet.ini，开启aoe调优（ aoe_mode + op_select_impl_mode ）。

```
# unet.ini
[ascend_context]
input_shape=sample:[2,4,64,64];timestep:[1];encoder_hidden_states:[2,77,768]
input_format=NCHW

aoe_mode="subgraph tuning, operator tuning"
op_select_impl_mode=high_performance
```

配置打印ASCEND日志。其中，ASCEND_GLOBAL_LOG_LEVEL的值对应的日志级别分别为：0-debug、1-info、2-warning、3-error。

```
# shell
export ASCEND_GLOBAL_LOG_LEVEL=1
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

模型转换时指定AOE调优配置文件。

```
# shell
# 模型转换时指定AOE调优配置文件并将调优日志输出到aoe_unet.log。
mkdir aoe_output
converter_lite --modelFile=/home_host/work/runwayml/onnx_models/unet/model.onnx --outputFile=./
aoe_output/aoe_unet --configFile=unet.ini  --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented >
aoe_unet.log
```

启动AOE调优后，模型转换时长会延长到数小时，因为其中包含了AOE的转化过程耗时较长。您也可以指定调优时间，一般情况下时间越长效果会越好，一般10h以内即可，推荐在后台执行。调优完成后，默认将AOE生成的知识库保存在“/root/Ascend/latest/data/aoe”路径下，同时会在aoe_output路径下输出对应的mindir模型，由于当前模型并没有吸收知识库信息，所以性能不佳，因此需要在保留AOE知识库的情况下，再次进行转换，以达到较优性能。

步骤4 删除编译缓存atc_data。

说明

注意相比第一次清除缓存操作，本次保留了AOE知识库。

```
#shell
# 删除编译缓存。
rm -rf /root/atc_data/*
```

步骤5 再次执行模型转换命令，确保AOE能够命中知识库。

配置config.ini，关闭AOE调优：

```
# unet.ini
[ascend_context]
input_shape=sample:[2,4,64,64];timestep:[1];encoder_hidden_states:[2,77,768]
input_format=NCHW
```

再次执行模型转换命令（ 此次运行关闭了AOE，速度会变快 ）：

```
#shell
converter_lite --modelFile=/home_host/work/runwayml/onnx_models/unet/model.onnx --outputFile=./
aoe_output/aoe_unet --configFile=unet.ini  --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented >
aoe_unet2.log
```

此时，aoe_output下面会有对应的mindir模型，包含了AOE知识库信息。使用benchmark工具测试新生成的mindir模型性能，同AOE调优前的模型进行对比，可以看到模型性能有所提升。

```
#shell
# 调优前命令如下：
benchmark --modelFile=/home_host/work/static_shape_convert/mindir_models/unet_graph.mindir --
device=Ascend --numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --
loopCount=100

# 调优后命令如下：
```



```
benchmark --modelFile=/home_host/work/aoe/aoe_output/aoe_unet_graph.mindir --device=Ascend --
numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --loopCount=100
```

图 10-24 调优前模型

```
[root@6861cf0c12ba aoe]# benchmark --modelFile=/home_host/work/static_shape_convert/mindir_models/unet_graph.mindir --device=Ascend --numThreads=1 --parallelNum
=1 --workersNum=1 --warmUpLoopCount=100 --loopCount=100
ModelPath = /home_host/work/static_shape_convert/mindir_models/unet_graph.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 100
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 100
NumThreads = 1
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 7786.27 ms
Running warm up loops...
Running benchmark loops...
Model = unet_graph.mindir, NumThreads = 1, MinRunTime = 72.867996 ms, MaxRuntime = 77.177002 ms, AvgRunTime = 74.184998 ms
Run Benchmark unet_graph.mindir Success.
```

图 10-25 调优后模型

```
[root@6861cf0c12ba aoe]# benchmark --modelFile=/home_host/work/aoe/aoe_output/aoe_unet_graph.mindir --device=Ascend --numThreads=1 --parallelNum=1 --workersNum=
1 --warmUpLoopCount=100 --loopCount=100
ModelPath = /home_host/work/aoe/aoe_output/aoe_unet_graph.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 100
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 100
NumThreads = 1
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 5907.39 ms
Running warm up loops...
Running benchmark loops...
Model = aoe_unet_graph.mindir, NumThreads = 1, MinRunTime = 44.772999 ms, MaxRuntime = 46.356998 ms, AvgRunTime = 45.570999 ms
Run Benchmark aoe_unet_graph.mindir Success.
```

AOE优化成功的mindir已经融合了优化的知识库，是一个独立可用的模型。即使AOE知识库删除，不影响该mindir的性能。可以备份这个模型优化产生的知识库，以后需要的话再使用。

----结束

10.3.8 常见问题

- **模型转换失败怎么办?**

  常见的模型转换失败原因可以通过查询转换失败错误码来确认具体导失败的原因。Stable Diffusion新推出的模型在转换中可能会遇到算子不支持的问题，您可以到华为云管理页面上提交工单来寻求帮助。

- **图片大Shape性能劣化严重怎么办?**

  在昇腾设备上，可能由于GPU内存墙导致在大shape下遇到性能问题。MindSporeLite提供了Flash Attention编译优化机制，您可以考虑升级最新版本的MindSporeLite Convertor来进行编译器的算子优化，在大Shape场景下会有明显的改善。

- **同样功能的PyTorch Pipeline，因为指导要求适配onnx pipeline，两个pipeline本身功能就有差别，如何适配?**

  由于Diffusers社区的“single model file policy”设计原则，不同的pipeline是不同路径在独立演进的。请先确保应用输出符合预期后，再进入到MindSpore Lite模型转换的过程，否则迁移昇腾后还是会遇到同样的问题。

- **AOE的自动性能调优使用上完全没有效果怎么办?**

  在MindSpore Lite Convertor2.1版本之前可能出现的调优不生效的场景，建议您直接使用MindSpore Lite Convertor2.1及以后的版本。配置文件指定选项进行

AOE调优。使用转换工具配置config参数，具体如下所示，其中“subgraph tuning”表示子图调优，“operator tuning”表示算子调优。

其中，“ge.op_compiler_cache_mode”在该场景下必须设置为“force”，表示该场景下要强制刷新缓存，保证AOE调优后的知识库能够命中，实现模型调优。

```
# config.ini

[ascend_context]
aoe_mode="subgraph tuning, operator tuning"
[acl_init_options]
ge.op_compiler_cache_mode="force"
```

- **迁移后应用出图效果相比GPU无法对齐怎么办?**

  扩散模型在噪音和随机数上的生成，本身就有一定的随机性，GPU和NPU（Ascend）硬件由于存在一定细小的差别，很难确保完全一致，较难达成生成图片100%匹配，建议通过盲测的方式对效果进行验证。

- **模型精度有问题怎么办?**

  首先考虑通过FP16的方式进行转换和执行，再通过精度诊断工具来进行分析，更进一步可以到华为云官网上提交工单处理。

- **模型转换失败时如何查看日志和定位原因?**

  在模型转换的过程，如果出现模型转换失败，可以参考以下步骤查看日志并定位原因：

  a. 设置DEBUG日志。

  b. 设置MindSpore日志环境变量。

  ```
  # shell
  export GLOG_v=0      # 0-DEBUG、1-INFO、2-WARNING、3-ERROR
  ```

  c. 设置CANN日志环境变量。

  ```
  # shell
  export ASCEND_GLOBAL_LOG_LEVEL=1          # 0：表示DEBUG、1：表示INFO、2：表示WARNING、3：表示ERROR 4: 表示NONE
  export ASCEND_SLOG_PRINT_TO_STDOUT=1      # 表示日志打印。
  ```

  d. 设置DUMP模型转换中间图。

  e. 设置DUMP中间图环境变量。

  ```
  # shell
  export DUMP_GE_GRAPH=2            # 1：表示dump图全量内容。 2：表示不dump权重数据的基础图。 3：表示只dump节点关系的精简图。
  export DUMP_GRAPH_LEVEL=2         # 1：表示dump图所有图。 2：表示dump除子图外的所有图。 3：表示只dump最后一张图。
  ```

  f. 问题分析。

  g. 配置以上的环境变量之后，再重新转换模型，导出对应的日志和dump图进行分析：

     i. 报错日志中搜到“not support onnx data type”，表示MindSpore暂不支持该算子。

     ii. 报错日志中搜到“Convert graph to om failed”，表示CANN模块进行图编译存在保存，需要结合CANN的报错日志和dump图进行具体分析。

- **Stable Diffusion WebUI如何适配?**

  WebUI一般可以分为前端和后端实现两部分，后端的实现模式种类多样，并且依赖了多个的第三方库，当前在WebUI适配时，并没有特别好的方式。在对后端实现比较理解的情况下，建议针对具体的功能进行Diffusers模块的适配与替换，然后针对替换上去的Diffusers，对其pipeline进行昇腾迁移适配，进而替代原有WebUI的功能。针对很多参数以及三方加速库（如xformers）的适配，当前没有特别好的处理方案。

- **LoRA适配流是怎么样的?**

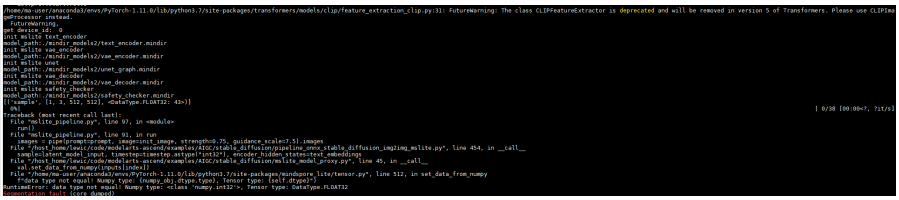
  因为现在PyTorch-npu推理速度比较慢（固定shape比mindir慢4倍），在现在pth-onnx-mindir的模型转换方式下，暂时只能把lora合并到unet主模型内，在每次加载模型前lora特性就被固定了（无法做到PyTorch每次推理都可以动态配置的能力）。

  目前临时的静态方案可参考**sd-scripts**， 使用其中的“networks/merge_lora.py”把lora模型合入unet和text-encoder模型。

- **数据类型不匹配问题如何处理?**

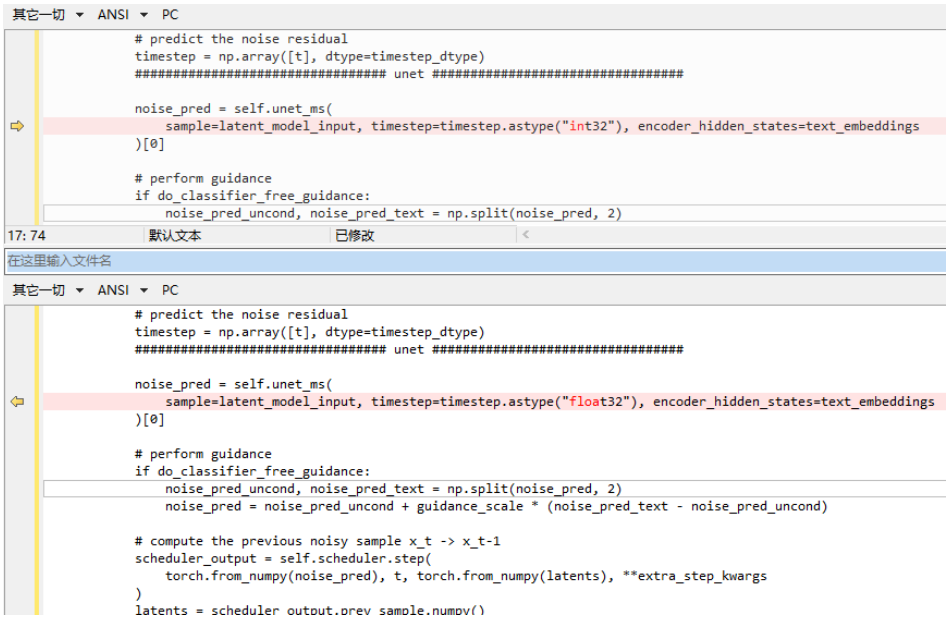
  报错“data type not equal”时，按照堆栈信息，将对应的行数的数据类型修改为匹配的类型。

图 10-26 报错信息



处理该问题时，pipeline_onnx_stable_diffusion_img2img_mslite.py文件的第454行修改如下：

图 10-27 修改内容



10.4 GPU 推理业务迁移至昇腾的通用指导

10.4.1 简介

场景介绍

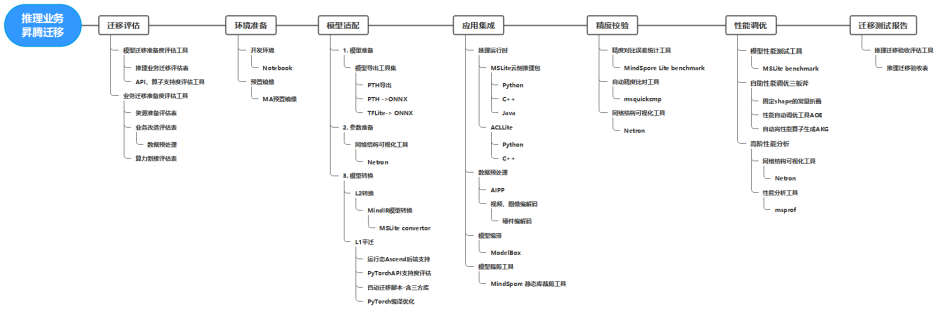
本文旨在指导客户将已有的推理业务迁移到昇腾设备上运行（单机单卡、单机多卡），并获得更好的推理性能收益。

ModelArts针对上述使用场景，在给出系统化推理业务昇腾迁移方案的基础上，提供了即开即用的云上集成开发环境，包含迁移所需要的算力资源和工具链，以及具体的Notebook代码运行示例和最佳实践，并对于实际的操作原理和迁移流程进行说明，包含迁移后的精度和性能验证、调试方法说明。

核心概念

- **推理业务昇腾迁移整体流程及工具链**

图 10-28 推理业务昇腾迁移整体路径



推理业务昇腾迁移整体分为七个大的步骤，并以完整工具链覆盖全链路：

a. 迁移评估：针对迁移可行性、工作量，以及可能的性能收益进行大致的预估。
b. 环境准备：利用ModelArts提供的开发环境一键式准备好迁移、调测需要的运行环境与工具链。
c. 模型适配：针对昇腾迁移模型必要的转换和改造。
    i. 模型准备，导出和保存确定格式的模型。
    ii. 转换参数准备，准备模型业务相关的关键参数。
    iii. 模型转换，包含模型转换、优化和量化等。
d. 应用集成。
    i. 针对转换的模型运行时应用层适配。
    ii. 数据预处理。
    iii. 模型编排。
    iv. 模型裁剪。
e. 精度校验。
    i. 精度对比误差统计工具。
    ii. 自动化精度对比工具。
    iii. 网络结构可视化工具。

f. 性能调优。

i. 性能测试。

ii. 性能调优三板斧。

iii. 性能分析与诊断。

g. 迁移测试报告。

i. 推理迁移验收表。

- **ModelArts开发环境**

  ModelArts作为华为云上的AI开发平台，提供交互式云上开发环境，包含标准化昇腾算力资源和完整的迁移工具链，帮助用户完成昇腾迁移的调测过程，进一步可在平台上将迁移的模型一键部署成为在线服务向外提供推理服务，或者运行到自己的运行环境中。

- **MindSpore Lite**

  华为自研的AI推理引擎，后端对于昇腾有充分的适配，模型转换后可以在昇腾上获得更好的性能，配合丰富的适配工具链，降低迁移成本，该工具在推理迁移工作的预置镜像已安装，可在镜像中直接使用（见**环境准备**）。关于MindSpore Lite详细介绍可参考**MindSpore Lite文档**。在使用MindSpore Lite过程中遇到问题时，可参考MindSpore Lite官网提供的**问题定位指南**进行问题定位。

迁移路线介绍

当前推理迁移时，不同的模型类型可能会采取不同的迁移技术路线。主要分为以下几类：

1. CV类小模型例如yolov5，以及部分AIGC场景的模型迁移，目前推荐使用MindSpore-Lite推理路线，可以利用MindSpore提供的图编译和自动调优能力，达到更好的模型性能。
2. LLM大语言模型场景，在GPU下通常会使用vLLM等大模型推理框架，因此迁移到昇腾时，推荐使用PyTorch + ascend-vllm技术路线进行迁移。

如果您使用的模型在上述案例文档中已包含，建议您直接使用案例中迁移好的模型，如果您的模型不在已提供的范围内，或者您因业务要求需要自行完成端到端的迁移，可以参考本迁移指导书介绍的步骤进行操作。

本文的迁移指导及快速入门案例均针对路线1也即MindSpore-Lite迁移路线进行介绍。使用ascend-vllm路线的迁移指导会在后续提供，您可以从上面的案例中下载相关代码并直接参考实现源码。

10.4.2 昇腾迁移快速入门案例

ModelArts提供了两个昇腾迁移案例，方便您快速了解并完成昇腾迁移过程。

约束限制

当前仅贵阳一区域支持选择本案例中的规格及镜像。

操作步骤

步骤1 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”管理页面。

步骤2 在“Notebook”页面，单击“创建Notebook”。

步骤3 在“创建Notebook”页面，配置相关信息，单击“立即创建”，确认信息无误后，单击“提交”。

部分配置项说明如下。关于配置项的更多信息，请参见**创建Notebook实例**。

表 10-5 配置项说明

配置项	说明
镜像	选择“公共镜像”，然后选择“mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b”。
类型	选择“ASCEND”。
实例规格	选择snt9b资源。
存储配置	选择“云硬盘EVS”。
磁盘规格	按照对应的存储使用情况选择存储大小。
SSH远程开发	如果需通过VS Code远程连接Notebook实例，可打开SSH远程开发，并选择自己的密钥对。

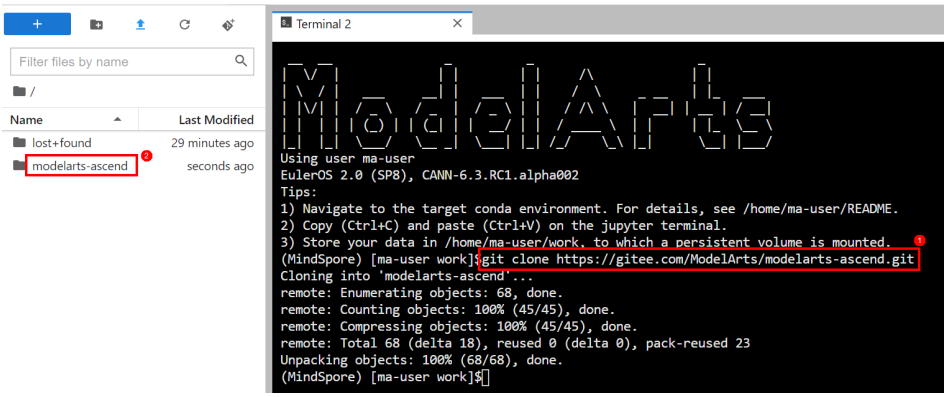
步骤4 在Notebook列表，单击“操作”列的“打开”，打开Notebook实例。

步骤5 克隆ModelArts Ascend代码库。

新建Terminal，执行下述命令将对应的repo克隆到Notebook实例。

```
git clone https://gitee.com/ModelArts/modelarts-ascend.git
```

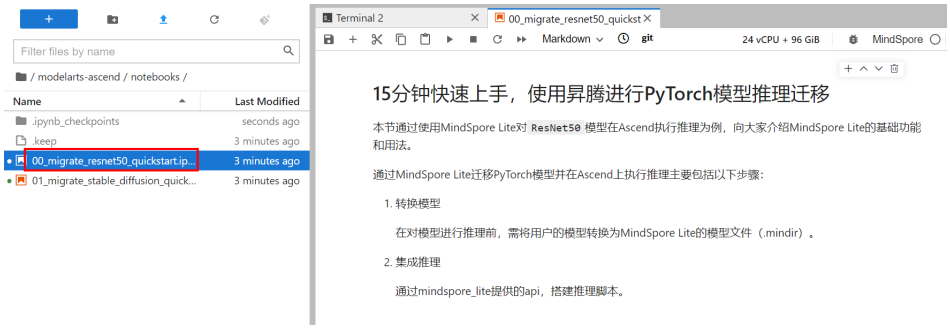
图 10-29 下载示例代码



步骤6 昇腾迁移案例在“~/work/modelarts-ascend/notebooks/”路径下，打开对应的“.ipynb”案例后运行即可。

- **ResNet50模型迁移到Ascend上进行推理**：通过使用MindSpore Lite对ResNet50模型在Ascend执行推理为例，向大家介绍MindSpore Lite的基础功能和用法。

图 10-30 ResNet50 模型迁移到 Ascend 上进行推理



- **Stable Diffusion模型迁移到Ascend上进行推理**：介绍如何将Stable Diffusion模型通过MSLite进行转换后，迁移在昇腾设备上运行。

图 10-31 Stable Diffusion 模型迁移到 Ascend 上进行推理

----结束

10.4.3 迁移评估

推理迁移包括模型迁移、业务迁移、精度性能调优等环节，是否能满足最终的迁移效果需要进行系统的评估。如果您仅需要了解迁移过程，可以先按照本文档的指导进行操作并熟悉迁移流程。如果您有实际的项目需要迁移，建议填写下方的推理业务迁移评估表，并将该调研表提供给华为云技术支持人员进行迁移评估，以确保迁移项目能顺利实施。

通用的推理业务及LLM推理可提供下表进行业务迁移评估：

表 10-6 通用的推理业务及 LLM 推理业务迁移评估表

收集项	说明	实际情况（请填写）
项目名称	项目名称，例如：XXX项目。	-
使用场景	例如： • 使用YOLOv5算法对工地的视频流裁帧后进行安全帽检测。 • 使用BertBase算法对用户在app上购买商品后的评论进行理解。	-

收集项	说明	实际情况（请填写）
CPU架构	X86/ARM，自有软件是否支持ARM。 例如：4个推理模型在ARM上运行，6个推理模型在X86上运行。	-
当前使用的操作系统及版本	当前推理业务的操作系统及版本，如：Ubuntu 22.04。 是否使用容器化运行业务，以及容器中OS版本，HostOS中是否有业务软件以及HostOS的类型和版本。 需要评估是否愿意迁移到华为云的通用OS。	-
AI引擎及版本	当前引擎（TF/PT/LibTorch），是否接受切换MindSpore。 例如：当前使用TF 2.6，PyTorch 1.10，可以接受切换MindSpore。	-
业务编程语言、框架、版本。	C++/Python/JAVA等。 例如：业务逻辑使用JAVA，推理服务模块使用C++自定义实现推理框架，Python 3.7等。	-
CPU使用率	业务中是否有大量使用CPU的代码，以及日常运行过程中CPU的占用率（占用多少个核心），以及使用CPU计算的业务功能说明和并发机制。	-
是否有Linux内核驱动	是否有业务相关的Linux内核驱动代码。	-
依赖第三方组件列表	当前业务依赖的第三方软件列表（自行编译的第三方软件列表）。 例如：Faiss等。	-
推理框架	TensorRT/Triton/MSLite等。 例如： ● 2个推理模型使用TensorRT框架，5个使用Triton框架。 ● 通过stable-diffusion的WebUI提供AIGC推理服务。	-
GPU卡的类型	Vnt1/Ant1/Ant03/Tnt004等。 例如： 20卡Ant1，运行Bert Large推理。 10卡Tnt004运行YOLOv5。	-

收集项	说明	实际情况（请填写）
Backbone类型	ResNet/DarkNet/Transformer等。 例如： ● 5个模型使用ResNet Backbone，应用与监控。 ● 3个模型使用Transformer，应用于自然语言处理xxx。 ● 使用stable-diffusion的典型模型：TextEncoder、VaeEncoder、unet、VaeDecoder、SafetyChecker，没有使用LoRA等动态加载的诉求。	-
模型训练方式	关于推理业务中使用的模型，填写该模型训练时使用的框架以及套件。 例如：模型使用PyTorch+Megatron+DeepSpeed进行训练。	-
自定义算子	是否有自定义算子，CPU还是CUDA，复杂程度。 例如：有5个CUDA自定义算子。1个高复杂度算子，基于C++开发2000行代码。4个中等复杂度算子，基于C++开发，平均每个自定义算子约500行代码。	-
动态shape	是否需要支持动态shape。 例如：需要动态Shape，需要动态Shape的模型有ResNet-50、YOLOv5。	-
参数类型（FP32/FP16）	FP32还是FP16混合，判断精度调优难度。 例如：ResNet-50、YOLOv5模型使用FP16。BertLarge使用FP32。	-
模型变更频率	模型变更场景如下： ● 数据增量，模型算子未变更。 ● 数据增量，模型算子变化，例如： – 网络结构变化。 – AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
是否使用华为MDC产品	如果使用华为MDC产品，请填写MDC版本号，如果没有可以不填。 例如：使用了C83版本。	-

收集项	说明	实际情况（请填写）
性能指标与预期	• 例1： 模型：YOLOv5 运行环境：Vnt1 单卡 性能指标：QPS 100/s （两进程） 性能约束：单次请求最大可以接受时延需小于100ms 性能预期：QPS 130/s • 例2： 模型：OCR 运行环境：6348（单核48U超线程） 性能指标：QPS 10/s（四进程） 性能约束：单次请求最大可以接受时延需小于1s 性能预期：QPS 20/s	-
业务访问方式	推理业务访问：“客户端 -> 云服务” 或 “云客户端 -> 云服务”。 推理业务时延要求，客户端到云服务端到端可接受时延。 例如：当前是“客户端 -> 云服务”模式，客户端请求应答可接受的最长时延为2秒。	-
模型参数规模，是否涉及分布式推理	10B/100B，单机多卡推理。	-
能否提供实际模型、网络验证的代码和数据等信息	提供实际模型、网络验证的代码和数据。 提供与业务类型类似的开源模型，例如GPT3 10B/13B。 提供测试模型以及对应的Demo代码路径（开源或共享）。 可以提前的完成POC评估，例如框架、算子支持度，以及可能的一些性能指标。	-

如果是AIGC场景的业务例如Stable Diffusion，请在上表的基础上，再提供以下信息：

表 10-7 AIGC 场景业务迁移评估表

收集项	说明	实际情况（请填写）
使用场景	例如： 1. 业务是文生图，图生图等。 2. 业务是否需要频繁更新模型，或者需要动态加载Lora。	-
stable-diffusion套件	1. 使用diffusers （ https://github.com/huggingface/diffusers ）。 2. stable-diffusion-webui （ https://github.com/AUTOMATIC1111/stable-diffusion-webui ）。 3. 如果是基于其他开源，需要附带开源代码仓地址。	-
具体使用库	例如： 1. 使用了哪个pipeline （ 例如lpw_stable_diffusion.py ）。 2. 使用了哪个huggingface的模型 （ 例如digiplay/majicMIX_realistic_v6 ）。 3. 如果有预处理，后处理，对应的模型是什么（ 例如后处理的超分模型 ）。	-
Lora/TextInversion	1. 是否有动态加载Lora的需求，可否接受把Lora固定到模型内。 2. 是否使用了TextInversion，是否需要动态加载。	-
动态shape	是否可接受分档shape（ 固定n个挡位的shape ）。	-
模型变更频率	模型变更场景如下： 1. 数据增量，模型算子未变更。 2. 数据增量，模型算子变化，例如： ● 网络结构变化。 ● AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
尺寸要求	超分前产生的图片尺寸要求： 1. 512*512 2. 720*720 3. 1080 *1080 4. 1920*1920 （ shape过大可能导致性能下降 ）	-

10.4.4 环境准备

迁移环境简介

ModelArts开发环境针对推理昇腾迁移的场景提供了云上可以直接访问的开发环境，具有如下优点：

- 利用云服务的资源使用便利性，可以直接使用到不同规格的昇腾设备。
- 通过指定对应的运行镜像，可以直接使用预置的、在迁移过程中所需的工具集，且已经适配到最新的版本可以直接使用。
- 开发者可以通过浏览器入口以Notebook方式访问，也可以通过VSCode远程开发的模式直接接入到云上环境中完成迁移开发与调测，最终生成适配昇腾的推理应用。

当前支持以下两种迁移环境搭建方式：

- ModelArts Standard：在Notebook中，使用预置镜像进行。
- ModelArts Lite DevServer：在裸金属服务器中 ，自助配置好存储、安装固件、驱动、配置网络等。

ModelArts Standard

ModelArts上昇腾规格如下。

表 10-8 昇腾规格

规格名称	描述
Ascend 1*ascend-snt9b|ARM 24核 192GB	Snt9b单卡规格，配搭ARM处理器，适合深度学习场景下的模型训练和调测

ModelArts提供了面向推理迁移工作的预置镜像，其中包含了最新商用版驱动、昇腾软件开发库，迁移工具链等。预置镜像可以做到即开即用，用户也可以基于预置镜像构建自定义环境内容。

ModelArts支持的昇腾迁移预置镜像如下：

表 10-9 预置镜像

区域	镜像名称
贵阳一	mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b
贵阳一	mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b
贵阳一	pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b

可通过如下方式接入Notebook开发环境进行调试。

- JupyterLab：在ModelArts管理控制台，直接打开Notebook示例的方式接入开发环境，详情请见**使用指导**。
- VS Code：利用ModelArts插件，实现VS Code远程连接Notebook示例完成远程开发，详情请见**使用指导**。

下文将介绍如何在ModelArts Standard上使用预置镜像创建Notebook实例。

步骤1 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”管理页面。

步骤2 在页面右上角单击“创建Notebook”，进入“创建Notebook”页面。

步骤3 在“创建Notebook”页面，配置相关信息，单击“立即创建”，确认信息无误后，单击“提交”。

部分配置项说明如下。关于配置项的更多信息，请参见**创建Notebook实例**。

表 10-10 配置项说明

配置项	说明
镜像	选择“公共镜像”，然后选择“mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b”。
类型	选择“ASCEND”。
实例规格	选择snt9b资源。
存储配置	选择“云硬盘EVS”。
磁盘规格	按照对应的存储使用情况选择存储大小。
SSH远程开发	如果需通过VS Code远程连接Notebook实例，可打开SSH远程开发，并选择自己的密钥对。

步骤4 在Notebook列表，单击“操作”列的“打开”，打开Notebook实例。

----结束

ModelArts Lite DevServer

开通裸金属服务器资源请见**DevServer资源开通**，在裸金属服务器上搭建迁移环境请见**裸金属服务器环境配置指导**。

10.4.5 模型适配

10.4.5.1 基于 MindSpore Lite 的模型转换

迁移推理业务的整体流程如下：

- **模型准备**
- **转换关键参数准备**
- **模型转换**

- **推理应用适配**

主要通过MindSpore Lite（简称MSLite）进行模型的转换，进一步通过MindSpore Runtime支持昇腾后端的能力来将推理业务运行到昇腾设备上。

模型准备

MindSpore Lite提供的模型convertor工具可以支持主流的模型格式到MindIR的格式转换，用户需要导出对应的模型文件，推荐导出为ONNX格式。

1. 如何导出ONNX模型
   - PyTorch转ONNX，操作指导请见**此处**。
   - PyTorch导出ONNX模型样例如下：

     ```
     import torch
     import torchvision
     model = torchvision.models.resnet50(pretrained=True)
     # 保存模型为ONNX格式
     torch.onnx.export(model, torch.randn(1, 3, 224, 224), "resnet50.onnx")
     ```

   - TensorFlow导出ONNX模型，操作指导请见**此处**。
2. 如何导出PTH模型

   PyTorch模型导出时需要包含模型的结构信息，需要利用jit.trace方式完成模型的导出与保存。

   ```
   # If you are instantiating the model with *from_pretrained* you can also easily set the TorchScript flag
   model = BertModel.from_pretrained("bert-base-uncased", torchscript=True)

   # Creating the trace
   traced_model = torch.jit.trace(model, [tokens_tensor, segments_tensors])
   torch.jit.save(traced_model, "traced_bert.pt")
   ```

转换关键参数准备

对应的模型转换成MindIR格式，通过后端绑定的编译形式来运行以达到更好的性能（类似静态图的运行模式），所以需要提前准备以下几个重点参数。

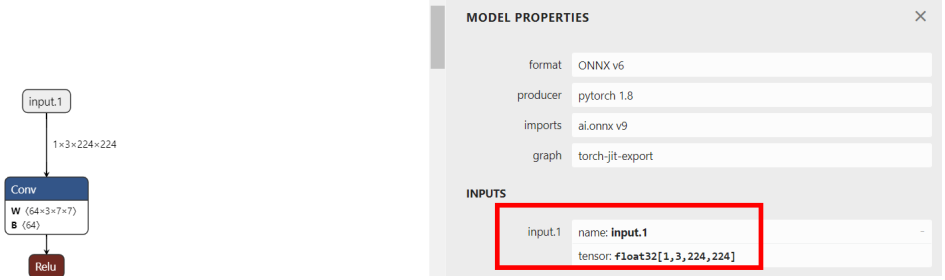
1. 输入的inputShape，包含batch信息。

   MSLite涉及到编译优化的过程，不支持完全动态的权重模式，需要在转换时确定对应的inputShape，用于模型的格式的编译与转换，可以在**netron官网**进行查看，或者对于模型结构中的输入进行shape的打印，并明确输入的batch。

   一般来说，推理时指定的inputShape和用户的业务及推理场景紧密相关，可以通过原始模型推理脚本或者网络模型进行判断。需要把Notebook中的模型下载到本地后，再放入netron官网中，查看其inputShape。

   如果netron中没有显示inputShape，可能由于使用了动态shape模型导致，请确保使用的是静态shape模型。静态shape模型文件导出方法请参考**模型准备**。

   图 10-32 netron 中查看 inputShape



2. 精度选择。

   精度选择需要在模型转换阶段进行配置，执行converter_lite命令时通过--configFile参数指定配置文件路径，配置文件通过**precision_mode**参数指定精度模式。可选的参数有“enforce_fp32”，“preferred_fp32”，“enforce_fp16”，“enforce_origin”或者“preferred_optimal”，默认为“enforce_fp16”。

   ```
   [ascend_context]
   precision_mode= preferred_fp32
   ```

模型转换

在ModelArts开发环境中，通过对应的转换预置镜像，直接执行对应的转换过程，对应的转换和评估工具都已经预置了最新版本，详细介绍请见**使用说明**。inputShape查看方法请见**转换关键参数准备**。

```
!converter_lite --modelFile=resnet50.onnx --fmk=ONNX --outputFile=resnet50 --saveType=MINDIR --inputShape="input.1:1,3,224,224" --device=Ascend
```

为了简化用户使用，ModelArts提供了Tailor工具便于用户进行模型转换，具体使用方式参考**Tailor指导文档**。

推理应用适配

MindSpore Lite提供了JAVA/C++/Python API，进行推理业务的适配，并且在构建模型时，通过上下文的参数来确定运行时的具体配置，例如运行后端的配置等。下文以Python接口为例。

使用MindSpore Lite推理框架执行推理并使用昇腾后端主要包括以下步骤：

- 创建运行上下文：创建**Context**，保存需要的一些基本配置参数，用于指导模型编译和模型执行，在昇腾迁移时需要特别指定target为“Ascend”，以及对应的device_id。

  ```
  context = mslite.Context()
  context.target = ["ascend"]
  context.ascend.device_id = 0
  ```

- 模型加载与编译：执行推理之前，需要调用**Model**的**build_from_file**接口进行模型加载和模型编译。模型加载阶段将文件缓存解析成运行时的模型。模型编译阶段会耗费较多时间所以建议Model创建一次，编译一次，多次推理。

  ```
  model = mslite.Model()
  model.build_from_file("./resnet50.mindir", mslite.ModelType.MINDIR, context)
  ```

- 输入数据：编译后的模型提供了predict接口用户执行模型推理任务，Inputs输入为List Tensor，这里的Tensor是MSLite的概念，具体的列表长度和tensor类型由转换时的InputShape来确定，由于后端指定了ascend，这些tensor都是在昇腾设备的显存中，用户需要在对应的tensor中填入数据，这些数据也会被搬移到显存中，进一步对于Inputs输入的内容进行处理。

  ```
  data = convert_img(input_image)
  in_data = [np.array(data)]
  inputs = model.get_inputs()
  for i, _input in enumerate(inputs):
  _input.set_data_from_numpy(in_data[i])
  ```

- 执行推理：使用**Model**的**predict**进行模型推理，返回值为Outputs，也是List Tensor类型，具体的长度和类别由模型定义，对应的Tensor数据由于指定了ascend后端，Output的内容在显存中，通过tensor的get_data_to_numpy方法来获取，并将数据读取到内存中使用。

  ```
  outputs = model.predict(inputs)
  outputs = [output.get_data_to_numpy() for output in outputs]
  ```

更多Python接口的高级用法与示例，请参考**Python API**。

10.4.5.2 动态 shape

在某些推理场景中，模型输入的shape可能是不固定的，因此需要支持用户指定模型的动态shape，并能够在推理中接收多种shape的输入。在CPU上进行模型转换时无需考虑动态shape问题，因为CPU算子支持动态shape；而在Ascend场景上，算子需要指定具体的shape信息，并且在模型转换的编译阶段完成对应shape的编译任务，从而能够在推理时支持多种shape的输入。

动态 batch

在模型转换阶段通过--configFile参数指定配置文件，并且在配置文件中配置input_shape及dynamic_dims动态参数。其中input_shape的-1表示动态shape所在的维度，dynamic_dims指定动态维度的取值范围，例如“[1~4],[8],[16]”表示该动态维度支持1、2、3、4、8、6共六种大小。

```
# config.ini
[ascend_context]
input_shape=input.1:[-1,3,224,224]
dynamic_dims=[1~4],[8],[16]
```

在执行convert_lite命令时，指定--configFile=config.ini即可自动编译指定的动态shape。

```
# shell
converter_lite --modelFile=resnet50.onnx --fmk=ONNX --device=Ascend --outputFile=resnet50_dynamic --
saveType=MINDIR --configFile=config.ini
```

注意：推理应用开发时，需要使用模型的Resize功能，改变输入的shape。而且Resize操作需要在数据从host端复制到device端之前执行，下面是一个简单的示例，展示如何在推理应用时使用动态Shape。

```
import mindspore_lite as mslite
import numpy as np
from PIL import Image
# 设置目标设备上下文为Ascend，指定device_id为0。
context = mslite.Context()
context.target = ["ascend"]
context.ascend.device_id = 0
# 构建模型。
model = mslite.Model()
model.build_from_file("./resnet50_dynamic.mindir", mslite.ModelType.MINDIR, context)
data = np.random.rand(8, 3, 224, 224).astype(np.float32)
inputs = model.get_inputs()
model.resize(inputs, [list(data.shape)])
inputs[0].set_data_from_numpy(data)
# 前向推理，并将结果从device侧传到host侧。
outputs = model.predict(inputs)[0].get_data_to_numpy()
print(outputs.shape)  # (8, 1000)
```

动态分辨率

动态分辨率可以用于设置输入图片的动态分辨率参数。适用于执行推理时，每次处理图片宽和高不固定的场景，该参数需要与input_shape配合使用，input_shape中-1的位置为动态分辨率所在的维度。使用方法可参考**Ascend配置文件说明**。

10.4.6 精度校验

转换模型后执行推理前，可以使用benchmark工具对MindSpore Lite云侧推理模型进行基准测试。它不仅可以对MindSpore Lite云侧推理模型前向推理执行耗时进行定量分析（性能），还可以通过指定模型输出进行可对比的误差分析（精度）。

精度测试

benchmark工具用于精度验证，主要工作原理是：固定模型的输入，通过benchmark工具进行推理，并将推理得到的输出与标杆数据进行相似度度量（余弦相似度和平均相对误差），得到模型转换后的精度偏差信息。使用benchmark进行精度比对的基本流程如下：

1. 将模型输入保存二进制文件。

```
# 数据读取，预处理
image = img_preprocess(image_path)
image = np.array(image, dtype=np.float32)
image = np.frombuffer(image.tobytes(), np.float32)
# 保存网络输入为二进制文件
image.tofile("input_data.bin")
```

2. 将基准模型的输出保存到文本文件。

   本例中输出节点名称为output_node_name，输出节点的shape为“(1, 1000)”，因此一共有两维，对应的输出文件为“output_node_name 2 1 1000”，再加上输出的值即可。

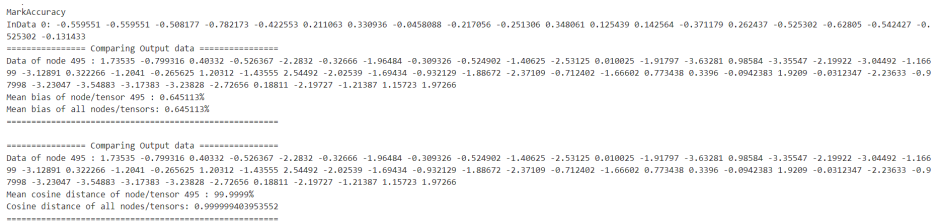
```
# 基于原始pth模型前向推理。
output = model_inference(input_data)
# 保存网络输出节点名称、维度、shape及输出到本地文件。
with open("output_data.txt", "w") as f:
f.write("output_node_name 2 1 1000\n")
f.write(" ".join([str(i) for i in output]))
```

3. 使用benchmark工具进行精度对比。

```
# shell
benchmark --modelFile=model.mindir --inputShapes=1,3,224,224 --inDataFile=input_data.bin --
device=Ascend --benchmarkDataFile=output_data.txt --accuracyThreshold=5 --
cosineDistanceThreshold=0.99
```

   其中，--accuracyThreshold=5表示平均绝对误差的容忍度最大为5%，--cosineDistanceThreshold =0.99表示余弦相似度至少为99%，--inputShapes可将模型放入到**netron官网**中查看。

图 10-33 benchmark 对接结果输出示例图



为了简化用户使用，ModelArts提供了Tailor工具便于用户进行Benchmark精度测试，具体使用方式参考**Tailor指导文档**。

10.4.7 性能调优

性能测试

benchmark工具也可用于性能测试，其主要的测试指标为模型单次前向推理的耗时。在性能测试任务中，与精度测试不同，并不需要用户指定对应的输入（inDataFile）和输出的标杆数据（benchmarkDataFile），benchmark工具会随机生成一个输入进行推理，并统计推理时间。执行的示例命令行如下。

```
# shell
benchmark --modelFile=resnet50.mindir --device=Ascend
```

为了简化用户使用，ModelArts提供了Tailor工具便于用户进行Benchmark性能测试，具体使用方式参考**Tailor指导文档**。

在某些推理场景中，模型输入的shape可能是不固定的，因此需要支持用户指定模型的动态shape，并能够在推理中接收多种shape的输入。在CPU上进行模型转换时无需考虑动态shape问题，因为CPU算子支持动态shape；而在昇腾场景上，算子需要指定具体的shape信息，并且在模型转换的编译阶段完成对应shape的编译任务，从而能够在推理时支持多种shape的输入。

绝大多数情况下，昇腾芯片推理性能相比于CPU会好很多，但是也可能会遇到和CPU推理性能并无太大差别甚至出现劣化的情况。造成这种情况的原因可能有如下几种：

1. 模型中存在大量的类似于Pad或者Strided_Slice等算子，其在CPU和Ascend上的实现方法存在差异（硬件结构不同），后者在运算此类算子时涉及到数组的重排，性能较差；
2. 模型的部分算子在昇腾上不支持，或者存在Transpose操作，会导致模型切分为多个子图，整体的推理耗时随着子图数量的增多而增长；
3. 模型没有真正的调用昇腾后端，而是自动切换到了CPU上执行，这种情况可以通过输出日志来进行判断。

自助性能调优三板斧

基于上一步完成的性能测试，为了最大化模型推理性能，首先确保当前使用的CANN版本是最新版本（最新版本请见**此处**），每个迭代的CANN版本都有一定的性能收益。在此基础上，可以进行三板斧自助工具式性能调优。这些调优过程由大量的项目交付经验总结，帮助您获得模型最佳推理性能，重复**性能测试**章节可以验证对应的收益情况。

自助性能调优三板斧分别为：**通过固定shape获取更好的常量折叠**、**AOE性能自动调优**、**自动高性能算子生成工具**。

- **通过固定shape获取更好的常量折叠**

  在MindIR格式转换时（即执行converter_lite命令时），通过指定具体的静态shape，并且打开--optimize参数指定“ascend_oriented”能够获得更好的常量折叠优化效果。inputShape查看方法请见**转换关键参数准备**。

  ```
  Ascend Optimization Engine
  converter_lite --modelFile=resnet50.onnx --fmk=ONNX --outputFile=resnet50 --saveType=MINDIR --
  inputShape="input.1:1,3,224,224" --optimize=ascend_oriented
  ```

  常量折叠是编译器优化中的通用技术之一，在编译节点简化常量表达。通过多数的现代编译器不会真的产生两个乘法的指令再将结果存储下来，取而代之的是会识别出语句的结构，并在编译时期将数值计算出来而不是运行时去计算（在本例子，结果为2,048,000）。

  ```
  i = 320 * 200 * 32;
  ```

AI编译器中，常量折叠是将计算图中预先可以确定输出值的节点替换成常量，并对计算图进行一些结构简化的操作，例如ADDN操作，以及在推理过程中的batch normalization操作等。

以BN折叠为例，如下表示折叠后获得的性能收益。

图 10-34 BN 折叠下前向运算性能收益

模型	CPU 前向时间	GPU 前向时间
Resnet50（合并前）	176.17ms	11.03ms
Resnet50（合并后）	161.69ms	7.3ms
提升	~8%	~34%

- **AOE性能自动调优**

  自动性能调优工具AOE(Ascend Optimization Engine)，可以对于模型的图和算子运行通过内置的知识库进行自动优化，以提升模型的运行效率。开启AOE调优后，模型转换时会自动进行性能调优操作，该过程耗时较长，可能需要数小时。

  AOE性能自动优化在模型转换阶段进行配置（即执行converter_lite命令时），通过--configFile参数指定配置文件aoe_config.ini，配置文件通过aoe_mode参数指定调优模式。可选值有：

  - “subgraph tuning”：子图调优。
  - “operator tuning”：算子调优。
  - “subgraph tuning, operator tuning”：先进行子图调优，再进行算子调优。

  推荐先进行子图调优，再进行算子调优，因为先进行子图调优会生成图的切分方式，子图调优后算子已经被切分成最终的shape了，再进行算子调优时，会基于这个最终shape去做算子调优。如果优先算子调优，这时调优的算子shape不是最终切分后的算子shape，不符合实际使用场景。

  本例同时指定了子图调优和算子调优，工具会先进行子图调优，再进行算子调优。

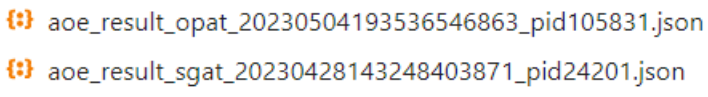
  ```
  # aoe_config.ini
  [ascend_context]
  aoe_mode="subgraph tuning, operator tuning"
  ```

  指定--configFile=aoe_config.ini即可自动进行性能优化。

  ```
  #shell
  converter_lite --modelFile=resnet50.onnx --fmk=ONNX --device=Ascend --outputFile=resnet50_aoe --saveType=MINDIR --configFile=aoe_config.ini
  ```

  命令执行成功后，性能自动优化前后的性能对比会打印到控制台上，同时会生成更为详细的json格式调优报告。

图 10-35 自动调优输出文件



需要注意的是，并不是所有的模型使用性能自动调优都是有收益的。在本例中，ResNet50模型自动调优收益甚微（模型转换时已经做了部分针对性优化），在有些比较复杂的模型场景下可能会有较好的收益。例如VAE_ENCODER模型使用算子调优收益为11.15%。

图 10-36 VAE_ENCODER 模型使用 AOE 自动调优在屏幕上显示日志

```
Start to subgraph tuning
.........[Aoe][vae_encoder_aoe] Model tuning process finished. No performance improvement.
[Aoe]Aoe process finished, cost time 152 s.


Start to operator tuning
.....................................................................................................................
.............................[Aoe][vae_encoder_aoe] Operator tuning process finished. Performance improved by 11.15%
[Aoe]Aoe process finished, cost time 2353 s.
```

图 10-37 AOE 自动调优的输出样例


```
▼ root: [] 2 items
  ▼ 0:
    ▶ basic:
  ▼ 1:
    ▼ OPAT:
        model_baseline_performance(ms): 23.059152
        model_performance_improvement: "11.15%"
        model_result_performance(ms): 20.746701
        opat_tuning_result: "tuning successful"
      ▼ repo_modified_operators:
        ▶ add_repo_operators: [] 19 items
      ▼ repo_summary:
          repo_add_num: 19
          repo_hit_num: 3
          repo_reserved_num: 3
          repo_unsatisfied_num: 5
          repo_update_num: 0
          total_num: 27
```


其中：

- model_baseline_performance表示调优前模型执行时间，单位为ms。
- model_performance_improvement表示调优后模型执行时间减少百分比。
- model_result_performance表示调优后模型执行时间。
- repo_summary中的信息表示调优过程中使用到的知识库算子个数或者追加到知识库的算子个数。

AOE自动调优更多介绍可参考**Ascend转换工具功能说明**。

- **自动高性能算子生成工具**

  自动高性能算子生成工具AKG（Auto Kernel Generator），可以对深度神经网络模型中的算子进行优化，并提供特定模式下的算子自动融合功能，可提升在昇腾硬件后端上运行模型的性能。

  AKG的配置也是在模型转换阶段进行配置（即执行converter_lite命令时），通过指定对应的配置文件akg.cfg，设置对应的akg优化级别，并且在模型转换时参考样例进行对应的配置。

  ```
  # akg.cfg
  [graph_kernel_param]
  opt_level=2
  ```

  执行命令：

  ```
  # shell
  converter_lite --fmk=ONNX --modelFile=model.onnx --outputFile=model --configFile=akg.cfg --optimize=ascend_oriented
  ```

  自动高性能算子生成工具AKG更多介绍可参考**图算融合配置说明**和**MindSpore AKG**。

10.4.8 迁移过程使用工具概览

基础的开发工具在迁移的预置镜像和开发环境中都已经进行预置，用户原则上不需要重新安装和下载，如果预置的版本不满足要求，用户可以执行下载和安装与覆盖操作。

模型自动转换评估工具 Tailor

为了简化用户使用，ModelArts提供了Tailor工具，将模型转换、精度benchmark、性能benchmark和profiling采集工具集成到同一个工具中，极大简化了用户的使用流程。建议在迁移过程中使用Tailor工具替代下面列举的原始工具MS Convertor、Benchmark和msprof。使用指导详见**AIGC工具tailor使用指导**。

模型转换工具

离线转换模型功能的工具**MSLite Convertor**，支持onnx、pth、tensorflowLite多种类型的模型转换，转换后的模型可直接运行在MindSpore运行时后端，用于昇腾推理。

精度性能检查工具

Benchmark精度检查工具，可以转换模型后执行推理前，使用其对MindSpore Lite模型进行基准测试，它不仅可以对MindSpore Lite模型前向推理执行耗时进行定量分析（性能），还可以通过指定模型输出进行可对比的误差分析（精度）。

模型自动调优工具

AOE（Ascend Optimization Engine）是一个昇腾设备上模型运行自动调优工具，作用是充分利用有限的硬件资源，以满足算子和整网的性能要求。在推理场景下使用，可以对于模型的图和算子运行内置的知识库进行自动优化，以提升模型的运行效率。

自动高性能算子生成工具 AKG

AKG（Auto Kernel Generator）对深度神经网络中的算子进行优化，并提供特定模式下的算子自动融合功能。提升在昇腾硬件后端上运行网络的性能。

AKG由三个基本的优化模块组成：规范化、自动调度和后端优化。

- 规范化： 为了解决polyhedral表达能力的局限性（只能处理静态的线性程序），需要首先对计算公式IR进行规范化。规范化模块中的优化主要包括自动运算符inline、自动循环融合和公共子表达式优化等。
- 自动调度： 自动调度模块基于polyhedral技术，主要包括自动向量化、自动切分、thread/block映射、依赖分析和数据搬移等。
- 后端优化： 后端优化模块的优化主要包括TensorCore使能、双缓冲区、内存展开和同步指令插入等。

性能分析工具

msprof命令行工具提供了采集通用命令以及AI任务运行性能数据、昇腾AI处理器系统数据、Host侧系统数据和采集和解析能力。面向推理的场景，可以对于模型的执行性能数据进行收集，可基于收集的性能数据进行性能分析。

10.4.9 常见问题

10.4.9.1 MindSpore Lite 问题定位指南

在MindSpore Lite使用中遇到问题时，例如模型转换失败、训练后量化转换失败、模型推理失败、模型推理精度不理想、模型推理性能不理想、使用Visual Studio报错、使用Xcode构建APP报错等，您可以先查看日志信息进行定位分析。

多数场景下的问题可以通过日志报错信息直接定位。如果日志的信息不能定位问题，您可以通过设置**环境变量**调整日志等级，打印更多调试日志。

关于如何对MindSpore Lite遇到的问题进行定位与解决，请参见MindSpore Lite官网提供的**问题定位指南**。

10.4.9.2 模型转换报错如何查看日志和定位?

通过如下的配置项打开对应的模型转换日志，可以看到更底层的报错。如配置以下的环境变量之后，再重新转换模型，导出对应的日志和dump图进行分析：

1. 报错日志中搜到“not support onnx data type”，表示MindSpore暂不支持该算子。
2. 报错日志中搜到“Convert graph to om failed”，表示CANN模块进行图编译存在保存，需要结合CANN的报错日志和dump图进行具体分析。

配置方式参考如下：

1. 打开DEBUG日志。
   - 设置MindSpore日志环境变量。
     ```
     export GLOG_v=0
     # 0-DEBUG、1-INFO、2-WARNING、3-ERROR
     ```
   - 设置CANN日志环境变量。
     ```
     # 0：表示DEBUG。1：表示INFO。2：表示WARNING。3：表示ERROR。4: 表示NONE。
     export ASCEND_GLOBAL_LOG_LEVEL=1
     # 表示日志打印
     export ASCEND_SLOG_PRINT_TO_STDOUT=1
     ```
2. DUMP模型转换中间图。

   设置DUMP中间图环境变量。

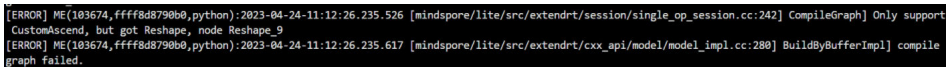
   ```
   #  1：表示dump图全量内容。2：表示不dump权重数据的基础图。3：表示只dump节点关系的精简图。
   export DUMP_GE_GRAPH=2
   #  1：表示dump图所有图。2：表示dump除子图外的所有图。3：表示只dump最后一张图。
   export DUMP_GRAPH_LEVEL=2
   ```

10.4.9.3 日志提示 Compile graph failed

问题现象

日志提示：Compile graph failed。

图 10-38 报错提示



原因分析

模型转换时未指定Ascend后端。

处理方法

需要在模型转换阶段指定“--device=Ascend”。

10.4.9.4 日志提示 Custom op has no reg_op_name attr

问题现象

日志提示：Custom op has no reg_op_name attr。

图 10-39 报错提示

原因分析

无。

处理方法

定义context时无需指定：

```
context.ascend.provider = "ge"
```

10.4.10 推理业务迁移评估表

通用的推理业务及LLM推理可提供下表进行业务迁移评估：

收集项	说明	实际情况（请填写）
项目名称	项目名称，例如：XXX项目。	-
使用场景	例如： ● 使用YOLOv5算法对工地的视频流裁帧后进行安全帽检测。 ● 使用BertBase算法对用户在app上购买商品后的评论进行理解。	-
CPU架构	X86/ARM，自有软件是否支持ARM。 例如：4个推理模型在ARM上运行，6个推理模型在X86上运行。	-

收集项	说明	实际情况（请填写）
当前使用的操作系统及版本	当前推理业务的操作系统及版本，如：Ubuntu 22.04。 是否使用容器化运行业务，以及容器中OS版本，HostOS中是否有业务软件以及HostOS的类型和版本。 需要评估是否愿意迁移到华为云的通用OS。	-
AI引擎及版本	当前引擎（TF/PT/LibTorch），是否接受切换MindSpore。 例如：当前使用TF 2.6，PyTorch 1.10，可以接受切换MindSpore。	-
业务编程语言、框架、版本。	C++/Python/JAVA等。 例如：业务逻辑使用JAVA，推理服务模块使用C++自定义实现推理框架，Python 3.7等。	-
CPU使用率	业务中是否有大量使用CPU的代码，以及日常运行过程中CPU的占用率（占用多少个核心），以及使用CPU计算的业务功能说明和并发机制。	-
是否有Linux内核驱动	是否有业务相关的Linux内核驱动代码。	-
依赖第三方组件列表	当前业务依赖的第三方软件列表（自行编译的第三方软件列表）。 例如：Faiss等。	-
推理框架	TensorRT/Triton/MSLite等。 例如： ● 2个推理模型使用TensorRT框架，5个使用Triton框架。 ● 通过stable-diffusion的WebUI提供AIGC推理服务。	-
GPU卡的类型	Vnt1/Ant1/Ant03/Tnt004等。 例如： 20卡Ant1，运行Bert Large推理。 10卡Tnt004运行YOLOv5。	-

收集项	说明	实际情况（请填写）
Backbone类型	ResNet/DarkNet/Transformer等。 例如： ● 5个模型使用ResNet Backbone，应用与监控。 ● 3个模型使用Transformer，应用于自然语言处理xxx。 ● 使用stable-diffusion的典型模型：TextEncoder、VaeEncoder、unet、VaeDecoder、SafetyChecker，没有使用LoRA等动态加载的诉求。	-
模型训练方式	关于推理业务中使用的模型，填写该模型训练时使用的框架以及套件。 例如：模型使用PyTorch+Megatron+DeepSpeed进行训练。	-
自定义算子	是否有自定义算子，CPU还是CUDA，复杂程度。 例如：有5个CUDA自定义算子。1个高复杂度算子，基于C++开发2000行代码。4个中等复杂度算子，基于C++开发，平均每个自定义算子约500行代码。	-
动态shape	是否需要支持动态shape。 例如：需要动态Shape，需要动态Shape的模型有ResNet-50、YOLOv5。	-
参数类型（FP32/FP16）	FP32还是FP16混合，判断精度调优难度。 例如：ResNet-50、YOLOv5模型使用FP16。BertLarge使用FP32。	-
模型变更频率	模型变更场景如下： ● 数据增量，模型算子未变更。 ● 数据增量，模型算子变化，例如： – 网络结构变化。 – AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
是否使用华为MDC产品	如果使用华为MDC产品，请填写MDC版本号，如果没有可以不填。 例如：使用了C83版本。	-

收集项	说明	实际情况（请填写）
性能指标与预期	• 例1： 模型：YOLOv5 运行环境：Vnt1 单卡 性能指标：QPS 100/s （两进程） 性能约束：单次请求最大可以接受时延需小于100ms 性能预期：QPS 130/s • 例2： 模型：OCR 运行环境：6348（单核48U超线程） 性能指标：QPS 10/s（四进程） 性能约束：单次请求最大可以接受时延需小于1s 性能预期：QPS 20/s	-
业务访问方式	推理业务访问：“客户端 -> 云服务” 或 “云客户端 -> 云服务”。 推理业务时延要求，客户端到云服务端到端可接受时延。 例如：当前是“客户端 -> 云服务”模式，客户端请求应答可接受的最长时延为2秒。	-
模型参数规模，是否涉及分布式推理	10B/100B，单机多卡推理。	-
能否提供实际模型、网络验证的代码和数据等信息	提供实际模型、网络验证的代码和数据。 提供与业务类型类似的开源模型，例如GPT3 10B/13B。 提供测试模型以及对应的Demo代码路径（开源或共享）。 可以提前的完成POC评估，例如框架、算子支持度，以及可能的一些性能指标。	-

如果是AIGC场景的业务例如Stable Diffusion，请在上表的基础上，再提供以下信息：

收集项	说明	实际情况（请填写）
使用场景	例如： 1. 业务是文生图，图生图等。 2. 业务是否需要频繁更新模型，或者需要动态加载Lora。	-

收集项	说明	实际情况（请填写）
stable-diffusion套件	1. 使用differusers （ https://github.com/huggingface/diffusers ）。 2. stable-diffusion-webui （ https://github.com/AUTOMATIC1111/stable-diffusion-webui ）。 3. 如果是基于其他开源，需要附带开源代码仓地址。	-
具体使用库	例如： 1. 使用了哪个pipeline （ 例如lpw_stable_diffusion.py ）。 2. 使用了哪个huggingface的模型 （ 例如digiplay/majicMIX_realistic_v6 ）。 3. 如果有预处理，后处理，对应的模型是什么（ 例如后处理的超分模型 ）。	-
Lora/TextInversion	1. 是否有动态加载Lora的需求，可否接受把Lora固定到模型内。 2. 是否使用了TextInversion，是否需要动态加载。	-
动态shape	是否可接受分档shape（ 固定n个挡位的shape ）。	-
模型变更频率	模型变更场景如下： 1. 数据增量，模型算子未变更。 2. 数据增量，模型算子变化，例如： ● 网络结构变化。 ● AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
尺寸要求	超分前产生的图片尺寸要求： 1. 512*512 2. 720*720 3. 1080 *1080 4. 1920*1920 （ shape过大可能导致性能下降 ）	-

10.5 基于 advisor 的昇腾训练性能自助调优指导

10.5.1 advisor 调优总体步骤

基于ModelArts performance advisor插件的昇腾PyTorch性能调优主要分为以下步骤：

1. **准确采集性能劣化时刻的profiling数据**。

2. 存储profiling数据。
3. 创建advisor分析环境。

操作步骤

步骤1 明确性能问题类型，准确采集性能劣化时刻的profiling数据。

1. 对于固定step出现性能劣化，如固定在16步出现性能劣化，则需要合理配置profiling参数，确保能采集到16步的数据。
2. 对于所有step稳定劣化的场景，避免采集第一个step的profiling即可，可以采集任意step如第15个step即可。
3. 对于偶现且劣化现象出现的step数不固定的场景，则需要确保能采集到该不固定的step。

profiling数据采集请参考Ascend PyTorch Profiler接口采集。文档中包含**torch_npu.profiler.profile**、**dynamic_profile**等多种采集方式。任意torch_npu版本均支持**torch_npu.profiler.profile**方式，而其他采集方式则要求特定版本的torch_npu（2024年0630之后版本）。推荐升级torch_npu后使用**dynamic_profile**方式进行采集，如果升级成本过高，也可以使用**torch_npu.profiler.profile**。

注意

当不明确性能劣化的可能原因时，profiling关键参数配置请务必复用如下设置：

```
# torch_npu.profiler._ExperimentalConfig
aic_metrics=torch_npu.profiler.AiCMetrics.PipeUtilization
profiler_level=torch_npu.profiler.ProfilerLevel.Level1
data_simplification=True

# torch_npu.profiler.profile
activities=[torch_npu.profiler.ProfilerActivity.CPU, torch_npu.profiler.ProfilerActivity.NPU]
with_stack=False
```

- **torch_npu.profiler.profile**采集方式介绍

  配置完如图10-40所示代码后需要启动训练作业触发采集且只能采集指定的step，对于已经明确需要采集step的场景可以使用该采集方式，此时需要重点关注图10-40中schedule参数以确保采集到需要的step数据。对于schedule参数，请参考图10-41。skip_first用于跳过指定的前n个step。wait、warmup和active三个参数构成了一次完整的采集，repeat参数表示重复多少次完整的采集。wait参数表示重复执行采集过程中每次采集跳过的step轮数，warmup表示预热的step轮数（推荐设置为1），active表示实际采集的连续m个step。

  参数示例：

  ```
  skip_first=10，wait=3, warmup=1, active=3, repeat=2
  ```

  采集时会先跳过前10个step（从step0开始）。然后完整的一次采集过程包括等待3个step，预热1个step和连续采集3个step（step14-step16）的profiling数据。这一次采集的step14-step16的profiling数据会保存在同一个json文件中。由于设置了repeat=2，则会再次重复一次采集过程，那么实际会二次采集step21-step23这三个连续step的profiling数据并保存至一个新的json文件中。

图 10-40 torch_npu.profiler.profile

采集并解析性能数据（torch_npu.profiler.profile）

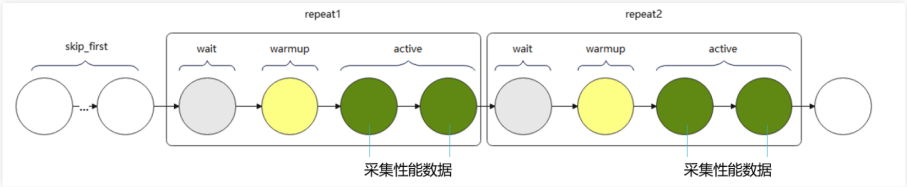
❶ 在训练脚本（如train_*.py文件）内添加如下示例代码进行性能数据采集参数配置，之后启动训练。下列示例代码中，加粗字段为需要配置的参数、方法、类和函数。

说明

以下示例代码中的torch_npu.profiler.profile接口详细介绍请参见Ascend PyTorch Profiler接口说明。

```
experimental_config = torch_npu.profiler._ExperimentalConfig(
    export_type=torch_npu.profiler.ExportType.Text,
    profiler_level=torch_npu.profiler.ProfilerLevel.Level0,
    msprof_tx=False,
    aic_metrics=torch_npu.profiler.AiCMetrics.AiCoreNone,
    l2_cache=False,
    op_attr=False,
    data_simplification=False,
    record_op_args=False,
    gc_detect_threshold=None
)

with torch_npu.profiler.profile(
    activities=[
        torch_npu.profiler.ProfilerActivity.CPU,
        torch_npu.profiler.ProfilerActivity.NPU
        ],
    schedule=torch_npu.profiler.schedule(wait=0, warmup=0, active=1, repeat=1, skip_first=1),
    on_trace_ready=torch_npu.profiler.tensorboard_trace_handler("./result"),
    record_shapes=False,
    profile_memory=False,
    with_stack=False,
    with_modules=False,
    with_flops=False,
    experimental_config=experimental_config) as prof:
        for step in range(steps):
            train_one_step(step, steps, train_loader, model, optimizer, criterion)
            prof.step()
```

图 10-41 torch_npu.profiler.profile schedule 参数释义



- **dynamic_profile**采集方式介绍

  对于上述提到的性能劣化且出现step不固定的场景，优先考虑使用动态profiling方式进行采集。如图3中所示"if step==5"处，需要在业务代码中添加如下判断逻辑：记录每一个step的耗时，如果某个step的耗时出现异常，即大于正常step耗时或者均值耗时的20%（根据训练日志的实际step耗时来确定异常耗时阈值），则认为出现性能劣化，从而执行'dp.start'触发profiling采集。

图 10-42 dynamic_profile

步骤2 存储profiling数据。

多机场景如果没有挂载共享存储如SFS Turbo，需要将多机上的profiling复制至同一个目录下才能进行性能分析，这个操作相对较为繁琐且耗时。使用ModelArts时推荐挂载共享网盘如sfs turbo，既能加快训练数据的读取速度又能用于存放性能profiling数据。如果没有共享网盘，profiling数据默认保存到ModelArts训练容器中，则请参考**创建ModelArts训练作业**中的配置训练参数部分配置好输出参数，训练过程中会自动将训练容器中输出路径下的数据回传至指定的OBS上。

步骤3 创建performance advisor分析环境。

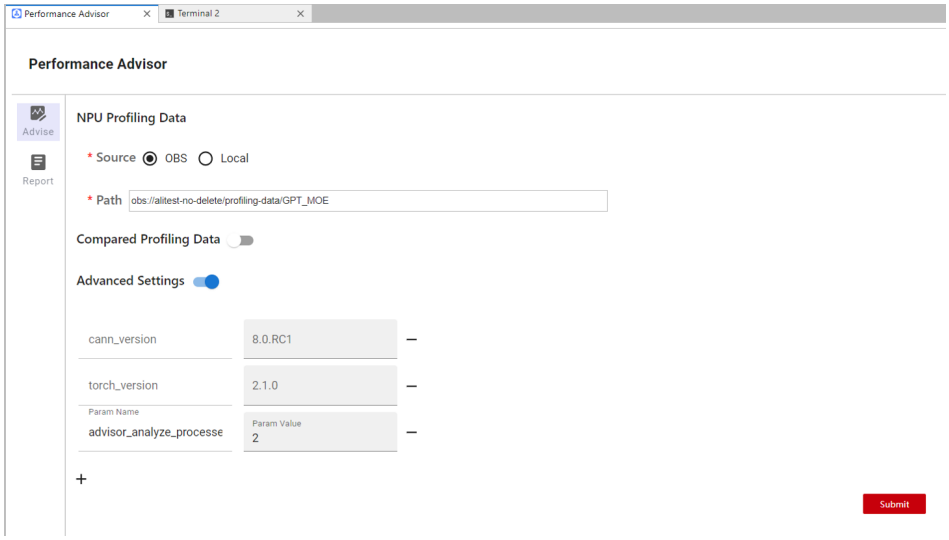
采集完profiling后如果ModelArts训练作业已经停止，则推荐参照**创建诊断任务**创建cpu规格的notebook进行性能分析，节省NPU计算资源。完成分析后，可以查看生成的html文件来进行快速的调优，html文件详情请参考**查看诊断报告**。

下面以开发环境Notebook为例介绍一个典型的性能调优案例。

64卡训练任务，模型为GPT MOE，tensor parallel(tp)为8，pipeline parallel(pp)为4。训练过程中发现每个step耗时均显著增大，基于**dynamic_profile**方式采集profiling并上传至OBS。选择任意镜像如PyTorch，创建一个2U8GB CPU规格（如果CPU资源充足，建议创建8U32G的分析环境）的notebook开发环境。在notebook中使用performance advisor插件进行性能分析，源数据选择OBS并指定profiling所在的OBS路径。由于pp参数为4，考虑到不同pp stage的计算量存在差异，advisor会自动对不同stage进行计算维度的分析，因此在Advanced Setting中设置分析进程为2（不建

议设置太大，避免占用过多CPU资源导致OOM类问题）使能并行分析，加快分析速度，如下图10-43所示。

图 10-43 基于 performance advisor 进行性能劣化分析



完成分析后单击下图图10-44中view查看报告。html（图10-45）中显示计算维度存在高优先级的AICORE降频问题，分别为pp stage0的8号卡和pp stage3的60号卡。查看对8号卡的降频分析（图10-46）可以发现节点降频主要影响了FlashAttention和MatMul两类算子，导致这两类算子的计算性能劣化，从而影响了整体的训练性能。按照html中给出的建议，需要检查8号卡和60号卡对应节点的温度和最大功率。

图 10-44 性能分析报告展示

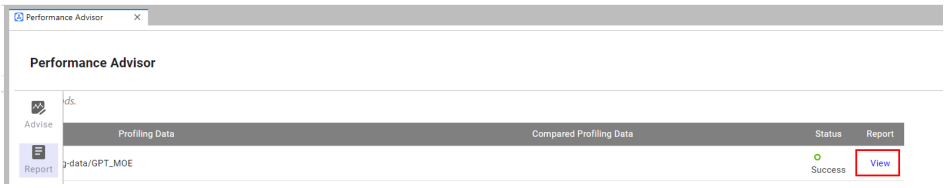


图 10-45 计算维度节点降频问题

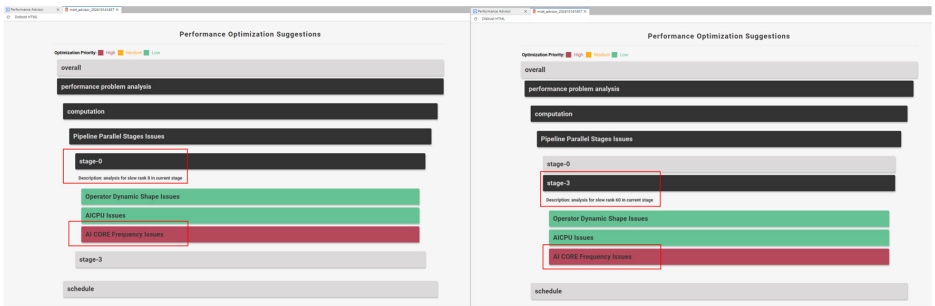


图 10-46 节点降频及其影响算子

AI CORE Frequency Issues

Analysis of rank 8. Issue: For rank 8, 11 operators are found during frequency reduction, and the reduction ratio is larger than 0.05. Only show 10 operators here, see latest mstt_advisor.xlsx for details.
Suggestion: Please check the temperature or max power of your machine.

Operator name	Count	Total duration(us)	AI CORE frequency decreased ratio	Average frequency	Max frequency	Min frequency
aclnnFlashAttentionScoreGrad_FlashAttentionScoreGrad_FlashAttentionScoreGrad	84	32740305.1	46.49%	963.1	1350.0	800.0
aclnnFlashAttentionScore_FlashAttentionScore_FlashAttentionScore	168	28073826.86	36.72%	1138.99	1800.0	800.0
aclnnAddmm_MatMulCommon_MatMulV2	2034	2345260.3	7.48%	1665.34	1800.0	800.0
aclnnCast_CastAiCore_Cast	8907	435354.8	5.10%	1708.12	1800.0	800.0
aclnnGeluBackward_GeluGradAiCore_GeluGrad	171	335080.52	6.27%	1687.13	1800.0	800.0
aclnnLayerNormWithImplMode_LayerNormV3WithImplMode_LayerNormV3	675	271187.08	6.85%	1676.74	1800.0	1050.0
aclnnGelu_Gelu_Gelu	339	209881.98	6.03%	1691.45	1800.0	800.0
aclnnDropoutDoMask_DropoutDoMaskAiCore_DropOutDoMask	1008	106569.32	9.19%	1634.62	1800.0	1000.0
aclnnAddmm_CastAiCore_Cast	2034	3511.6	7.35%	1667.63	1800.0	850.0
aclnnAdds_AddAiCore_Add	777	2364.3	5.44%	1702.06	1800.0	1150.0

----结束

10.5.2 创建诊断任务

本文介绍如何创建Notebook并执行性能诊断任务。

操作步骤

步骤1 创建Notebook实例。

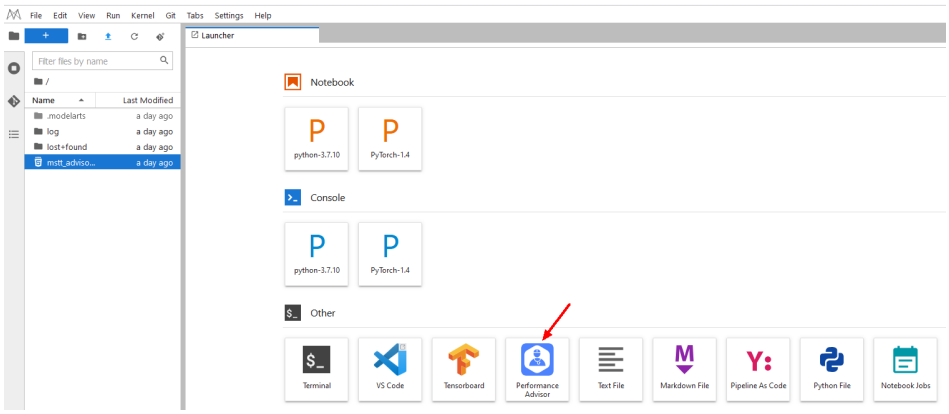
在ModelArts控制台创建一个Notebook实例，选择要使用的AI框架，创建Notebook时可以选择任意镜像。具体参见**创建Notebook实例**。

步骤2 创建成功后，Notebook实例的状态为“运行中”，单击操作列的“打开”，访问JupyterLab。

图 10-47 打开 Notebook 实例

步骤3 进入JupyterLab页面后，自动打开Launcher页面，如下图所示。

图 10-48 JupyterLab 主页

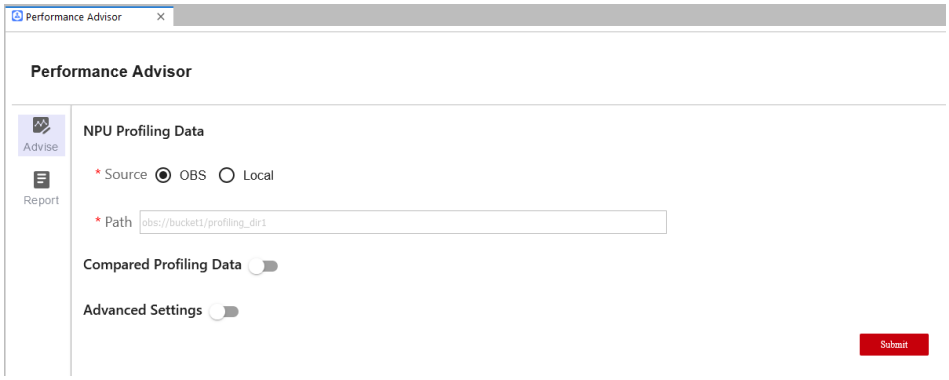


说明

不同AI引擎的Notebook，打开后Launcher页面呈现的Notebook和Console内核及版本均不同，图10-48仅作为示例，请以实际控制台为准。

步骤4 单击Launcher页面的“Performance Advisor”图标，界面将如下图所示

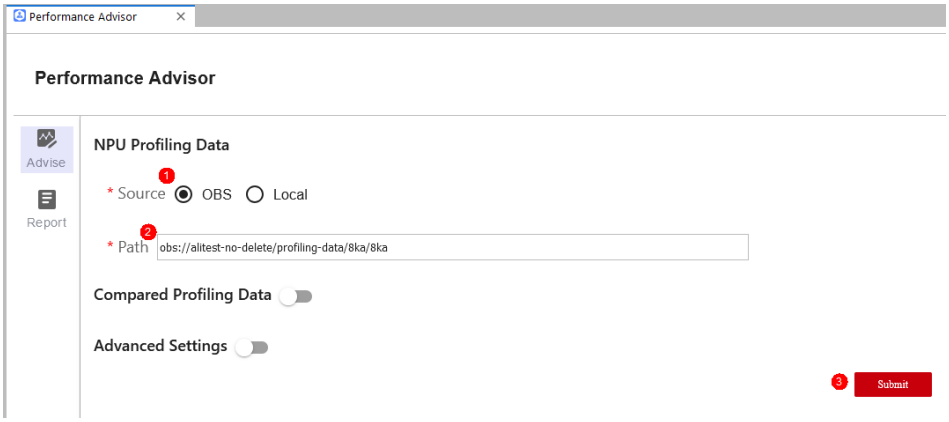
图 10-49 Performance Adviso 主页面



步骤5 提交性能诊断任务

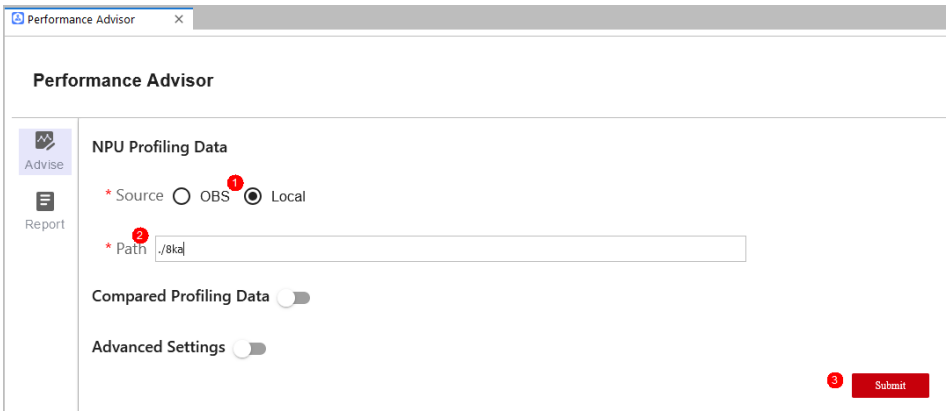
1. 如果您的NPU性能数据存放在OBS上，Source选择OBS，Path输入OBS地址，格式如obs://bucket1/profiling_dir1，单击Submit按钮。界面参考下图。

图 10-50 分析 OBS 上的性能数据



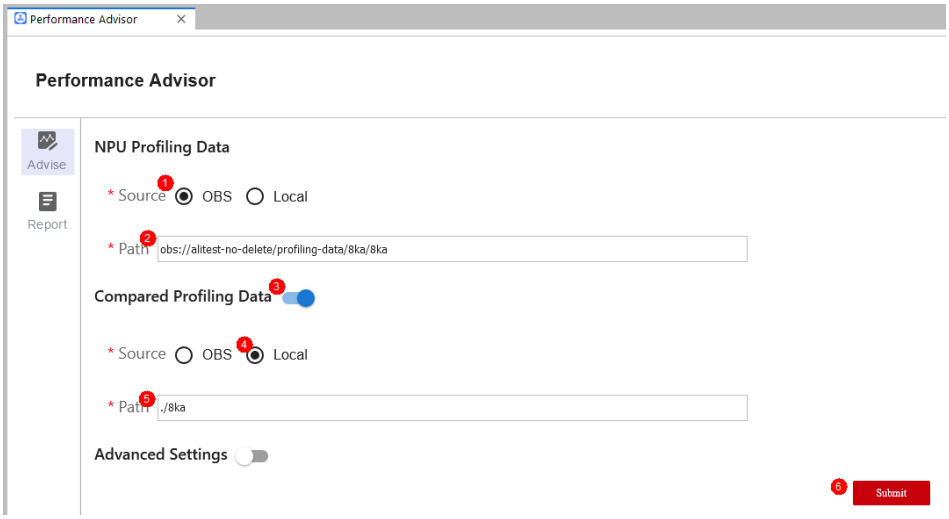
2. 如果您的NPU性能数据存放在Notebook上，Source选择Local，Path输入Notebook的绝对或相对路径（相对于/home/ma-user/work），格式如/home/ma-user/work/profiling_dir1或者./profiling_dir1，单击Submit按钮。界面参考下图。

图 10-51 分析 Notebook 本地的性能数据



3. 如果您有两份性能数据想进行对比，可以点开Compared Profiling Data选项开关，然后分别在NPU Profiling Data和Compared Profiling Data项中输入性能数据所在的Notebook本地或OBS路径，单击Submit按钮。界面参考下图。

图 10-52 对比两份性能数据



4. 性能诊断插件支持设置高级参数，当前支持的高级参数列表如下表所示。

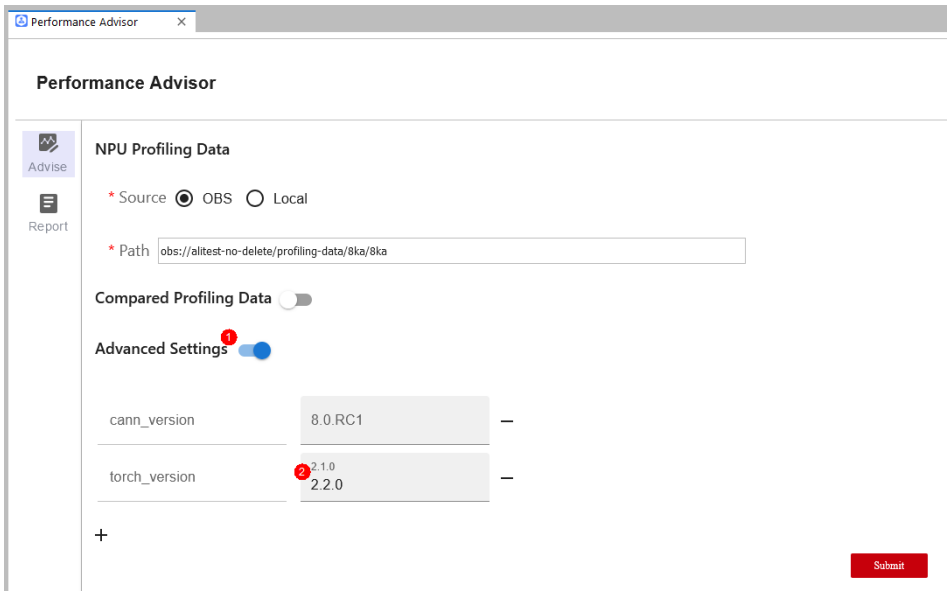
表 10-11 高级参数介绍

序号	键	默认值	是否必填	说明
1	cann_version	8.0.RC1	否	可选值包括6.3.RC2、7.0.RC1、7.0.0和8.0.RC1。当运行环境实际cann版本与可选值不匹配时选择大版本相近的可选值即可。主要影响亲和api分析和aicpu算子分析。
2	torch_version	2.1.0	否	可选值包括1.11.0和2.1.0，当运行环境实际torch版本与可选值不匹配时选择大版本相近的可选值即可。主要有影响亲和api分析。
3	analysis_dimensions	computation, communication, schedule, memory	否	默认进行计算、通信、下发和内存的全维度分析。可以指定默认值中任意单维度进行分析，如仅指定computation进行计算维度分析，仅指定schedule进行下发维度分析。推荐不填写该参数，即使用默认值进行分析。
4	advisor_analyze_processes	1	否	advisor分析进程数，可选范围为1-8的任意整数。当LLM类模型训练的流水并行参数pp大于1时，advisor会对不同pp stage的训练profilingg数据进行分析。通过设置更大的进程数可以使能并行分析从而加快分析速度，但也会增大分析占用的cpu资源。通常单进程需要占用1U的cpu和一定cpu memory（取决于模型大小），请根据实际分析环境的资源规格调整该参数，避免因cpu资源占用过大或者OOM类问题导致的notebook实例异常。

序号	键	默认值	是否必填	说明
5	disable_profiling_comparison	False	否	关闭快慢卡算子比对。对于集群任务的profiling分析，如果存在快慢卡问题则会自动进行集群内部快慢卡的算子性能比对，包括npu侧计算的算子比对和cpu侧torch算子下发比对。当模型较大时，算子比对将会比较耗时，建议设置为True来提升分析速度。
6	disable_affinity_api	False	否	关闭亲和算子（融合算子、亲和优化器）API分析。对于首次从gpu迁移至npu的训练任务性能分析，建议保留该参数，替换亲和算子API通常能获得一定性能收益。对于完成迁移后在npu上长训的训练任务，如果出现性能问题，建议设置为True来提升分析速度。
7	output_path	/home/ma-user/work	否	advisor分析结果输出路径，包含html和xlsx两个文件。

如果您想修改参数配置，可以点开Advanced Settings选项开关，然后对参数进行新增或修改。界面参考下图。

图 10-53 修改高级参数



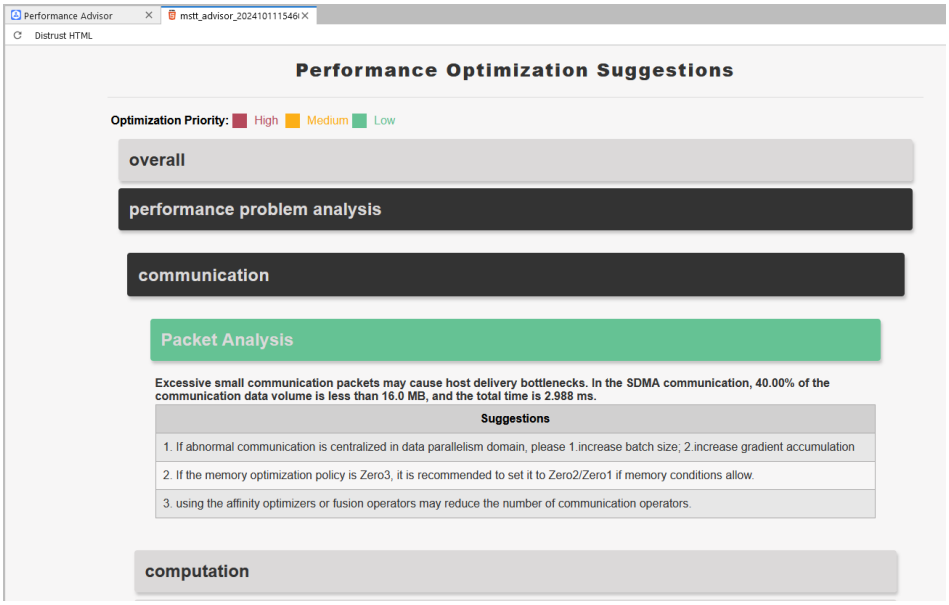
步骤6 查看性能诊断任务结果。

1. 单击Performance Advisor页面的Report选项，可以看到已提交的性能诊断任务详情。

图 10-54 查看性能诊断任务结果

2. 当前支持的状态有“分析中（Analyzing）、成功（Success）和失败（Failed）”。分析中的任务根据性能诊断数据量大小预计将在1~10分钟内完成；成功的任务可单击Report列的View链接查看详细的诊断报告，如下图所示，失败的任务可将鼠标放到Failed字段上，将弹出具体的失败原因。诊断报告详细介绍请查看**查看诊断报告**。

图 10-55 查看性能诊断报告



说明

Report页面将每隔5s自动刷新一次。

----结束

10.5.3 查看诊断报告

Advisor分析profiling会输出html和xlsx两份文件。请优先查看html报告进行训练作业性能调优。xlsx中记录了html中全量数据，如集群计算、通信和下发的耗时，可以基于xlsx对计算耗时、下发耗时和带宽等列进行排序，从而快速过滤出计算慢卡、下发慢卡、带宽最小卡。

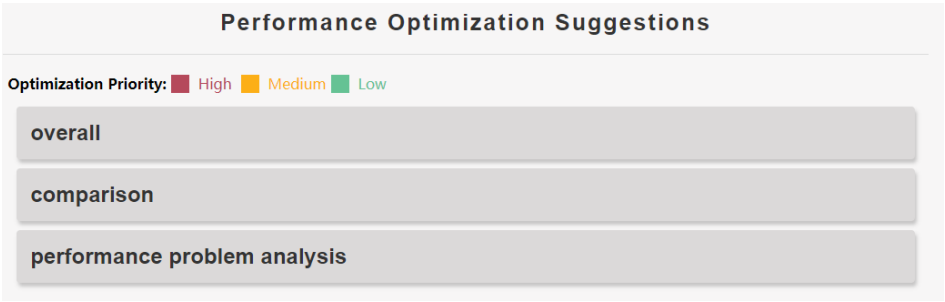
html 总览

html中包括总体性能分析（overall）、快慢卡算子性能比对（comparison）和性能问题分析（performance problem analysis）三大模块。overall模块包含对单卡或者集群的性能统计数据，comparison模块包含目标集群profiling与标杆集群profiling或目标集群内部快慢卡的算子比对数据，performance problem analysis模块包含计算（computation）、下发（schedule）、通信（communication）、内存（memory）和数据加载（dataloader）五个维度的具体分析。用户首先需要查看overall模块初步明确是否存在计算维度的慢卡和下发维度慢卡，然后再重点关注performance problem analysis中对应维度的各项分析及其优先级。

说明

红色为高优先级，黄色为中等优先级，绿色为低优先级。参考html进行分析调优时，请按照优先级从高到低依次进行并测试调优后性能，快速解决重点问题。

图 10-56 html 报告总览-三大模块



当前advisor的performance problem analysis中包含如下分析项。

表 10-12 性能分析能力概览

分析维度	分析项	释义
overall	overall summary	对于单卡profiling进行性能拆解，获取单步计算、下发和通信耗时。
	slow rank	对于集群profiling进行性能统计，获取每张卡不同step的计算、下发和通信耗时。
	slow link	对于集群profiling进行性能统计，获取每张卡不同step的带宽信息。
	environment variable	识别错误配置且会影响性能的环境变量，如PLOG日志级别，HCCL相关环境变量，依赖24年930版本的pta。
comparison	kernel compare	两张卡NPU侧计算算子对比。
	api compare	两张卡CPU侧torch aten算子下发对比。
performance problem analysis	computation - AI CORE frequency	计算维度，识别降频的节点，节点降频会导致flash attention和matmul类算子计算性能变差。

分析维度	分析项	释义
	computation - AICPU	计算维度，识别AICPU算子，部分AICPU算子计算性能较差。
	computation - operator dynamic shape	计算维度，检测动态shape，动态shape会触发频繁的算子编译。
	computation - operator bound	计算维度，算子计算性能分析，例如算子是否充分使用AICORE核数。
	schedule - synchronize stream	下发维度，异常同步流分析，过多同步流会打断CPU侧任务异步下发。
	schedule - garbage collection(GC)	下发维度，识别异常耗时的垃圾回收，垃圾回收会造成大段空闲。
	schedule - operator dispatch	下发维度，算子下发时编译分析，大量算子编译会导致整体训练性能变差。
	schedule - syncBatchNorm	下发维度，NPU上分布式训练使用syncBN性能较差。
	schedule - affinity api	下发维度，自动识别可替换的亲和API（融合算子API如rms_norm，亲和优化器如NpuFusedAdamw）。
	communication - small packet	通信维度，识别因batch过小或者梯度累积较少导致的未充分利用机内通信带宽。
	communication - bandwidth contention	通信维度，识别计算和通信相互掩盖，可能会抢占通信带宽。
	communication - retransmission	通信维度，识别通信重传问题，单次重传耗时4秒以上。
	memory	内存维度，识别异常内存算子。
	dataloader	数据加载维度，异常耗时的数据读取将会导致明显的训练性能劣化。

overall 模块介绍

- **单卡overall summary**

  下图展示了单卡上一个step的端到端耗时为1353ms，其中计算耗时（昇腾硬件上算子执行耗时）是57ms，未掩盖通信耗时为0ms，空闲耗时（硬件上没有进行计算和通信的其他时间）为1295ms。基于这三项数据可以初步判断当前训练任务的主要耗时瓶颈为空闲耗时。空闲耗时通常是任务下发(schedule)、数据加载(dataloader)和内存(memory)三个维度问题导致的，因此可以重点关注

performance problem analysis中对应三个维度的分析。同理如果计算耗时占比较大，则应该重点关注计算维度的分析。

图 10-57 单卡性能拆解总体描述

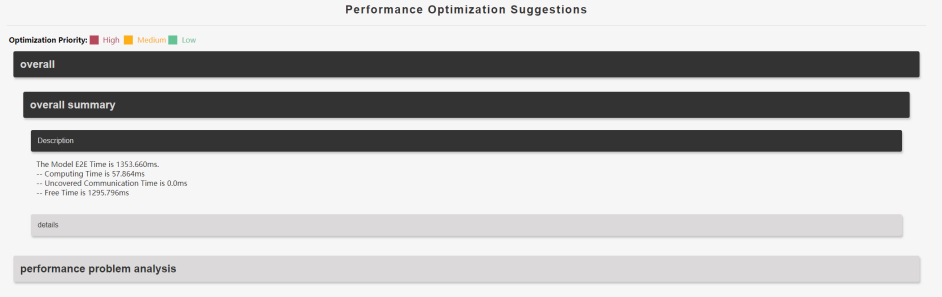
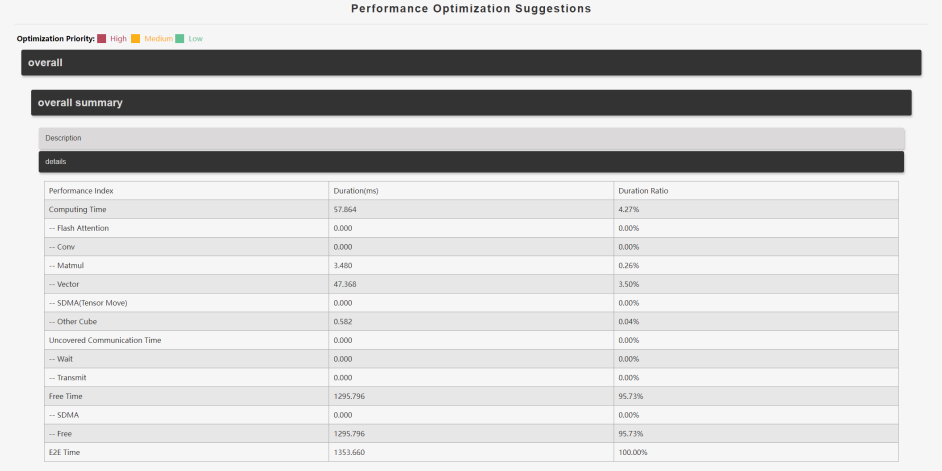


图 10-58 单卡性能拆解详情

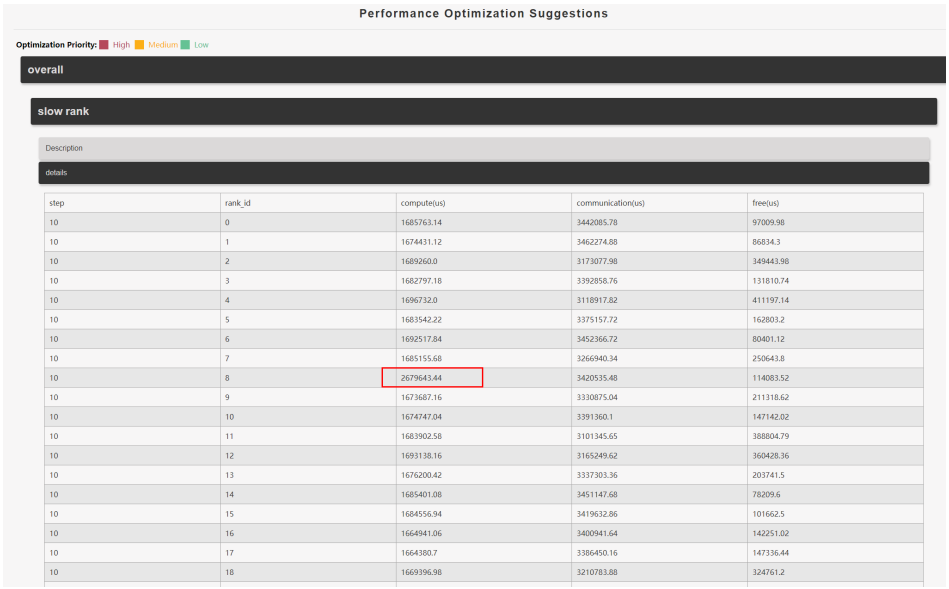


Performance Index	Duration(ms)	Duration Ratio
Computing Time	57.864	4.27%
-- Flash Attention	0.000	0.00%
-- Conv	0.000	0.00%
-- Matmul	3.480	0.26%
-- Vector	47.368	3.50%
-- SDMA(Tensor Move)	0.000	0.00%
-- Other Cube	0.582	0.04%
Uncovered Communication Time	0.000	0.00%
-- Wait	0.000	0.00%
-- Transmit	0.000	0.00%
Free Time	1295.796	95.73%
-- SDMA	0.000	0.00%
-- Free	1295.796	95.73%
E2E Time	1353.660	100.00%

- **多卡slow rank & slow link**

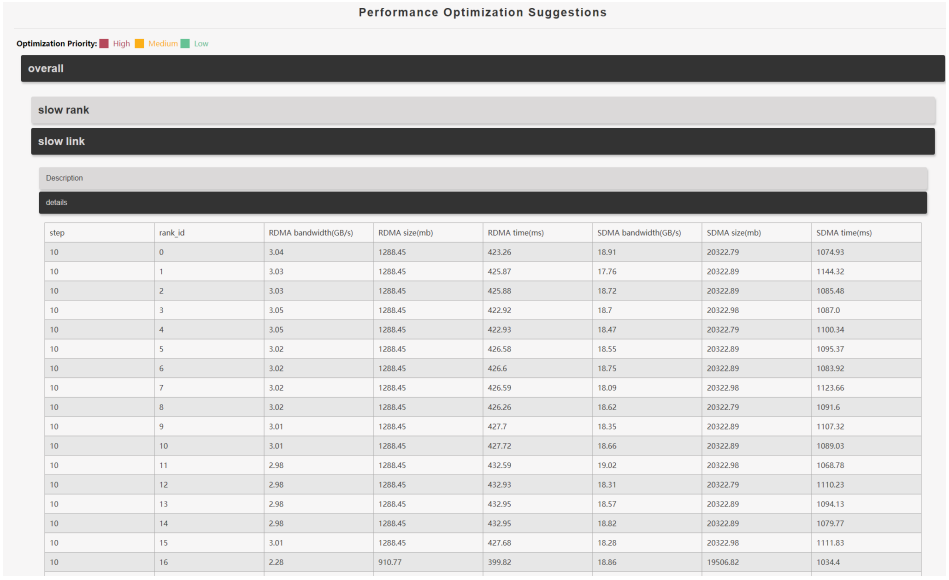
  下图展示了多卡profiling分析的overall模块，包含集群快慢卡统计数值(slow rank，用于分析计算和任务下发的快慢卡)和集群带宽统计数值(slow link，用于分析集群中的网络通信慢链路)。点开slow rank模块，html中会基于表格展示每张卡不同step的计算耗时、通信耗时和空闲耗时。基于该表格，通常关注计算耗时(compute)和空闲耗时(free)这两列，可以初步分析当前瓶颈点是计算还是任务下发，以及是否存在计算快慢卡和下发快慢卡。如下图所示，可以看到8号卡的计算耗时明显大于其他卡，因此8号卡的“短板效应”将会拖慢集群的整体训练速度，后续性能分析需要重点关注8号卡的计算维度。

图 10-59 多卡不同 step 计算、下发和通信耗时统计值



step	rank_id	compute(us)	communication(us)	free(us)
10	0	1685763.14	3442085.78	97009.98
10	1	1674431.12	3462274.88	86834.3
10	2	1689260.0	3173077.98	349443.98
10	3	1682797.18	3392858.76	131810.74
10	4	1696732.0	3118917.82	411197.14
10	5	1683542.22	3375157.72	162803.2
10	6	1692517.84	3452366.72	80401.12
10	7	1685155.68	3266940.34	250643.8
10	8	2679643.44	3420535.48	114083.52
10	9	1673687.16	3330875.04	211318.62
10	10	1674747.04	3391360.1	147142.02
10	11	1683902.58	3101345.65	388804.79
10	12	1693138.16	3165249.62	360428.36
10	13	1676200.42	3337303.36	203741.5
10	14	1685401.08	3451147.68	78209.6
10	15	1684556.94	3419632.86	101662.5
10	16	1664941.06	3400941.64	142251.02
10	17	1664380.7	3386450.16	147336.44
10	18	1669396.98	3210783.88	324761.2

图 10-60 多卡不同 step 通信带宽统计值



step	rank_id	RDMA bandwidth(GB/s)	RDMA size(mb)	RDMA time(ms)	SDMA bandwidth(GB/s)	SDMA size(mb)	SDMA time(ms)
10	0	3.04	1288.45	423.26	18.91	20322.79	1074.93
10	1	3.03	1288.45	425.87	17.76	20322.89	1144.32
10	2	3.03	1288.45	425.88	18.72	20322.89	1085.48
10	3	3.05	1288.45	422.92	18.7	20322.98	1087.0
10	4	3.05	1288.45	422.93	18.47	20322.79	1100.34
10	5	3.02	1288.45	426.58	18.55	20322.89	1095.37
10	6	3.02	1288.45	426.6	18.75	20322.89	1083.92
10	7	3.02	1288.45	426.59	18.09	20322.98	1123.66
10	8	3.02	1288.45	426.26	18.62	20322.79	1091.6
10	9	3.01	1288.45	427.7	18.35	20322.89	1107.32
10	10	3.01	1288.45	427.72	18.66	20322.89	1089.03
10	11	2.98	1288.45	432.59	19.02	20322.98	1068.78
10	12	2.98	1288.45	432.93	18.31	20322.79	1110.23
10	13	2.98	1288.45	432.95	18.57	20322.89	1094.13
10	14	2.98	1288.45	432.95	18.82	20322.89	1079.77
10	15	3.01	1288.45	427.68	18.28	20322.98	1111.83
10	16	2.28	910.77	399.82	18.86	19506.82	1034.4

- **环境变量Environment Variable Issues**
  识别模型训练环境中设置的昇腾相关环境变量并给出建议。

图 10-61 环境变量分析

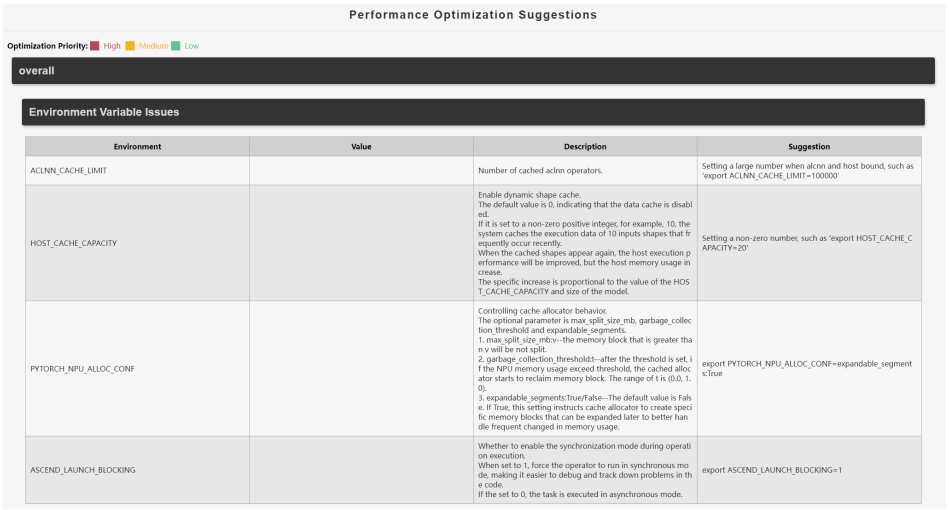


表 10-13 当前支持的环境变量

环境变量名称	释义
ASCEND_GLOBAL_LOG_LEVEL	plog日志级别，推荐设置为2（warning级别），低级别日志等级会导致cpu侧性能问题。
HCCL_RDMA_TC	HCCL通信相关环境变量，通常无需设置该环境变量，建议unset该环境变量。具体参考**拥塞控制与纠错配置策略**
HCCL_RDMA_SL	HCCL通信相关环境变量，通常无需设置该环境变量，建议unset该环境变量。具体参考**拥塞控制与纠错配置策略**
ACLNN_CACHE_LIMIT	用于缓存cann侧的aclnn算子，当空闲时间（free）较大时，可以尝试设置一个较大的数值，如export ACLNN_CACHE_LIMIT=100000
HOST_CACHE_CAPACITY	用于动态shape缓存，当存在动态shape时，设置一个非零正整数，如export HOST_CACHE_CAPACITY=20
ASCEND_ENHANCE_ENABLE	使能HCCL的FFTS+模式，export ASCEND_ENHANCE_ENABLE=1
PYTORCH_NPU_ALLOC_CONF	控制缓存分配，当存在内存碎片时，执行export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
ASCEND_LAUNCH_BLOCKING	是否启动同步下发，同步下发会导致严重的性能劣化，建议执行unset ASCEND_LAUNCH_BLOCKING

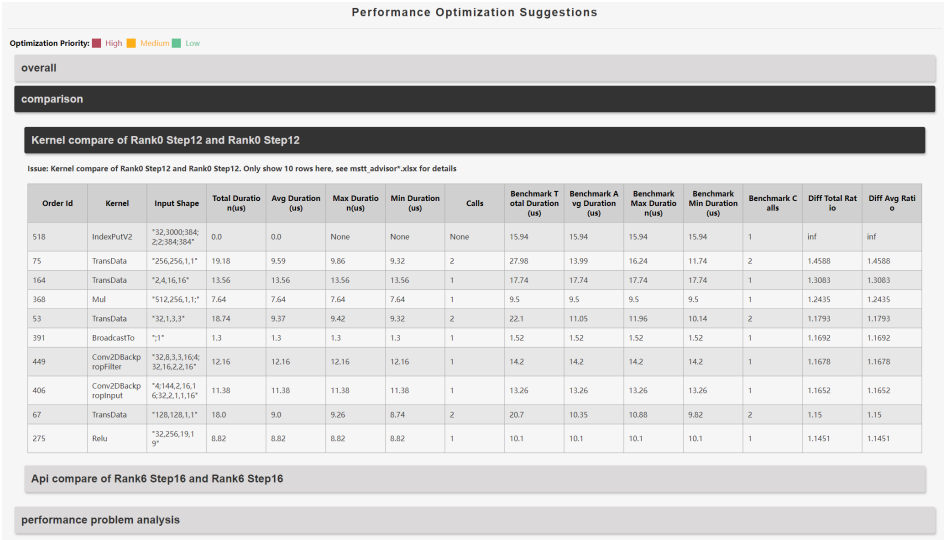
comparison 模块介绍

当同时指定目标集群profiling和标杆集群profiling或者目标集群内部存在快慢卡时，advisor会针对计算和下发性能存在差异的卡（快慢卡）进行算子级的对比。

如下图所示，当分析时显式指定了标杆集群profiling数据，advisor识别到两次训练任务中0号卡的step12存在计算性能差异，则会对目标集群的0号卡step12与标杆集群的0

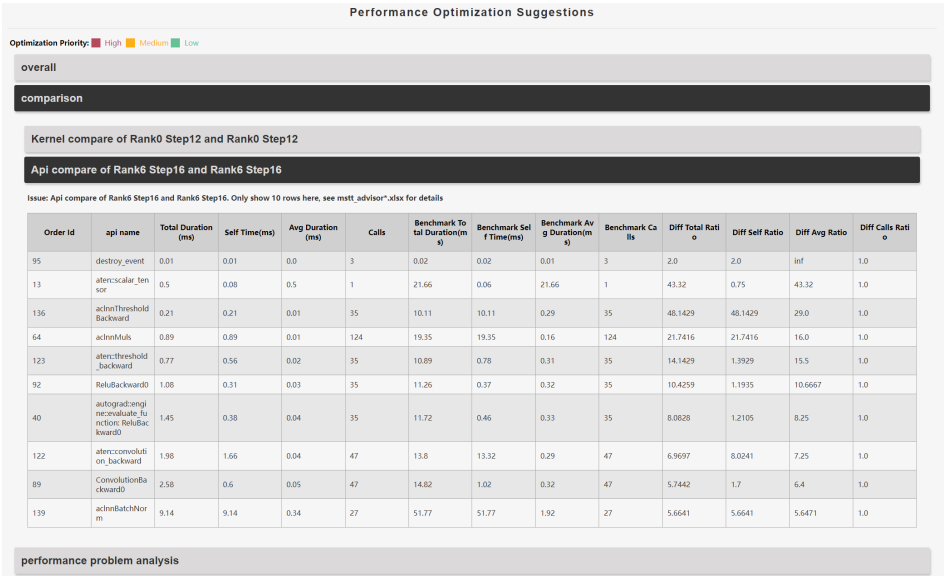
号卡step12进行kernel（npu侧计算的算子）性能对比。基于该对比数据，可以判断两张卡上的npu算子是否存在计算性能差异。

图 10-62 目标集群 profiling 数据与标杆集群 profiling 数据的 kernel 算子对比



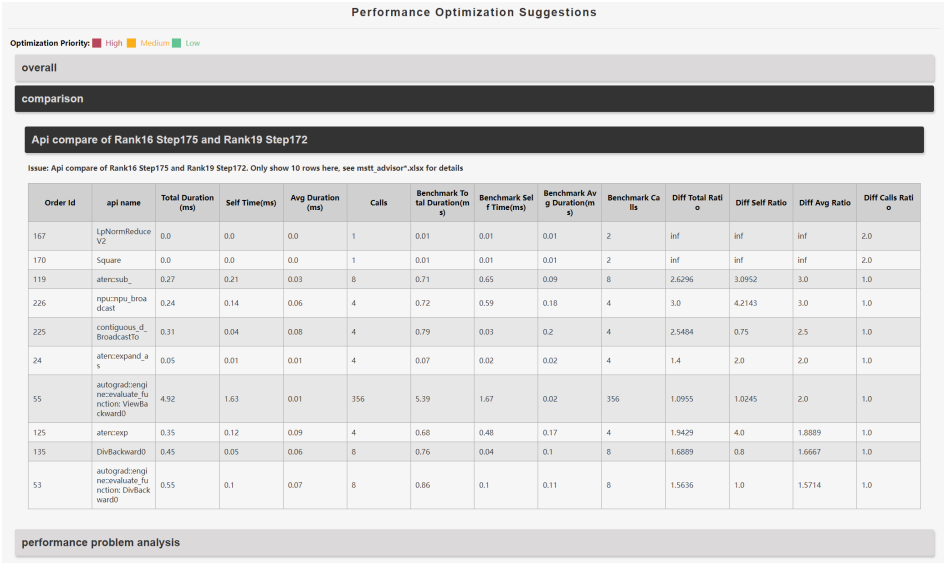
如下图所示，当分析时显式指定了标杆集群profiling数据，advisor识别到两次训练任务中6号卡的step16存在api下发性能差异，对目标集群的6号卡step16与标杆集群的6号卡step16进行了api（cpu侧的torch aten算子任务下发）的性能对比。基于该对比数据，可以判断两张卡上的aten算子是否存在下发性能差异。

图 10-63 目标集群 profiling 数据与标杆集群 profiling 数据的 api 下发对比



如下图所示，分析时并没有指定标杆集群profiling数据，但advisor识别到目标集群存在任务下发快慢卡（16和19号卡）现象，因此对比了16号卡step175和19号卡step172的api下发性能。

图 10-64 目标集群（无标杆集群 profiling）内部快慢卡 api 下发对比

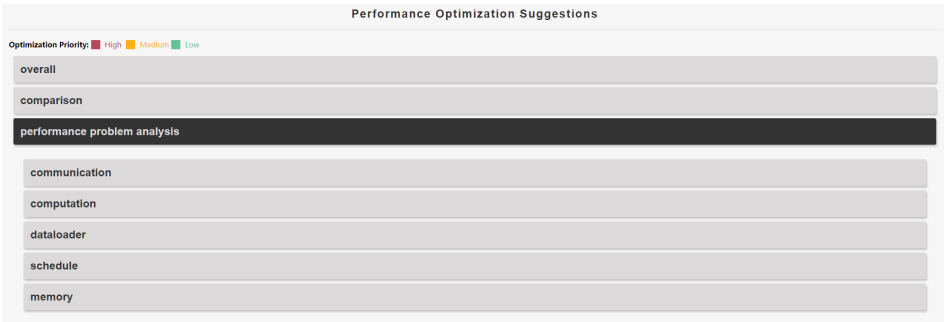


Order Id	api name	Total Duration (ms)	Self Time(ms)	Avg Duration (ms)	Calls	Benchmark Total Duration(ms)	Benchmark Self Time(ms)	Benchmark Avg Duration(ms)	Benchmark Calls	Diff Total Ratio	Diff Self Ratio	Diff Avg Ratio	Diff Calls Ratio
167	LpNormReduceV2	0.0	0.0	0.0	1	0.01	0.01	0.01	2	inf	inf	inf	2.0
170	Square	0.0	0.0	0.0	1	0.01	0.01	0.01	2	inf	inf	inf	2.0
119	aten::sub_	0.27	0.21	0.03	8	0.71	0.65	0.09	8	2.6296	3.0952	3.0	1.0
226	npu::npu_broadcast	0.24	0.14	0.06	4	0.72	0.59	0.18	4	3.0	4.2143	3.0	1.0
225	contiguous_d_BroadcastTo	0.31	0.04	0.08	4	0.79	0.03	0.2	4	2.5484	0.75	2.5	1.0
24	aten::expand_as	0.05	0.01	0.01	4	0.07	0.02	0.02	4	1.4	2.0	2.0	1.0
55	autograd::engine::evaluate_function: ViewBackward0	4.92	1.63	0.01	356	5.39	1.67	0.02	356	1.0955	1.0245	2.0	1.0
125	aten::exp	0.35	0.12	0.09	4	0.68	0.48	0.17	4	1.9429	4.0	1.8889	1.0
135	DivBackward0	0.45	0.05	0.06	8	0.76	0.04	0.1	8	1.6889	0.8	1.6667	1.0
53	autograd::engine::evaluate_function: DivBackward0	0.55	0.1	0.07	8	0.86	0.1	0.11	8	1.5636	1.0	1.5714	1.0

performance problem analysis 模块介绍

perfomance problem analysis中会细分为计算（computation），下发（schedule），通信（communication）、内存（memory）和数据加载（dataloader）五个维度，根据训练作业卡数、训练实际性能问题有不同的呈现，并非所有训练任务都有上述五个维度的分析。

图 10-65 html 报告总览-性能分析五维度



- **computation**

  计算维度通常包含如下几类问题：

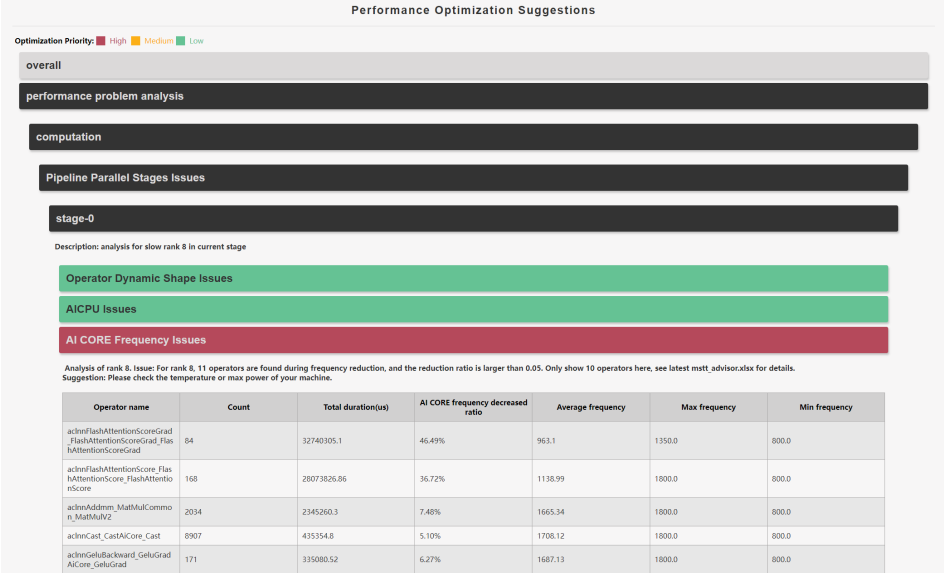
  - 降频：对应html中的'AI CORE Frequency Issues'。NPU AICORE主频降低，导致flash attention和matmul类算子计算性能严重劣化。
  - AICPU算子：对应html中的'AICPU Issues'。部分算子因NPU支持度或者输入数据shape/dtype等原因，无法在AICORE上运行，因此会放在AICPU上进行计算，部分AICPU算子会存在明显的性能劣化。
  - 动态shape：对应html中的'Operator Dynamic Shape Issues'。动态shape场景，部分昇腾算子会重复编译，导致计算延后，影响性能。
  - operator bound等问题：算子未打满AICORE的核数，导致计算利用率较低。

  具体介绍如下：

  - **AI CORE Frequency Issues**

下图展示了高优先级的降频问题，从表格中可以看到flash attention算子耗时最长且降频比率最高，因此降频严重影响了整体的训练性能。对于降频问题，用户通常无法自行解决，需要联系服务方如华为云技术支持排查机器的温度和功耗。

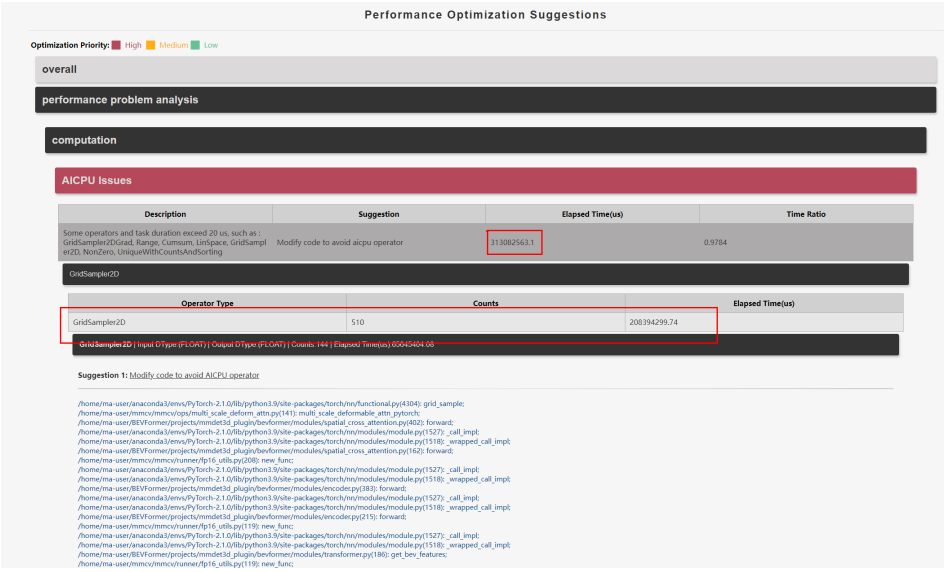
图 10-66 降频分析



- **AICPU Issues**

  下图展示了高优先级的AICPU问题，AICPU算子单步计算耗时313秒，GridSample2D算子单步计算耗时208秒，因此需要重点关注该算子。可以通过html中提供的堆栈信息查看源码中对该算子的调用是否可以替换成其他torch api，如果分析后无法替换可以求助昇腾算子侧的算子开发人员进行算子优化分析。

图 10-67 AICPU 算子分析

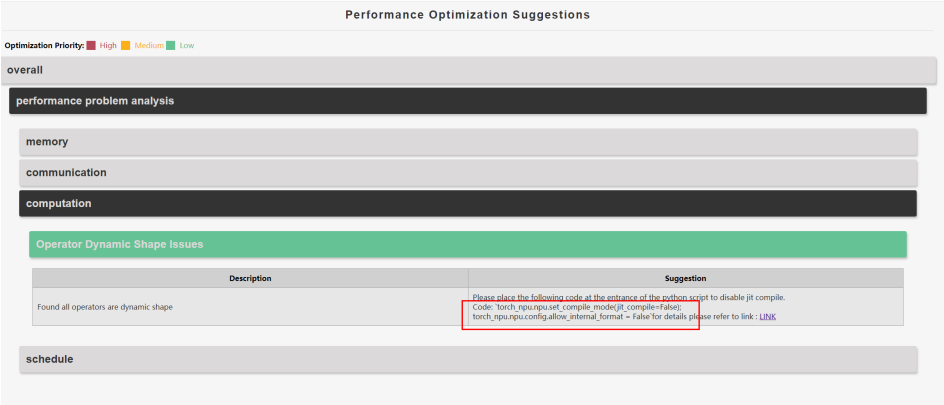


- **Operator Dynamic Shape Issues**

  下图展示了低优先级的动态shape问题，在NPU上动态shape可能导致频繁的算子编译从而影响训练性能，可以按照html中的提示在训练脚本开头加上如

下红框中的两行代码（分布式训练请确保分布式训练的每个进程都可以使能这两行代码）。

图 10-68 动态 shape 分析



- **schedule**

  下发维度通常包含如下几类问题

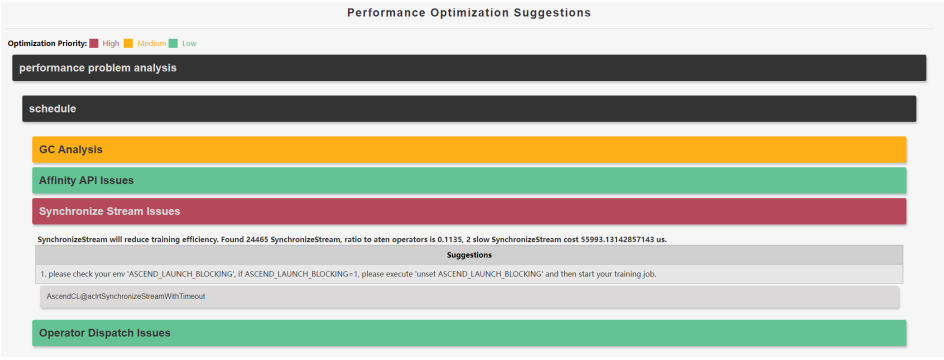
  - 同步流：对应html中的'Synchronize Stream Issues'。用户设置了ASCEND_LAUNCH_BLOCKING环境变量，打断了CPU侧算子的异步下发，严重影响训练性能。
  - GC：对应html中的'GC Analysis'。python garbage collection机制，大规模集群训练时GC任务会导致部分step训练耗时异常增大。
  - 算子下发：对应html中的'Operator Dispatch Issues'。训练时如果频繁进行算子编译会严重影响训练性能，可以增加两行python代码关闭算子编译。
  - 亲和API：对应html中的'Affinity API Issues'。通过使能亲和API（NPU融合算子API如rms_norm，NPU亲和优化器如NPUFusedAdamw）可以减少算子下发数量，从而提升训练性能。
  - syncBatchNorm：对应html中的'SyncBatchNorm Issues'。多卡DDP训练时如果使用syncBatchNorm，会存在明显的算子下发和通信瓶颈。

  具体介绍如下：

  - **Synchronize Stream Issues**

    下图展示了高优先级的同步流问题，html中提示发现大量同步算子，可以尝试`unset ASCEND_LAUNCH_BLOCKING` 环境变量后再进行训练。

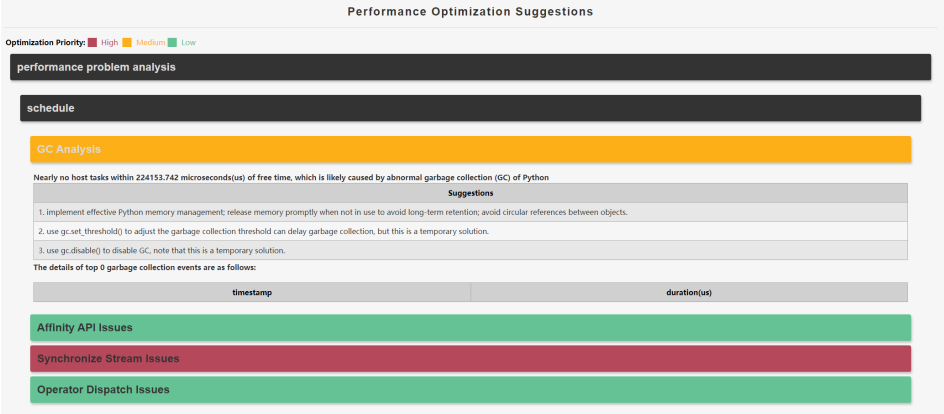
    图 10-69 异常同步流分析

  - **GC Analysis**

下图展示了中优先级的GC问题，html中提示发现单步训练中存在200ms左右的空闲时间且在该时间窗内cpu侧没有进行训练算子下发，怀疑是GC导致，可以尝试加上`gc.disable()`关闭GC。

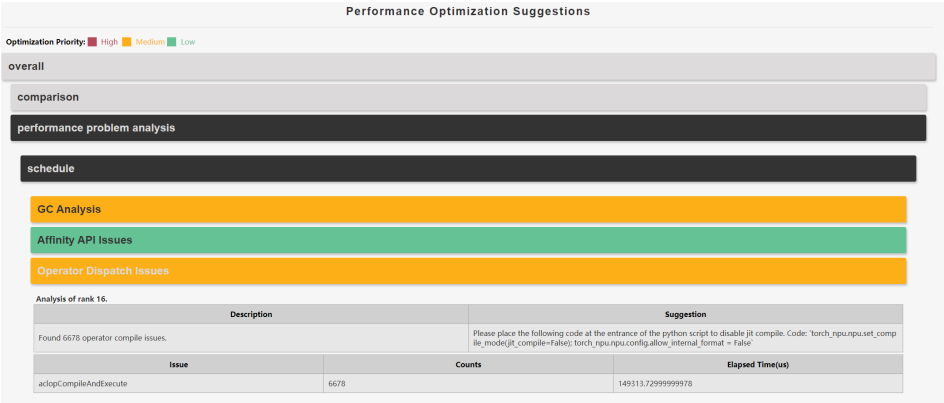
图 10-70 python 垃圾回收（GC）分析



- **Operator Dispatch Issues**

  下图展示了中优先级的算子下发问题，html中提示识别到单步训练中存在6678个算子进行了算子编译，编译耗时149ms，可以在训练脚本最开头加上suggestion中的两行代码关闭算子编译（分布式训练请确保每个进程都可以使能这两行代码）。

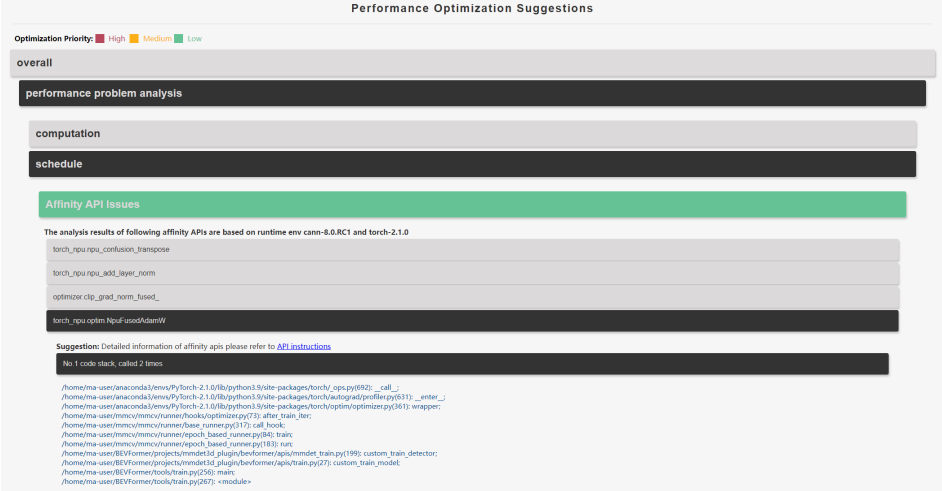
图 10-71 算子编译分析



- **Affinity API Issues**

  下图展示了低优先的亲和API替换，通常仅在首次将训练任务从GPU迁移至NPU时需要关注这部分内容。已经在NPU上进行长训的任务出现性能问题，可以忽略该部分。html中提示存在torch_npu.confusion_transpose, 梯度裁剪和亲和优化器等多个可替换的API，用户可根据代码堆栈找到需要替换的具体源码，然后根据API instruction跳转后的参考文档修改源代码，从而使能亲和API提升训练性能。注意这里提示的亲和API并非都能提升训练性能，需要用户替换后实测，由于有一定代码修改和测试成本，因此优先级可以视作最低。

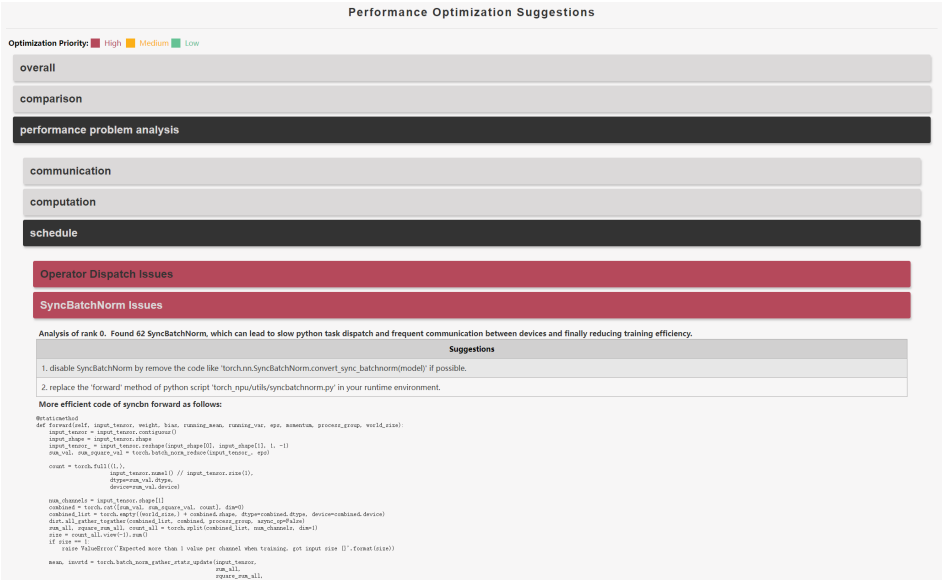
图 10-72 亲和 API 分析



- **SyncBatchNorm Issues**

  下图展示了高优先级的syncBatchNorm问题，html中提示可以通过去除convert_sync_batchnorm代码使能普通的batchNorm，如果模型开发者评估认为对训练精度存在影响，可以使用html中给出的代码段替换torch_npu中syncbatchnorm.py文件的forward方法（可以在训练环境中执行`pip show torch_npu`查看torch_npu的安装路径）。这类优化通常可以较显著地提升训练速度。

图 10-73 SyncBatchNorm 分析



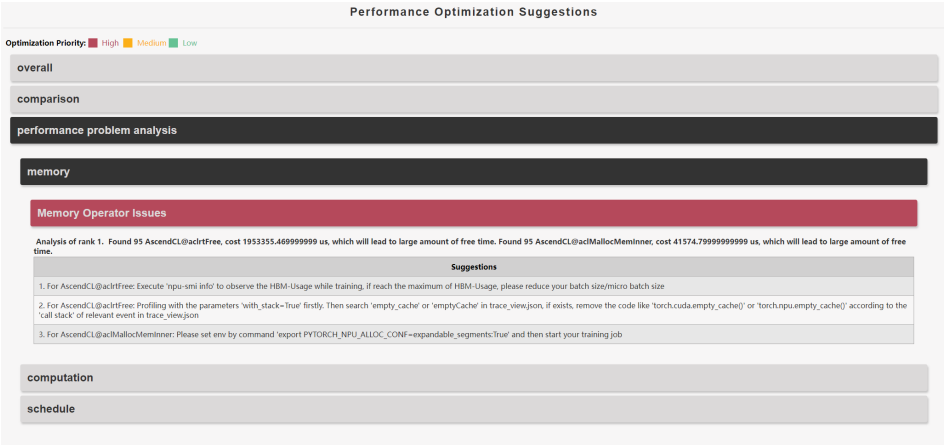
- **memory**

  内存维度当前识别的问题较为简单，通常是NPU HBM占用过大或者存在内存碎片导致自动触发昇腾内存释放/重整算子（Memory Operator Issues），进而影响了训练性能。

  下图展示了高优先级的内存算子问题，html中提示对于1号卡存在大量aclrtFree和aclMalloc算子，导致存在较大的空闲时间。按照建议，对于aclrtFree算子类问题，首先观察NPU HBM是否已经打满，如果打满则尝试减小micro batch size或者调整训练策略，例如训练LLM模型时使用ZeRO 3优化内存。其次采集profiling

时设置“with_stack=True参数”，然后基于MindInsight Studio可视化profiling数据查看代码中是否存在empty_cache操作，如果存在则注释对应代码。对于aclMalloc算子类问题，按照建议设置环境变量即可减少内存碎片。

图 10-74 内存算子分析



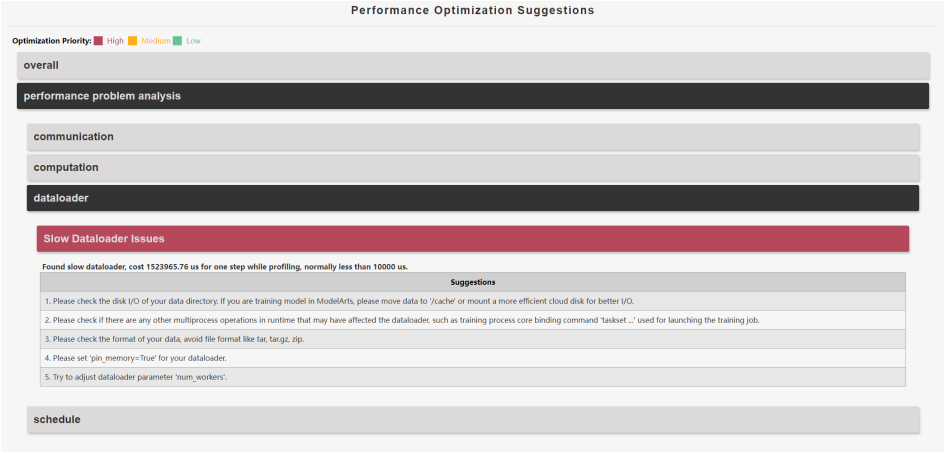
- **dataloader**

  数据加载维度（Slow Dataloader Issues）通常包含如下几类问题：

  - 数据放在读写性能较差的存储盘上，如云上的EVS和EFS。
  - 多卡训练时使用单进程dataloader，即num_workers参数默认为0。
  - 存在其他多进程操作影响了数据多进程读取。
  - 数据格式问题，例如zip、tar.gz等压缩包。
  - dataloader参数设置不合理，如没有配置锁页内存pin_memory=True。

    下图中展示了高优先级的dataloader问题，较慢的数据加载将会严重影响训练速度。html中提示分别从数据盘、多进程、dataloader参数设置、数据格式等不同方向进行排查。

    图 10-75 数据加载分析

- **communication**

  通信维度当前支持检测如下几类问题。

  - 通信计算并行时抢占通信带宽：对应html中的'BandWidth Contention Analysis'。在LLM类模型训练过程中，对于megatron类框架可以配置overlap

相关参数使能计算和通信互相掩盖，进而提升训练性能。但部分场景中计算抢占通信带宽反而会导致性能劣化。

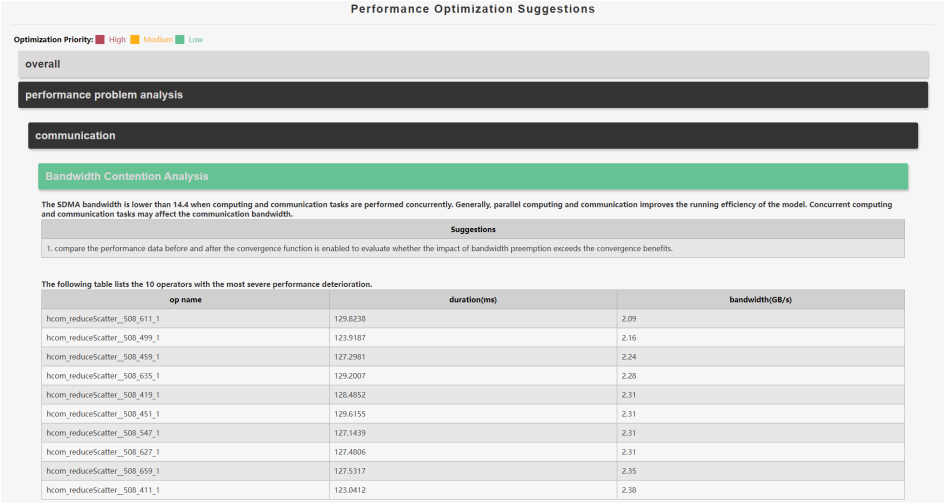
- 小包分析：对应html中的'Packet Analysis'。当BatchSize较小或其他场景，并没有打慢NPU HBM，卡间通信数据包较小，没有充分利用通信带宽。
- RDMA重传（跨节点通信）：对应html中的“Communication Retransmission Analysis”。当网络通信配置出现冲突情况下，RDMA通信传输可能出现重传，导致通信耗时异常大幅增加。

具体介绍如下：

- **BandWidth Contention Analysis**

  下图展示了低优先级的通信带宽抢占问题，大部分场景中通信计算互相掩盖是有较大训练性能收益的，可以通过修改overlap或其他相关参数来测试是否存在性能提升。

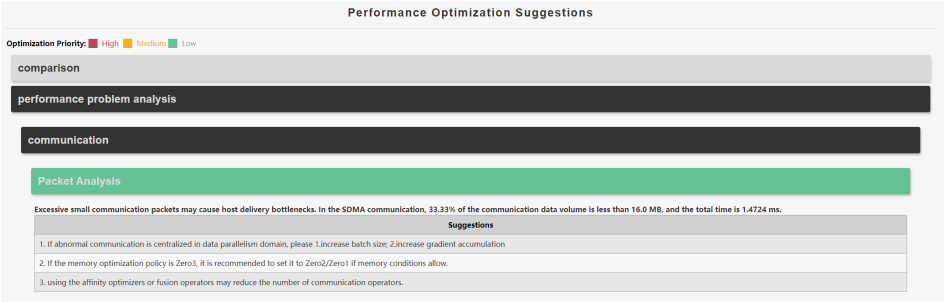
  图 10-76 通信带宽抢占分析



- **Packet Analysis**

  下图展示了低优先级的通信小包问题，html中提示SDMA（机内通信）带宽相对较小，可以尝试增大batchSize或者梯度累积参数，如果配置了ZeRO3则推荐使用ZeRO1或者ZeRO2（如果内存够）。

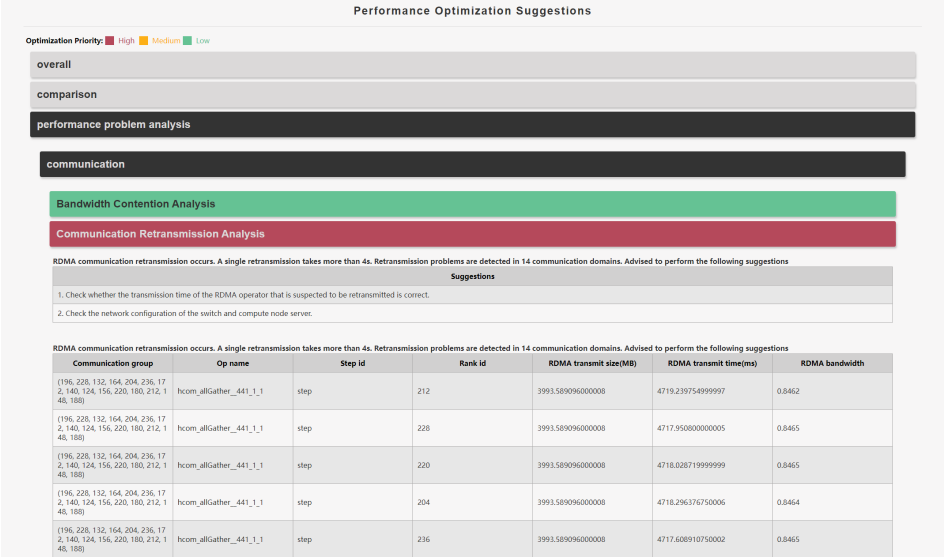
  图 10-77 通信小包分析



- **Communication Retransmission Analysis**

  单次通信重传将会耗时4秒以上，会导致较严重的通信性能劣化，这类问题通常是由于节点网络配置错误导致，可以联系服务方如华为云技术支持排查网络配置。

图 10-78 通信重传分析



10.6 Dit 模型 PyTorch 迁移与精度性能调优

10.6.1 场景介绍及环境准备

场景介绍

DiT（Diffusion Transformers）模型是一种将Transformer架构引入扩散模型的新方法。传统的扩散模型通常使用U-Net架构，而DiT模型则用Transformer替代了U-Net，处理图像生成和去噪等任务。核心思想是通过Transformer的自注意力机制来捕捉序列中的依赖关系，从而提高生成图像的质量。研究表明，具有较高GFLOPs的DiT模型在图像生成任务中表现更好，尤其是在ImageNet 512×512和256×256的测试中，DiT-XL/2模型实现了2.27的FID值。

下文以Dit模型为例，介绍如何在昇腾设备上如何进行模型迁移，精度及性能调优。

环境准备

迁移环境准备有以下两种方式：

表 10-14 迁移环境准备方式

方式	说明
ModelArts Notebook	该环境为在线调试环境，主要面向演示、体验和快速原型调试场景。 环境开通指导请参考Notebook环境创建。
ModelArts Lite DevServer	该环境为裸机开发环境，主要面向深度定制化开发场景。 环境开通指导请参考DevServer资源开通；环境配置指导请参考Snt9B裸金属服务器环境配置指南。

本文基于ModelArts Lite DevServer进行操作，请参考上表说明在贵阳一环境开通和配置指导完成裸机和容器开发初始化配置。

镜像地址为swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a。

请注意业务基础镜像选择**Ascend+PyTorch**镜像。

10.6.2 训练迁移适配

完成环境准备之后，本节将详细介绍Dit模型训练迁移过程。

步骤1 执行以下命令，下载代码。

```
git clone https://github.com/facebookresearch/DiT.git
cd Dit
```

步骤2 执行以下命令，安装依赖项。

```
pip install diffusers==0.28.0 accelerate==0.30.1 timm==0.9.16
```

步骤3 准备数据集。

下载**Kaggle**官网提供的imagenet-mini数据集，解压之后文件大小4.1GB。该数据集是从**[imagenet-2012]**数据集中筛选的少量数据集。

步骤4 准备预训练权重。

下载**Hugging Face**权重。

步骤5 迁移适配。

1. 入口函数train.py导入自动迁移接口。

   执行以下命令，导入自动迁移接口。

   ```
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   ```python
   # Copyright (c) Meta Platforms, Inc. and affiliates.
   # All rights reserved.

   # This source code is licensed under the license found in the
   # LICENSE file in the root directory of this source tree.

   """
   A minimal training script for DiT using PyTorch DDP.
   """
   import torch
   # the first flag below was False when we tested this script but True makes A100 training a lot faster:
   torch.backends.cuda.matmul.allow_tf32 = True
   torch.backends.cudnn.allow_tf32 = True
   import torch.distributed as dist
   from torch.nn.parallel import DistributedDataParallel as DDP
   from torch.utils.data import DataLoader
   from torch.utils.data.distributed import DistributedSampler
   from torchvision.datasets import ImageFolder
   from torchvision import transforms
   import numpy as np
   from collections import OrderedDict
   from PIL import Image
   from copy import deepcopy
   from glob import glob
   from time import time
   import argparse
   import logging
   import os

   from models import DiT_models
   from diffusion import create_diffusion
   from diffusers.models import AutoencoderKL

   try:
       import torch_npu
       from torch_npu.contrib import transfer_to_npu
   except ImportError:
       print('package torch_npu not found.')

   ```

2. 将预训练模型指定为实际下载路径。
3. 开始训练。

   单卡训练启动方式：

```
torchrun --nnodes=1 --nproc_per_node=1 train.py --model DiT-XL/2 --data-path imagenet/train --global-batch-size 16
```

多卡训练启动方式：

```
torchrun --nnodes=1 --nproc_per_node=8 train.py --model DiT-XL/2 --data-path imagenet/train --global-batch-size 128
```

正常训练回显日志：

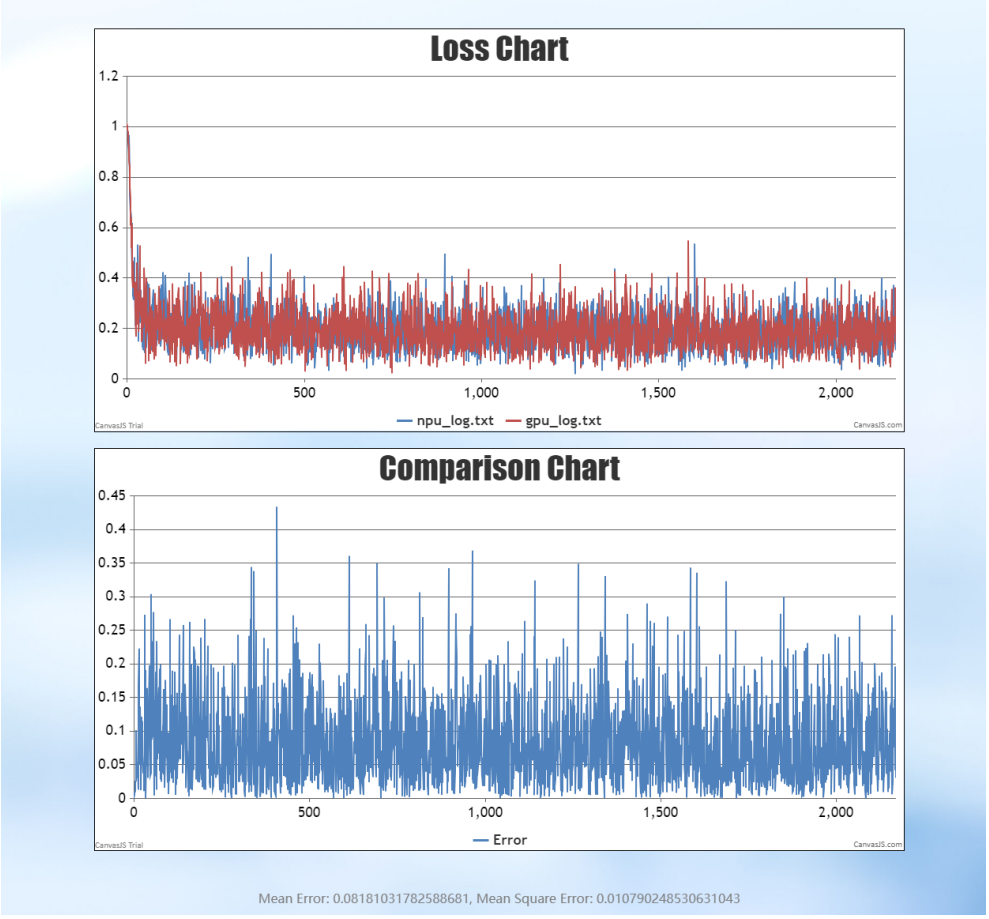
```
(PyTorch-2.1.0) [root@9986a4a94cb9 DiT]# ASCEND_RT_VISIBLE_DEVICES=7 torchrun --nnodes=1 --nproc_per_node=1 train.py --model DiT-XL/2 --data-path /efs_guiyang/r30049464/Dit/imagenet-mini/train/ --global-batch-size 1 --log-every 1 --epochs 1
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch_npu/contrib/transfer_to_npu.py:299: ImportWarning:
    ********************************************************************************************************************
    The torch.Tensor.cuda and torch.nn.Module.cuda are replaced with torch.Tensor.npu and torch.nn.Module.npu now..
    The torch.cuda.DoubleTensor is replaced with torch.npu.FloatTensor cause the double type is not supported now..
    The backend in torch.distributed.init_process_group set to hccl now..
    The torch.cuda.* and torch.cuda.amp.* are replaced with torch.npu.* and torch.npu.amp.* now..
    The device parameters have been replaced with npu in the function below:
    torch.logspace, torch.randint, torch.hann_window, torch.rand, torch.full_like, torch.ones_like, torch.rand_like, torch.randperm, torch.arange, torch.frombuffer, torch.normal, torch._empty_per_channel_affine_quantized, torch.empty_strided, torch.empty_like, torch.scalar_tensor, torch.tril_indices, torch.bartlett_window, torch.ones, torch.sparse_coo_tensor, torch.randn, torch.kaiser_window, torch.tensor, torch.triu_indices, torch.as_tensor, torch.zeros, torch.randint_like, torch.full, torch.eye, torch._sparse_csr_tensor_unsafe, torch.empty, torch._sparse_coo_tensor_unsafe, torch.blackman_window, torch.zeros_like, torch.range, torch.sparse_csr_tensor, torch.randn_like, torch.from_file, torch._cudnn_init_dropout_state, torch._empty_affine_quantized, torch.linspace, torch.hamming_window, torch.empty_quantized, torch._pin_memory, torch.autocast, torch.load, torch.Generator, torch.Tensor.new_empty, torch.Tensor.new_empty_strided, torch.Tensor.new_full, torch.Tensor.new_ones, torch.Tensor.new_tensor, torch.Tensor.new_zeros, torch.Tensor.to, torch.nn.Module.to, torch.nn.Module.to_empty
    ********************************************************************************************************************

  warnings.warn(msg, ImportWarning)
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch_npu/contrib/transfer_to_npu.py:260: RuntimeWarning: torch.jit.script and torch.jit.script_method will be disabled by transfer_to_npu, which currently does not support them, if you need to enable them, please do not use transfer_to_npu.
  warnings.warn(msg, RuntimeWarning)
[W compiler_depend.ts:623] Warning: expandable_segments currently defaults to false. You can enable this feature by `export PYTORCH_NPU_ALLOC_CONF = expandable_segments:True`. (function operator())
[W compiler_depend.ts:631] Warning: expandable_segments feature is not supportted                    and the possible cause is that driver and firmware packages do not match. (function operator())
Starting rank=0, seed=0, world_size=1.
[2024-09-11 14:34:12] Experiment directory created at results/001-DiT-XL-2
[2024-09-11 14:34:26] DiT Parameters: 675,129,632
[2024-09-11 14:34:27] Dataset contains 34,745 images (/efs_guiyang/r30049464/Dit/imagenet-mini/train/)
[2024-09-11 14:34:27] Training for 1 epochs...
[2024-09-11 14:34:27] Beginning epoch 0...
/efs_guiyang/r30049464/Dit/DiT/models.py:86: UserWarning: AutoNonVariableTypeMode is deprecated and will be removed in 1.10 release. For kernel implementations please use AutoDispatchBelowADInplaceOrView instead, If you are looking for a user facing API to enable running your inference-only workload, please use c10::InferenceMode. Using AutoDispatchBelowADInplaceOrView in user code is under risk of producing silent wrong result in some edge cases. See Note [AutoDispatchBelowAutograd] for more details. (Triggered internally at build/CMakeFiles/torch_npu.dir/compiler_depend.ts:74.)
  labels = torch.where(drop_ids, self.num_classes, labels)
[2024-09-11 14:34:36] (step=0000001) Train Loss: 0.9857, Train Steps/Sec: 0.12
[2024-09-11 14:34:36] (step=0000002) Train Loss: 0.9872, Train Steps/Sec: 4.97
[2024-09-11 14:34:36] (step=0000003) Train Loss: 0.9830, Train Steps/Sec: 5.20
[2024-09-11 14:34:36] (step=0000004) Train Loss: 0.9862, Train Steps/Sec: 5.21
[2024-09-11 14:34:37] (step=0000005) Train Loss: 0.9759, Train Steps/Sec: 5.21
[2024-09-11 14:34:37] (step=0000006) Train Loss: 0.8968, Train Steps/Sec: 5.20
[2024-09-11 14:34:37] (step=0000007) Train Loss: 1.0263, Train Steps/Sec: 5.24
[2024-09-11 14:34:37] (step=0000008) Train Loss: 0.9263, Train Steps/Sec: 5.19
[2024-09-11 14:34:37] (step=0000009) Train Loss: 0.8757, Train Steps/Sec: 5.20
[2024-09-11 14:34:38] (step=0000010) Train Loss: 0.8448, Train Steps/Sec: 5.22
[2024-09-11 14:34:38] (step=0000011) Train Loss: 0.8599, Train Steps/Sec: 5.23
[2024-09-11 14:34:38] (step=0000012) Train Loss: 0.7933, Train Steps/Sec: 5.24
```

----结束

10.6.3 精度对齐

10.6.3.1 长训 Loss 比对结果

在单卡环境下，执行一个Epoch训练任务，GPU和NPU训练叠加效果如下：



上图中的红色曲线为GPU Loss折线图，蓝色曲线为NPU训练Loss折线图。在整网训练单个Epoch情况下，Loss总体的绝对偏差大约为0.08181。

10.6.3.2 使用 Msprobe 工具分析偏差

观察上一章Loss趋势，在首个Step有较小偏差，所以对第一个Step进行比对分析。此处使用Msprobe的整网Dump和比对分析功能。

1. 首先安装社区Msprobe工具，命令如下：
   ```
   pip install mindstudio-probe
   ```
2. 使能工具进行数据Dump分析。本实验可在train.py中如下两处添加使能代码：

```python
# Copyright (c) Meta Platforms, Inc. and affiliates.
# All rights reserved.

# This source code is licensed under the license found in the
# LICENSE file in the root directory of this source tree.

"""
A minimal training script for DiT using PyTorch DDP.
"""
import torch
# the first flag below was False when we tested this script but True makes A100 training a lot faster:
torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
from torchvision.datasets import ImageFolder
from torchvision import transforms
import numpy as np
from collections import OrderedDict
from PIL import Image
from copy import deepcopy
from glob import glob
from time import time
import argparse
import logging
import os

from models import DiT_models
from diffusion import create_diffusion
from diffusers.models import AutoencoderKL

try:
    import torch_npu
    from torch_npu.contrib import transfer_to_npu
except ImportError:
    print('package torch_npu not found.')

from msprobe.pytorch import PrecisionDebugger
debugger = PrecisionDebugger('./config.json')
#################################################################################
#                             Training Helper Functions                         #
#################################################################################
```

```python
    logger.info(f"Training for {args.epochs} epochs...")
    for epoch in range(args.epochs):
        sampler.set_epoch(epoch)
        logger.info(f"Beginning epoch {epoch}...")
        for x, y in loader:

            debugger.start()

            x = x.to(device)
            y = y.to(device)
            with torch.no_grad():
                # Map input images to latent space + normalize latents:
                x = vae.encode(x).latent_dist.sample().mul_(0.18215)
            t = torch.randint(0, diffusion.num_timesteps, (x.shape[0],), device=device)
            model_kwargs = dict(y=y)
            loss_dict = diffusion.training_losses(model, x, t, model_kwargs)
            loss = loss_dict["loss"].mean()
            opt.zero_grad()
            loss.backward()
            opt.step()
            update_ema(ema, model.module)

            debugger.stop()
            debugger.step()

            # Log loss values:
            running_loss += loss.item()
            log_steps += 1
            train_steps += 1
            if train_steps % args.log_every == 0:
                # Measure training speed:
                torch.cuda.synchronize()
                end_time = time()
                steps_per_sec = log_steps / (end_time - start_time)
                # Reduce loss history over all processes:
                avg_loss = torch.tensor(running_loss / log_steps, device=device)
                dist.all_reduce(avg_loss, op=dist.ReduceOp.SUM)
                avg_loss = avg_loss.item() / dist.get_world_size()
                logger.info(f"(step={train_steps:07d}) Train Loss: {avg_loss:.4f}, Train Steps/Sec: {steps_per_sec:.2f}")
                # Reset monitoring variables:
                running_loss = 0
                log_steps = 0
                start_time = time()
```

其中config.json的内容如下：

```
{
"task": "statistics",
"dump_path": "/home/data_dump",
"rank": [],
"step": [0],
"level": "L1",
"seed": 1234,
"is_deterministic": false,
"enable_dataloader": false,
"statistics": {
"scope": [],
"list": [],
"data_mode": ["all"],
"summary_mode": "statistics"
```

```
}
}
```

这里Step指定为0表示只对首个Step进行数据Dump。task指定为statistics表示使用统计量模式，该模式下针对整网训练API输入输出保存最大值、最小值、均值等统计量信息比对，落盘数据量较小。GPU和NPU环境依次进行数据Dump，正常执行结束标识如下图回显Exception: msprobe: exit after iteration 0。

```
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.572.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.865.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.573.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.866.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.574.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.867.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.575.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.868.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.576.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.869.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.577.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.870.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.578.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.871.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.579.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.872.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.580.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.873.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.581.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.mul_.874.forward.
2024-09-11 15:28:12 (854755) [INFO] msprobe is collecting data on Tensor.add_.582.forward.
2024-09-11 15:28:12 (854755) [INFO] dump.json is at /efs_guiyang/r30049464/Dit/dump_data/npu_dump_V1/step0.
(step=0000001) Train Loss: 1.0110, Train Steps/Sec: 0.03
Traceback (most recent call last):
  File "/efs_guiyang/r30049464/Dit/DiT/train.py", line 282, in <module>
    main(args)
  File "/efs_guiyang/r30049464/Dit/DiT/train.py", line 207, in main
    debugger.start()
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/msprobe/pytorch/debugger/precision_debugger.py", line 75, in start
    instance.service.start(instance.model, instance.api_origin)
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/msprobe/pytorch/service.py", line 127, in start
    raise Exception("msprobe: exit after iteration {}".format(max(self.config.step)))
Exception: msprobe: exit after iteration 0
[ERROR] 2024-09-11-15:28:16 (PID:854755, Device:0, RankID:0) ERR99999 UNKNOWN application exception
[2024-09-11 15:28:26,725] torch.distributed.elastic.multiprocessing.api: [ERROR] failed (exitcode: 1) local_rank: 0 (pid: 854755) of binary: /home/ma-user/anaconda3/envs/PyTorch-2.1.0/bin/python3.9
Traceback (most recent call last):
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/bin/torchrun", line 8, in <module>
    sys.exit(main())
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch/distributed/elastic/multiprocessing/errors/__init__.py", line 346, in wrapper
    return f(*args, **kwargs)
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch/distributed/run.py", line 806, in main
    run(args)
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch/distributed/run.py", line 797, in run
    elastic_launch(
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch/distributed/launcher/api.py", line 134, in __call__
    return launch_agent(self._config, self._entrypoint, list(args))
  File "/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/torch/distributed/launcher/api.py", line 264, in launch_agent
    raise ChildFailedError(
torch.distributed.elastic.multiprocessing.errors.ChildFailedError:
============================================================
```

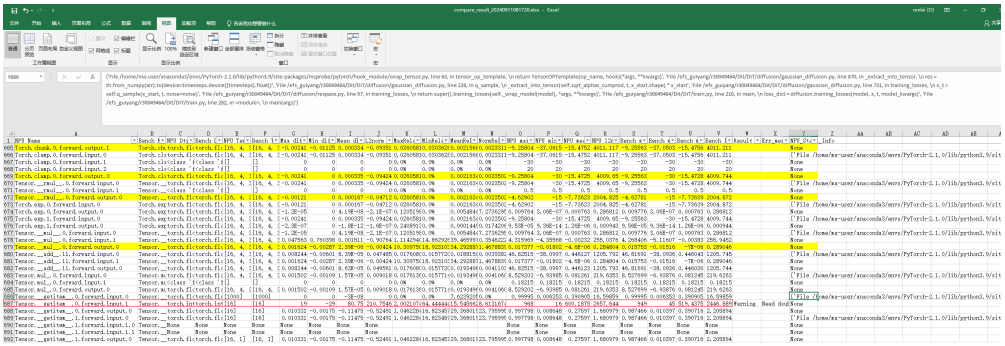
3. 创建如下compare.json文件。

```
{
"npu_path": "./npu_dump/dump.json",
"bench_path": "./bench_dump/dump.json",
"stack_path": "./npu_dump/stack.json",
"is_print_compare_log": true
}
```

4. 指定对应Dump数据目录后进行比对分析。

```
msprobe -f pytorch compare -i ./compare.json -o ./output -s
```

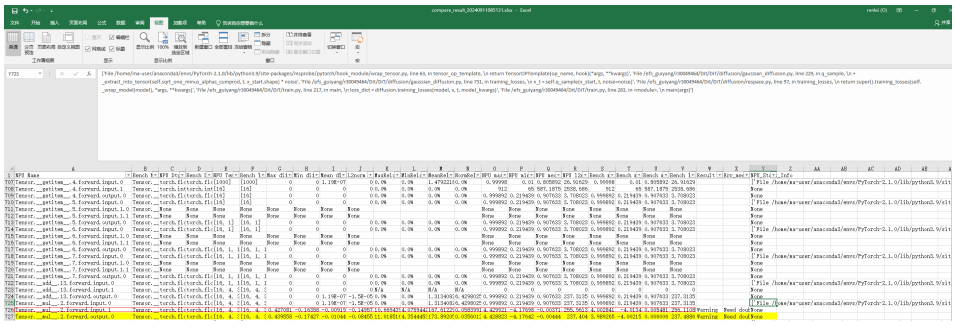
生成CSV分析表格之后进行分析，该问题第一个偏差来源如下：



Tensor.__getitem__.0 在forward阶段的第一个输入存在偏差，追溯输入来源发现是torch.randint()函数在device侧随机初始化（下图第214行），由于device侧随机性无法通过seed等自动化方式固定，先通过切换CPU侧计算初始化之后再切回device侧。在train.py中做如下图第215行代码修改。

```
189
190     # Prepare models for training:
191     update_ema(ema, model.module, decay=0)  # Ensure EMA is initialized with synced weights
192     model.train()  # important! This enables embedding dropout for classifier-free guidance
193     ema.eval()  # EMA model should always be in eval mode
194
195     # Variables for monitoring/logging purposes:
196     train_steps = 0
197     log_steps = 0
198     running_loss = 0
199     start_time = time()
200
201     logger.info(f"Training for {args.epochs} epochs...")
202     for epoch in range(args.epochs):
203         sampler.set_epoch(epoch)
204         logger.info(f"Beginning epoch {epoch}...")
205         for x, y in loader:
206
207             debugger.start()
208
209             x = x.to(device)
210             y = y.to(device)
211             with torch.no_grad():
212                 # Map input images to latent space + normalize latents:
213                 x = vae.encode(x).latent_dist.sample().mul_(0.18215)
214             # t = torch.randint(0, diffusion.num_timesteps, (x.shape[0],), device=device)
215             t = torch.randint(0, diffusion.num_timesteps, (x.shape[0],), device="cpu").to(device)
216             model_kwargs = dict(y=y)
217             loss_dict = diffusion.training_losses(model, x, t, model_kwargs)
218             loss = loss_dict["loss"].mean()
```

5. 重新训练Dump比对分析后续计算是否存在偏差。比对之后发现：Tensor.__mul__.2在forward计算阶段的第一个input存在偏差。

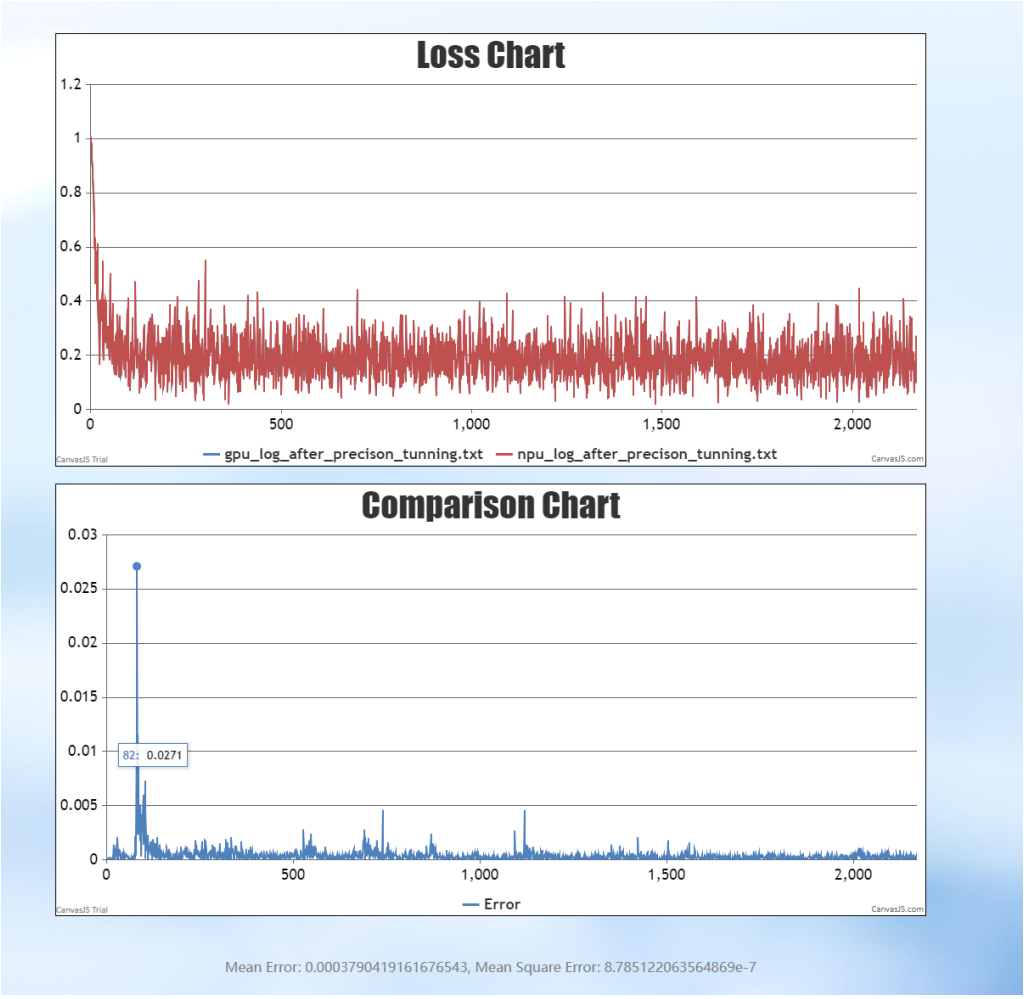


追溯代码实现是下图中noise变量使用torch.rand_like ()作noise变量的初始化（下图第730行）。由于torch.rand_like()该函数会根据输入的input构造同样size、dtype、device、layout信息的数据，详情请参见**PyTorch docs介绍**。所以同样是在device侧做变量初始化引入精度偏差，在diffusion/gaussian_diffusion.py中用等CPU侧初始化实现替换完成计算之后再切回device进行计算(下图第731行)。

```
709
710         # At the first timestep return the decoder NLL,
711         # otherwise return KL(q(x_{t-1}|x_t,x_0) || p(x_{t-1}|x_t))
712         output = th.where((t == 0), decoder_nll, kl)
713         return {"output": output, "pred_xstart": out["pred_xstart"]}
714
715     def training_losses(self, model, x_start, t, model_kwargs=None, noise=None):
716         """
717         Compute training losses for a single timestep.
718         :param model: the model to evaluate loss on.
719         :param x_start: the [N x C x ...] tensor of inputs.
720         :param t: a batch of timestep indices.
721         :param model_kwargs: if not None, a dict of extra keyword arguments to
722             pass to the model. This can be used for conditioning.
723         :param noise: if specified, the specific Gaussian noise to try to remove.
724         :return: a dict with the key "loss" containing a tensor of shape [N].
725                  Some mean or variance settings may also have other keys.
726         """
727         if model_kwargs is None:
728             model_kwargs = {}
729         if noise is None:
730             #noise = th.randn_like(x_start)
731             noise = th.randn(x_start.size(), dtype=x_start.dtype, layout=x_start.layout, device="cpu").to(x_start.device)
732         x_t = self.q_sample(x_start, t, noise=noise)
733
```

然后再比对分析发现所有API计算都已对齐结果，转而查看Loss对齐情况。

10.6.3.3 Loss 对齐结果



在排查完精度偏差来源之后发现，Loss最大绝对偏差减少为0.0003，Loss结果对齐。

需要注意训练引入随机性的目的是为了增加结果的鲁棒性，理论上不会对训练模型的收敛与否造成影响。

此处做随机性固定主要的目的是为了训练结果可完全复现，从而实现NPU和标杆的精度对齐。

10.6.4 性能调优

10.6.4.1 Profiling 数据采集

在train.py的main()函数Step迭代处添加配置，添加位置如下图所示：

```
experimental_config = torch_npu.profiler._ExperimentalConfig(
                aic_metrics=torch_npu.profiler.AiCMetrics.PipeUtilization,
                profiler_level=torch_npu.profiler.ProfilerLevel.Level1,
                l2_cache=False,
                data_simplification=False)

with torch_npu.profiler.profile(
                activities=[
                    torch_npu.profiler.ProfilerActivity.CPU,
                    torch_npu.profiler.ProfilerActivity.NPU],
                schedule=torch_npu.profiler.schedule(wait=1, warmup=1, active=1, repeat=1, skip_first=10),
                on_trace_ready=torch_npu.profiler.tensorboard_trace_handler("./result"),
                record_shapes=True,
                profile_memory=True,
                with_stack=True,
                experimental_config=experimental_config) as prof:
    for epoch in range(args.epochs):
        sampler.set_epoch(epoch)
        logger.info(f"Beginning epoch {epoch}...")
        for x, y in loader:

            #debugger.start()

            x = x.to(device)
            y = y.to(device)
            with torch.no_grad():
                # Map input images to latent space + normalize latents:
                x = vae.encode(x).latent_dist.sample().mul_(0.18215)
            #t = torch.randint(0, diffusion.num_timesteps, (x.shape[0],), device=device)
            t = torch.randint(0, diffusion.num_timesteps, (x.shape[0],), device="cpu").to(device)
            model_kwargs = dict(y=y)
            loss_dict = diffusion.training_losses(model, x, t, model_kwargs)
            loss = loss_dict["loss"].mean()
            opt.zero_grad()
            loss.backward()
            opt.step()
            update_ema(ema, model.module)

            #debugger.stop()
            #debugger.step()

            # Log loss values:
            running_loss += loss.item()
            log_steps += 1
            train_steps += 1
            if train_steps % args.log_every == 0:
                # Measure training speed:
                torch.cuda.synchronize()
                end_time = time()
                steps_per_sec = log_steps / (end_time - start_time)
                # Reduce loss history over all processes:
                avg_loss = torch.tensor(running_loss / log_steps, device=device)
                dist.all_reduce(avg_loss, op=dist.ReduceOp.SUM)
                avg_loss = avg_loss.item() / dist.get_world_size()
                logger.info(f"(step={train_steps:07d}) Train Loss: {avg_loss:.4f}, Train Steps/Sec: {steps_per_sec:.2f}")
                # Reset monitoring variables:
                running_loss = 0
                log_steps = 0
                start_time = time()

            # Save DiT checkpoint:
            if train_steps % args.ckpt_every == 0 and train_steps > 0:
                if rank == 0:
                    checkpoint = {
                        "model": model.module.state_dict(),
                        "ema": ema.state_dict(),
                        "opt": opt.state_dict(),
                        "args": args
                    }
                    checkpoint_path = f"{checkpoint_dir}/{train_steps:07d}.pt"
                    torch.save(checkpoint, checkpoint_path)
                    logger.info(f"Saved checkpoint to {checkpoint_path}")
                dist.barrier()
            prof.step()

    model.eval()  # important! This disables randomized embedding dropout
    # do any sampling/FID calculation/etc. with ema (or model) in eval mode ...
```

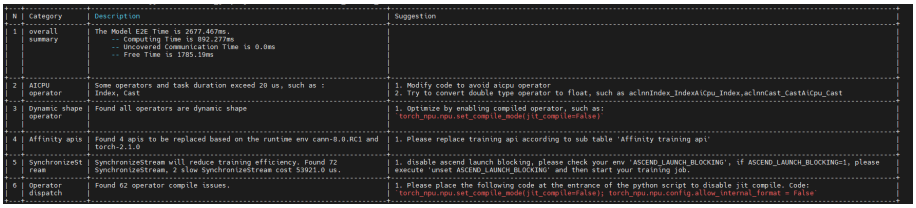
此处需要注意的是prof.step()需要加到dataloder迭代循环的内部以保证采集单个Step迭代的Profiling数据。

更多信息，请参见**Ascend PyTorch Profiler接口采集**。

10.6.4.2 使用 Advisor 工具分析生成调优建议

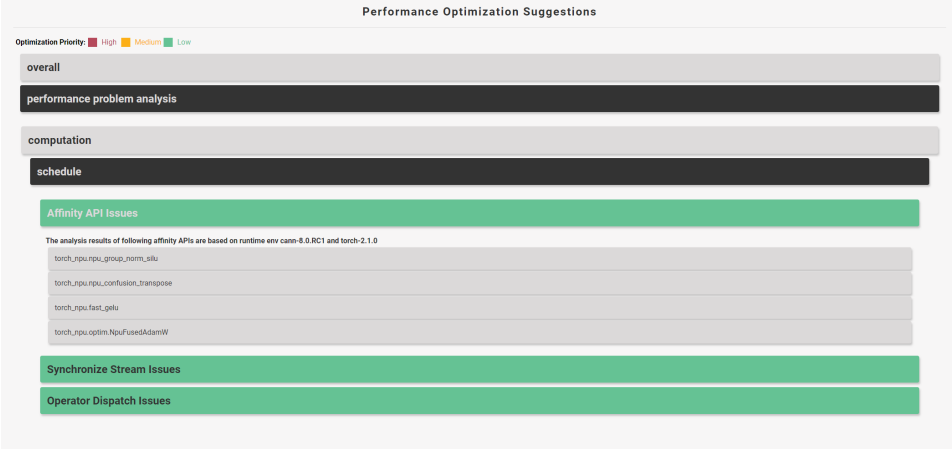
关于Advisor使用及安装过程请参见**昇腾社区Gitee**。最后生成导出的各类场景的建议包含以下两种：

1. Terminal日志信息的概览建议。



N	Category	Description	Suggestion
1	overall summary	The Model E2E Time is 2677.467ms -- Computing Time is 892.277ms -- Uncovered Communication Time is 0.0ms -- Free Time is 1785.19ms	
2	AICPU operator	Some operators and task duration exceed 20 us, such as : Index, Cast	1. Modify code to avoid aicpu operator 2. Try to convert double type operator to float, such as aclnnIndex_IndexAiCpu_Index,aclnnCast_CastAiCpu_Cast
3	Dynamic shape operator	Found all operators are dynamic shape	1. Optimize by enabling compiled operator, such as: 'torch_npu.npu.set_compile_mode(jit_compile=False)'
4	Affinity apis	Found 4 apis to be replaced based on the runtime env cann-8.0.RC1 and torch-2.1.0	1. Please replace training api according to sub table 'Affinity training api'
5	SynchronizeStream	SynchronizeStream will reduce training efficiency. Found 72 SynchronizeStream, 2 slow SynchronizeStream cost 53921.0 us.	1. disable ascend launch blocking, please check your env 'ASCEND_LAUNCH_BLOCKING', if ASCEND_LAUNCH_BLOCKING=1, please execute 'unset ASCEND_LAUNCH_BLOCKING' and then start your training job.
6	Operator dispatch	Found 62 operator compile issues.	1. Please place the following code at the entrance of the python script to disable jit compile. Code: 'torch_npu.npu.set_compile_mode(jit_compile=False); torch_npu.npu.config.allow_internal_format = False'

2. 包含Detail信息及修改示例的HTML信息。



按照建议信息做如下修改：

亲和优化器使能，在train.py中修改优化器为apex混合精度模式下的DDP优化方式（修改点：注释第161和167行，增加第168~170行）。

```
136     print(f"Starting rank={rank}, seed={seed}, world_size={dist.get_world_size()}.")
137
138     # Setup an experiment folder:
139     if rank == 0:
140         os.makedirs(args.results_dir, exist_ok=True)  # Make results folder (holds all experiment subfolders)
141         experiment_index = len(glob(f"{args.results_dir}/*"))
142         model_string_name = args.model.replace("/", "-")  # e.g., DiT-XL/2 --> DiT-XL-2 (for naming folders)
143         experiment_dir = f"{args.results_dir}/{experiment_index:03d}-{model_string_name}"  # Create an experiment folder
144         checkpoint_dir = f"{experiment_dir}/checkpoints"  # Stores saved model checkpoints
145         os.makedirs(checkpoint_dir, exist_ok=True)
146         logger = create_logger(experiment_dir)
147         logger.info(f"Experiment directory created at {experiment_dir}")
148     else:
149         logger = create_logger(None)
150
151     # Create model:
152     assert args.image_size % 8 == 0, "Image size must be divisible by 8 (for the VAE encoder)."
153     latent_size = args.image_size // 8
154     model = DiT_models[args.model](
155         input_size=latent_size,
156         num_classes=args.num_classes
157     )
158     # Note that parameter initialization is done within the DiT constructor
159     ema = deepcopy(model).to(device)  # Create an EMA of the model for use after training
160     requires_grad(ema, False)
161     # model = DDP(model.to(device), device_ids=[rank])
162     diffusion = create_diffusion(timestep_respacing="")  # default: 1000 steps, linear noise schedule
163     vae = AutoencoderKL.from_pretrained(f"/efs_guiyang/r30049464/models/sd-vae-ft-mse").to(device)
164     logger.info(f"DiT Parameters: {sum(p.numel() for p in model.parameters()):,}")
165
166     # Setup optimizer (we used default Adam betas=(0.9, 0.999) and a constant learning rate of 1e-4 in our paper):
167     # opt = torch.optim.AdamW(model.parameters(), lr=1e-4, weight_decay=0)
168     opt = apex.optimizers.NpuFusedAdamW(model.parameters(), lr=1e-4, weight_decay=0)
169     model, opt = apex.amp.initialize(model.to(device), opt, opt_level="O1", combine_grad=True, combine_ddp=True)
170     model = DDP(model.to(device), device_ids=[rank])
```

二进制调优使能，减少算子编译耗时，在train.py头文件导入之后添加（修改点：增加第37行）。

```
torch_npu.npu.set_compile_mode(jit_compile=False)
```

```python
# Copyright (c) Meta Platforms, Inc. and affiliates.
# All rights reserved.

# This source code is licensed under the license found in the
# LICENSE file in the root directory of this source tree.

"""
A minimal training script for DiT using PyTorch DDP.
"""
import torch
# the first flag below was False when we tested this script but True makes A1
torch.backends.cuda.matmul.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.utils.data import DataLoader
from torch.utils.data.distributed import DistributedSampler
from torchvision.datasets import ImageFolder
from torchvision import transforms
import numpy as np
from collections import OrderedDict
from PIL import Image
from copy import deepcopy
from glob import glob
from time import time
import argparse
import logging
import os

from models import DiT_models
from diffusion import create_diffusion
from diffusers.models import AutoencoderKL
import apex
try:
    import torch_npu
    from torch_npu.contrib import transfer_to_npu
    torch_npu.npu.set_compile_mode(jit_compile=False)
except ImportError:
    print('package torch_npu not found.')
```

AICPU算子调优，Double类型输入切换成为Float减少cast算子调用耗时，修改diffusion/gaussian_diffusion.py (修改点：注释第871行，增加第872行)。

```python
def _extract_into_tensor(arr, timesteps, broadcast_shape):
    """
    Extract values from a 1-D numpy array for a batch of indices.
    :param arr: the 1-D numpy array.
    :param timesteps: a tensor of indices into the array to extract.
    :param broadcast_shape: a larger shape of K dimensions with the batch
                            dimension equal to the length of timesteps.
    :return: a tensor of shape [batch_size, 1, ...] where the shape has K dims.
    """
    # res = th.from_numpy(arr).to(device=timesteps.device)[timesteps].float()
    res = th.from_numpy(arr.astype(np.float32)).to(device=timesteps.device)[timesteps]
    while len(res.shape) < len(broadcast_shape):
        res = res[..., None]
    return res + th.zeros(broadcast_shape, device=timesteps.device)
```

10.6.4.3 调优前后性能对比

在完成上一章几类调优方式之后，在单卡场景下实测性能调优比对结果如下表所示：

设备	batch_size	Steps/Sec
1p-GPU Ant8	16	3.17
1p-NPU snt9b 313T	16	2.17
1p-NPU snt9b 313T调优后	16	2.58

10.7 msprobe 工具使用指导

10.7.1 msprobe API 预检

msprobe是MindStudio Training Tools工具链下精度调试部分的工具包，主要包括精度预检、溢出检测和精度比对等功能，目前适配PyTorch和MindSpore框架。这些子工具侧重不同的训练场景，可以定位模型训练中的精度问题。

精度预检工具旨在计算单个API在整网计算中和标杆场景下的差异，对于无明确精度差异来源情况或者对模型了解不多的情形下都推荐使用预检工具，检查第一个步骤或Loss明显出现问题的步骤。它可以抓取模型中API输入的数值范围，根据范围随机生成输入，用相同的输入分别在NPU（GPU）和CPU上执行算子，比较输出差异。预检最大的好处是，它能根据算子（API）的精度标准来比较输出结果并判定其是否有精度问题。预检工具使用包含以下三步：dump、run_ut以及api_precision_compare。基本步骤如下：

步骤1 通过pip安装msprobe工具。

```
# shell
pip install mindstudio-probe
```

步骤2 获取NPU和GPU的dump数据。

PyTorch训练脚本插入dump接口方式如下：

```
from msprobe.pytorch import PrecisionDebugger
debugger = PrecisionDebugger(config_path='./config.json')
...
debugger.start() # 一般在训练循环开头启动工具。
... # 循环体
debugger.stop() # 一般在训练循环末尾结束工具。
debugger.step() # 在训练循环的最后需要重置工具，非循环场景不需要。
```

具体的config.json的配置要求请参见介绍。

步骤3 使用run_ut.py执行预检。

```
msprobe -f pytorch run_ut -api_info ./dump.json
```

这里-api_info指定的是步骤2导出的dump.json文件，表示整网计算过程中API的输入输出情况。执行完成run_ut命令之后将输出api_precision_compare_result_{timestamp}.csv和api_precision_compare_details_{timestamp}.csv文。accuracy_checking_result_{timestamp}.csv属于API级，标明每个API是否通过测试。建议用户先查看accuracy_checking_result_{timestamp}.csv文件，对于其中没有通过测试的或者特定感兴趣的API，根据其API name字段在accuracy_checking_details_{timestamp}.csv中查询其各个输出的达标情况以及比较指标。

步骤4 比对NPU和GPU预检结果。

```
msprobe -f pytorch api_precision_compare -npu /home/xxx/npu/accuracy_checking_details_{timestamp}.csv -gpu /home/xxx/gpu/accuracy_checking_details_{timestamp}.csv -o /home/xxx
```

这里-npu指定步骤3NPU预检结果中accuracy_checking_details_{timestamp}.csv文件路径，-gpu指定GPU预检结果accuracy_checking_details_{timestamp}.csv文件路径，-o需指定执行比对结果的存盘路径。执行完成后输出api_precision_compare_result_{timestamp}.csv和api_precision_compare_details_{timestamp}.csv文件。用户可以通过先查看

api_precision_compare_result_{timestamp}.csv文件的Forward Test Success和Backward Test Success，判断是否存在未通过测试的API，再查看api_precision_compare_details_{timestamp}.csv文件的API详细达标情况。

----结束

详细工具的使用指导请参考**离线预检**和**在线预检**介绍。

10.7.2 msprobe 精度比对

精度比对功能主要针对两类场景的问题：

1. 同一模型，从CPU或GPU移植到NPU中存在精度下降问题，对比NPU芯片中的API计算数值与CPU或GPU芯片中的API计算数值，进行问题定位。
2. 同一模型，进行迭代（模型、框架版本升级或设备硬件升级）时存在的精度下降问题，对比相同模型在迭代前后版本的API计算数值，进行问题定位。

首先通过在PyTorch训练脚本中插入dump接口，跟踪计算图中算子的前向传播与反向传播时的输入与输出，然后再使用子命令compare进行比对生成比对表格。当前比对结果支持计算Cosine（余弦相似度）、MaxAbsErr（最大绝对误差）和MaxRelativeErr（最大相对误差）、One Thousandth Err Ratio（双千分之一）和Five Thousandths Err Ratio（双千分之五）这几种评价指标，工具通过阈值过滤筛选出不达标API的输入输出提示用户进行重点关注。使用步骤如下：

步骤1 通过pip安装msprobe工具。

```
# shell
pip install mindstudio-probe
```

步骤2 获取NPU和标杆的dump数据。

PyTorch训练脚本插入dump接口方式如下：

```
from msprobe.pytorch import PrecisionDebugger
debugger = PrecisionDebugger(config_path='./config.json')
...
debugger.start() # 一般在训练循环开头启动工具。
... # 循环体
debugger.stop() # 一般在训练循环末尾结束工具。
debugger.step() # 在训练循环的最后需要重置工具，非循环场景不需要。
```

具体的config.json的配置要求请参见**介绍**。

步骤3 创建比对compare.json文件。

- 单卡场景

  ```
  {
  "npu_path": "./npu_dump/dump.json",
  "bench_path": "./bench_dump/dump.json",
  "stack_path": "./npu_dump/stack.json",
  "is_print_compare_log": true
  }
  ```

- 多卡场景

  ```
  {
  "npu_path": "./npu_dump/step0",
  "bench_path": "./bench_dump/step0",
  "is_print_compare_log": true
  }
  ```

单卡场景npu_path、bench_path、stack_path分别表示从**步骤2**中NPU环境所生成的dump.json、标杆环境生成的dump.json及NPU环境生成的stack.json文件，is_print_compare_log配置是否开启日志打屏。

多卡场景区别于单卡场景会在**步骤2**按rank标号信息生成多个rank的dump文件结果，npu_path指定NPU环境生成包含多rank目录，gpu_path指定标杆环境包含多rank目录，is_print_compare_log配置是否开启日志打屏。

步骤4 精度比对生成比对报告

```
msproe -f pytorch compare -i ./compare.json -o ./output -s
```

这里-i指定**步骤3**所创建compare.json文件，-o指定比对结果文件存盘目录，-s配置是否生成堆栈信息。生成结果为advisor_{timestamp}.txt和compare_result_{timestamp}.xlsx文件，advisor_{timestamp}.txt列出了可能存在精度问题的API的专家建议，compare_result_{timestamp}.xlsx文件列出了所有执行精度比对的API详细信息和比对结果。

----结束

详细工具的使用指导请参考**Pytorch精度比对**介绍。

10.7.3 msprobe 梯度监控

梯度监控工具提供了将模型梯度数据导出的能力。使用梯度监控工具，可以实现对训练过程模型每一层梯度信息进行监控，目前支持两种能力：

1. 将模型权重的梯度数据导出。这种功能可以将模型权重的梯度值以统计量的形式采集出来，用以分析问题，例如检测确定性问题，使用训练状态监控工具监控NPU训练过程中的确定性计算问题。
2. 将两份梯度数据进行相似度对比。在有标杆问题中，可以确认训练过程中精度问题出现的Step，以及抓取反向过程中的问题。

使用步骤如下：

步骤1 通过pip安装msprobe工具。

```
# shell
pip install mindstudio-probe
```

步骤2 创建配置文件config.json。

```
{
"task": "grad_probe",
"dump_path": "./dump_path",
"rank": [],
"step": [],
"grad_probe": {
    "grad_level": "L1",
    "param_list": [],
    "bounds": [-1, 0, 1]
}
}
```

task参数需指定为grad_probe，dump_path表示输出目录，需手工指定，默认输出到dump_path目录。参数grad_level可取值L0、L1、L2，级别越大导出的数据越详细。更多详细参数说明请参考**参数说明**。

步骤3 监控逻辑插入训练脚本。

```
from msprobe.pytorch import PrecisionDebugger
debugger = PrecisionDebugger("config_json_path")
# 模型初始化之后位置添加。
debugger.monitor(model)
...
# 结束训练之后，调用stop接口。
debugger.stop()
```

步骤4 （可选）梯度数据相似度比对。

```
from msprobe import *
GradComparator.compare_distributed("配置文件里写的dump_path",
                                   "配置文件里写的dump_path",
                                   "比对结果输出目录")
```

最终生成结果为similarities.csv表示每个Step各个权重参数两次比对相似度值，以及{param_name}.png和summary_similarities.png以折线图方式表示各个Step相似度不比对结果。

----结束

详细工具的使用指导请参考**梯度状态监控工具**介绍。

11 Standard 权限管理

11.1 ModelArts 权限管理基本概念

ModelArts作为一个完备的AI开发平台，支持用户对其进行细粒度的权限配置，以达到精细化资源、权限管理之目的。这类特性在大型企业用户的使用场景下很常见，但对个人用户则显得复杂而意义不足，所以建议个人用户在使用ModelArts时，参照**个人用户快速配置ModelArts访问权限**来进行初始权限设置。

说明

您是否需要阅读本文档?

如果下述问题您的任何一个回答为“是”，则需要阅读此文档

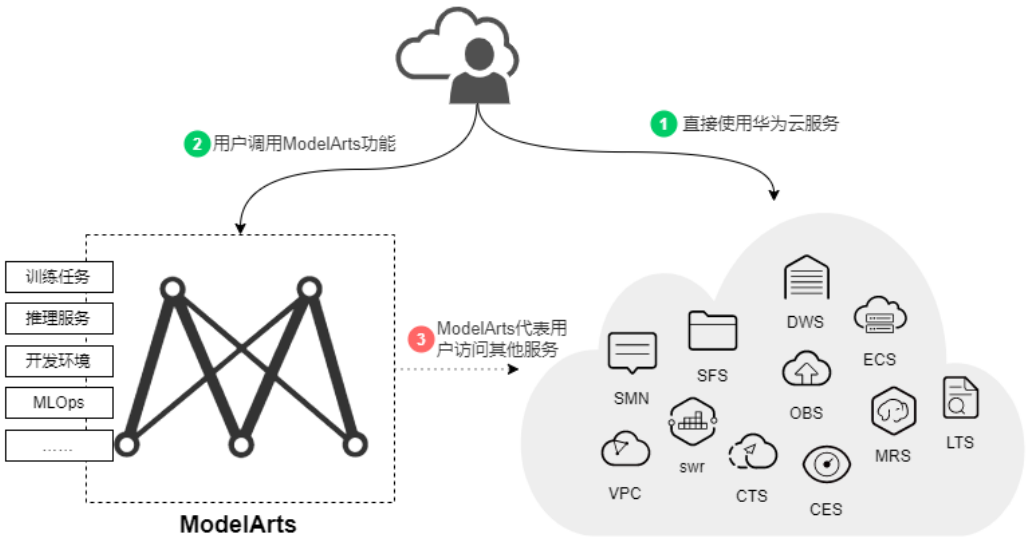
- 您是企业用户，且
  - 存在多个部门，且需要限定不同部门的用户只能访问其专属资源、功能
  - 存在多种角色（如管理员、算法开发者、应用运维），希望限制不同角色只能使用特定功能
  - 逻辑上存在多套“环境”且相互隔离（如开发环境、预生产环境、生产环境），并限定不同用户在不同环境上的操作权限
  - 其他任何需要对特定子账号（组）做出特定权限限制的情况
- 您是个人用户，但已经在IAM创建多个子账号，且期望限定不同子账号所能使用的ModelArts功能、资源不同。
- 希望了解ModelArts的权限控制能力细节，期望理解其概念和实操方法。

ModelArts的大部分权限管理能力均基于统一身份认证服务（Identity and Access Management，简称IAM）来实现，在您继续往下阅读之前，强烈建议您先行熟悉**IAM基本概念**，如果能完整理解IAM的所有概念，将更加有助于您理解本文档。

为了支持用户对ModelArts的权限做精细化控制，提供了3个方面的能力来支撑，分别是：权限、委托和工作空间。下面分别讲解。

理解 ModelArts 的权限与委托

图 11-1 权限管理抽象



ModelArts与其他服务类似，功能都通过IAM的权限来进行控制。比如，用户（此处指IAM子账号，而非租户）希望在ModelArts创建训练作业，则该用户必须拥有"modelarts:trainJob:create" 的权限才可以完成操作（无论界面操作还是API调用）。关于如何给一个用户赋权（准确讲是需要先将用户加入用户组，再面向用户组赋权），可以参考IAM的文档《**权限管理**》。

而ModelArts还有一个特殊的地方在于，为了完成AI计算的各种操作，AI平台在任务执行过程中需要访问用户的其他服务，典型的就是训练过程中，需要访问OBS读取用户的训练数据。在这个过程中，就出现了ModelArts“代表”用户去访问其他云服务的情形。从安全角度出发，ModelArts代表用户访问任何云服务之前，均需要先获得用户的授权，而这个动作就是一个“委托”的过程。用户授权ModelArts再代表自己访问特定的云服务，以完成其在ModelArts平台上执行的AI计算任务。

综上，对于**图1 权限管理抽象**可以做如下解读：

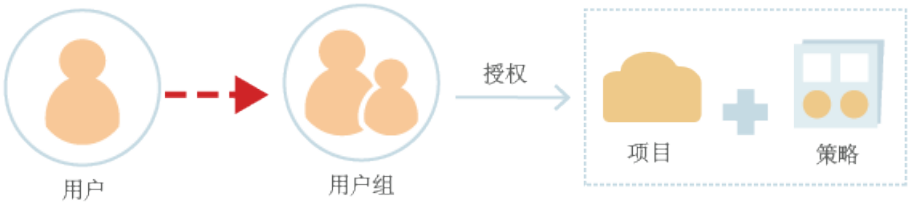
- 用户访问任何云服务，均是通过标准的IAM权限体系进行访问控制。用户首先需要具备相关云服务的权限（根据您具体使用的功能不同，所需的相关服务权限亦有差异）。
- **权限**：用户使用ModelArts的任何功能，亦需要通过IAM权限体系进行正确权限授权。
- **委托**：ModelArts上的AI计算任务执行过程中需要访问其他云服务，此动作需要获得用户的委托授权。

ModelArts 权限管理

默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于授予的权限对云服务进行操作。

⚠ 注意

- ModelArts部署时通过物理区域划分，为项目级服务，授权时“选择授权范围方案”可以选择“指定区域项目资源”，如果授权时指定了区域（如华北-北京4）对应的项目（cn-north-4），则该权限仅对此项目生效；简单的做法是直接选择“所有资源”。
- ModelArts也支持企业项目，所以选择授权范围方案时，也可以指定企业项目。具体操作参见《**创建用户组并授权**》。



IAM在对用户组授权的时候，并不是直接将具体的某个权限进行赋权，而是需要先将权限加入到“策略”当中，再把策略赋给用户组。为了方便用户的权限管理，各个云服务都提供了一些预置的“系统策略”供用户直接使用。如果预置的策略不能满足您的细粒度权限控制要求，则可以通过“自定义策略”来进行精细控制。

表11-1列出了ModelArts的所有预置系统策略。

表 11-1 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限	系统策略

通常来讲，只给管理员开通“ModelArts FullAccess”，如果不需要太精细的控制，直接给所有用户开通“ModelArts CommonOperations”即可满足大多数小团队的开发场景诉求。如果您希望通过自定义策略做深入细致的权限控制，请阅读**ModelArts的IAM权限控制详解**。

说明

ModelArts的权限不会凌驾于其他服务的权限之上，当您给用户进行ModelArts赋权时，系统不会自动对其他相关服务的相关权限进行赋权。这样做的好处是更加安全，不会出现预期外的“越权”，但缺点是，您必须同时给用户赋予不同服务的权限，才能确保用户可以顺利完成某些ModelArts操作。

举例，如果用户需要用OBS中的数据进行训练，当已经为IAM用户配置ModelArts训练权限时，仍需同时为其配置对应的OBS权限（读、写、列表），才可以正常使用。其中OBS的列表权限用于支持用户从ModelArts界面上选择要进行训练的数据路径；读权限主要用于数据的预览以及训练任务执行时的数据读取；写权限则是为了保存训练结果和日志。

- 对于个人用户或小型组织，一个简单做法是为IAM用户配置“作用范围”为“全局级服务”的“Tenant Administrator”策略，这会使用户获得除了IAM以外的所有用户权限。在获得便利的同时，由于用户的权限较大，会存在相对较大的安全风险，需谨慎使用。（对于个人用户，其默认IAM账号就已经属于admin用户组，且具备Tenant Administrator权限，无需额外操作）
- 当您需要限制用户操作，仅为ModelArts用户配置OBS相关的最小化权限项，具体操作请参见**OBS权限管理**。对于其他云服务，也可以进行精细化权限控制，具体请参考对应的云服务文档。

ModelArts 委托授权

前文已经介绍，ModelArts在执行AI计算任务过程中，需要“代表”用户去访问其他云服务，而此动作需要提前获得用户的授权。在IAM权限体系下，此类授权动作是通过“委托”来完成。

关于委托的基本概念及操作可以参考对应的IAM文档《**委托其他云服务管理资源**》。

为了简化用户的委托授权操作，ModelArts增加了自动配置委托授权的支持，用户仅需在ModelArts控制台的“权限管理”页面中，为自己或特定用户配置委托即可。

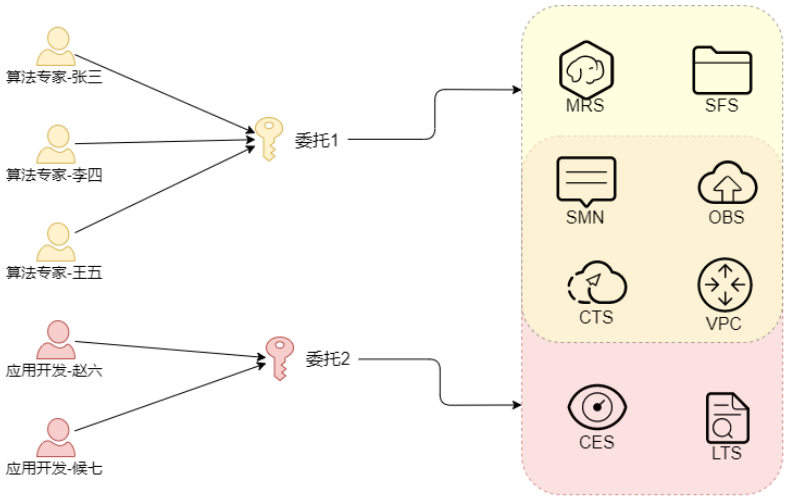
说明

- 只有具备IAM委托管理权限的用户才可以进行此项操作，通常是IAM admin用户组的成员才具备此权限。
- 目前ModelArts的委托授权操作是分区域操作的，这意味着您需要在每个您所用到的区域均执行委托授权操作。

在ModelArts控制台的“权限管理”页面，单击“添加授权”后，系统会引导您为特定用户或所有用户进行委托配置，通常默认会创建一个名为“modelarts_agency_<用户名>_随机ID”的委托条目。在权限配置的区域，您可以选择ModelArts提供的预置配置，也可以自定义选择您所授权的策略。如果这两种形态对于您的诉求均过于粗犷，您也可以直接在IAM管理页面里创建完全由您进行精细化配置的委托（需要委托给ModelArts服务），然后在此页面的委托选择里使用“已有委托”“”（而非“新增委托”）。

至此，您应该已经发现了一个细节，ModelArts在使用委托时，是将其与用户进行关联的，用户与委托的关系是多对1的关系。这意味着，如果两个用户需要配置的委托一致，那么不需要为每个用户都创建一个独立的委托项，只需要将两个用户都“指向”同一个委托项即可。

图 11-2 用户与委托对应关系



说明

每个用户必须关联委托才可以使用ModelArts，但即使委托所赋之权限不足，在API调用之初也不会报错，只有到系统具体使用到该功能时，才会发生问题。例如，用户在创建训练任务时打开了“消息通知”，该功能依赖SMN委托授权，但只有训练任务运行过程中，真正需要发送消息时，系统才会“出错”，而有些错误系统会选择“忽略”，另一些错误则可能导致任务直接失败。当您做深入的“权限最小化”限制时，请确保您在ModelArts上将要执行的操作仍旧有足够的权限。

严格授权模式

严格授权模式是指在IAM中创建的子账号必须由账号管理员显式在IAM中授权，才能访问ModelArts服务，管理员用户可以通过授权策略为普通用户精确添加所需使用的ModelArts功能的权限。

相对的，在非严格授权模式下，子账号不需要显式授权就可以使用ModelArts，管理员需要在IAM上为子账号配置Deny策略来禁止子账号使用ModelArts的某些功能。

账号的管理员用户可以在“权限管理”页面修改授权模式。

须知

如无特殊情况，建议优先使用严格授权模式。在严格授权模式下，子账号要使用ModelArts的功能都需经过授权，可以更精确的控制子账号的权限范围，达成权限最小化的安全策略。

用工作空间限制资源访问

工作空间是ModelArts面向企业用户提供的一个高阶功能，用于进一步将用户的资源划分在多个**逻辑隔离**的空间中，并支持以空间维度进行访问的权限限定。目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持经理申请开通。

在开通工作空间后，系统会默认为您创建一个“default”空间，您之前所创建的所有资源，均在该空间下。当您创建新的工作空间之后，相当于您拥有了一个新的

“ModelArts分身”，您可以通过菜单栏的左上角进行工作空间的切换，不同工作空间中的工作互不影响。

创建工作空间时，必须绑定一个企业项目。多个工作空间可以绑定到同一个企业项目，但一个工作空间**不可以**绑定多个企业项目。借助工作空间，您可以对不同用户的资源访问和权限做更加细致的约束，具体为如下两种约束：

- 只有被授权的用户才能访问特定的工作空间（在创建、管理工作空间的页面进行配置），这意味着，像数据集、算法等AI资产，均可以借助工作空间做访问的限制。
- 在前文提到的权限授权操作中，如果“选择授权范围方案”时设定为“指定企业项目资源”，那么该授权仅对绑定至该企业项目的工作空间生效。

说明

- 工作空间的约束与权限授权的约束是叠加生效的，意味着对于一个用户，必须同时拥有工作空间的访问权和训练任务的创建权限（且该权限覆盖至当前的工作空间），他才可以在这个空间里提交训练任务。
- 对于已经开通企业项目但没有开通工作空间的用户，其所有操作均相当于在“default”企业项目里进行，请确保对应权限已覆盖了名为default的企业项目。
- 对于未开通企业项目的用户，不受上述约束限制。

本章小结

对于ModelArts的权限管理，总结了如下几条关键点：

- 如果您是个人用户，则不需要考虑细粒度权限问题，您的账户默认具备使用ModelArts的所有权限。
- ModelArts平台的所有功能均通过IAM体系进行了权限管控，您可以通过标准的IAM**授权**动作，来对特定用户进行精细化的权限管控。
- 对于所有用户（包括个人用户），需要完成对ModelArts的**委托**授权（ModelArts > 权限管理 > 添加授权），才能使用特定的功能，否则会造成您的操作出现不可预期的错误。
- 对于开通了企业项目的用户，可以进一步申请开通ModelArts的**工作空间**，通过组合使用基础授权和工作空间，来达成更加复杂的权限控制目的。

11.2 权限控制方式

11.2.1 IAM

介绍ModelArts所有功能涉及到的IAM权限配置。

IAM 权限简介

如果您需要为企业中的员工设置不同的权限访问ModelArts资源，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制云服务资源的访问。如果华为账号已经能满足您的要求，不需要通过IAM对用户进行权限管理，您可以跳过本章节，不影响您使用ModelArts服务的其他功能。

IAM是提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。

通过IAM，您可以通过授权控制用户对服务资源的访问范围。例如您的员工中有负责软件开发的人员，您希望这些用户拥有ModelArts的使用权限，但是不希望这些用户拥有删除ModelArts等高危操作的权限，那么您可以使用IAM进行权限分配，通过授予用户仅能使用ModelArts，但是不允许删除ModelArts的权限，控制用户对ModelArts资源的使用范围。

关于IAM的详细介绍，请参见**IAM产品介绍**。

角色与策略权限管理

ModelArts服务支持角色与策略授权。默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

ModelArts部署时通过物理区域划分，为项目级服务。授权时，“授权范围”需要选择“指定区域项目资源”，然后在指定区域（如华北-北京1）对应的项目（cn-north-1）中设置相关权限，并且该权限仅对此项目生效；如果“授权范围”选择“所有资源”，则该权限在所有区域项目中都生效。访问ModelArts时，需要先切换至授权区域。

如**表11-2**所示，包括了ModelArts的所有系统策略权限。如果系统预置的ModelArts权限，不满足您的授权要求，可以创建自定义策略，可参考**策略JSON格式字段介绍**。

表 11-2 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限。	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限。	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限。	系统策略

ModelArts对其他云服务有依赖关系，因此在ModelArts控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，依赖服务及其预置的权限如下。

表 11-3 ModelArts 控制台依赖服务的角色或策略

控制台功能	依赖服务	需配置角色/策略
数据管理（数据集/ 数据标注/数据处理）	对象存储服务OBS	OBS Administrator
	数据湖探索DLI	DLI FullAccess
	MapReduce服务MRS	MRS Administrator

控制台功能	依赖服务	需配置角色/策略
	数据仓库服务 GaussDB(DWS)	DWS Administrator
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
开发环境 Notebook/镜像管理/弹性节点 Server	对象存储服务OBS	OBS Administrator
	凭据管理服务CSMS	CSMS ReadOnlyAccess
	云审计服务CTS	CTS Administrator
	弹性云服务器ECS	ECS FullAccess
	容器镜像服务SWR	SWR Admin
	弹性文件服务SFS	SFS Turbo FullAccess
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
算法管理/训练管理/Workflow/自动学习	对象存储服务OBS	OBS Administrator
	消息通知服务SMN	SMN Administrator
	云审计服务CTS	CTS Administrator
	企业项目管理服务EPS	EPS FullAccess
	弹性文件服务SFS	SFS ReadOnlyAccess
	容器镜像服务SWR	SWR Admin
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	虚拟私有云服务VPC	VPC FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
模型管理/在线服务/批量服务/边缘服务/边缘部署专属资源池	对象存储服务OBS	OBS Administrator
	云监控服务CES	CES ReadOnlyAccess
	消息通知服务SMN	SMN Administrator
	企业项目管理服务EPS	EPS FullAccess

控制台功能	依赖服务	需配置角色/策略
	云审计服务CTS	CTS Administrator
	云日志服务LTS	LTS FullAccess
	虚拟私有云VPC	VPC FullAccess
	容器镜像服务SWR	SWR Admin
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI Gallery	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Admin
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
弹性集群Cluster（包含Standard资源池和Lite资源池）	云审计服务CTS	CTS Administrator
	云容器引擎CCE	CCE Administrator
	裸金属服务器BMS	BMS FullAccess
	镜像服务IMS	IMS FullAccess
	数据加密服务DEW	DEW KeypairReadOnlyAccess
	虚拟私有云VPC	VPC FullAccess
	弹性云服务器ECS	ECS FullAccess
	弹性文件服务SFS	SFS Turbo FullAccess
	对象存储服务OBS	OBS Administrator
	应用运维管理服务AOM	AOM FullAccess
	标签管理服务TMS	TMS FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
	费用中心	BSS Administrator
	云硬盘EVS	EVS FullAccess

如果系统预置的权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考**ModelArts资源权限项**。

目前支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。下面为您介绍常用的ModelArts自定义策略样例。

- 示例1：授权镜像管理的权限。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- 示例2：拒绝用户创建、更新、删除专属资源池。

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循**Deny优先原则**。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "swr:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "smn:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
            ],
            "Effect": "Deny"
        }
    ]
}
```

- 示例3：多个授权项策略。

  一个自定义策略中可以包含多个授权项，且除了可以包含本服务的授权项外，还可以包含其他服务的授权项，可以包含的其他服务必须跟本服务同属性，即都是项目级服务或都是全局级服务。多个授权语句策略描述如下：

```
{
    "Version": "1.1",
    "Statement": [
        {
```



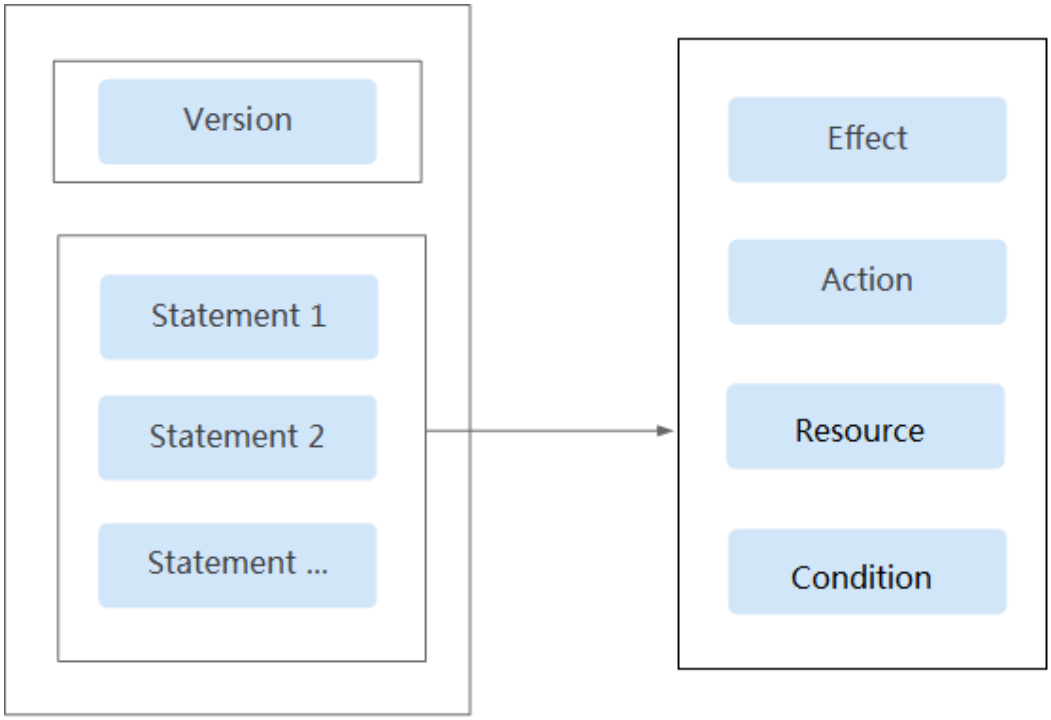
```
            "Effect": "Allow",
            "Action": [
                "modelarts:service:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lts:logs:list"
            ]
        }
    ]
}
```

策略 JSON 格式字段介绍

策略结构

策略结构包括Version（策略版本号）和Statement（策略权限语句）两部分，其中Statement可以有多个，表示不同的授权项。

图 11-3 策略结构



策略参数

下面介绍策略参数详细说明。了解策略参数后，您可以根据场景自定义策略。具体可以参考文档**自定义策略使用样例**。

表 11-4 策略参数说明

参数	含义	值
Version	策略的版本。	1.1：代表基于策略的访问控制。

参数		含义	值
Statement：策略的授权语句	Effect：作用	定义Action中的操作权限是否允许执行。	• Allow：允许执行。 • Deny：不允许执行。 **说明** 当同一个Action的Effect既有Allow又有Deny时，遵循Deny优先的原则。
	Action：授权项	操作权限。	格式为“服务名:资源类型:操作”。授权项支持通配符号*，通配符号*表示所有。 示例： "modelarts:notebook:list"：表示查看Notebook实例列表权限，其中modelarts为服务名，notebook为资源类型，list为操作。 您可以在对应服务“API参考”资料中查看该服务所有授权项。
	Condition：条件	使策略生效的特定条件，包括**条件键**和**运算符**。	格式为“条件运算符:{条件键：[条件值1,条件值2]}”。 如果您设置多个条件，同时满足所有条件时，该策略才生效。 示例: "StringEndWithIfExists":{"g:UserName":["specialCharacter"]}：表示当用户输入的用户名以"specialCharacter"结尾时该条statement生效。
	Resource: 资源类型	策略所作用的资源。	格式为“服务名:<region>:<account-id>:资源类型:资源路径”，资源类型支持通配符号*，通配符号*表示所有。 **说明** ModelArts的授权不支持指定具体资源路径。

ModelArts 资源类型

管理员可以按ModelArts的资源类型选择授权范围。ModelArts支持的资源类型如下表：

表 11-5 ModelArts 资源类型（角色与策略授权）

资源类型	说明
notebook	开发环境的Notebook实例
exemlProject	自动学习项目
exemlProjectInf	自动学习项目的在线推理服务
exemlProjectTrain	自动学习项目的训练作业

资源类型	说明
exemlProjectVersion	自动学习项目的版本
workflow	Workflow项目
pool	专属资源池
network	专属资源池网络连接
trainJob	训练作业
trainJobLog	训练作业的运行日志
trainJobInnerModel	系统预置模型
model	模型
service	在线服务
nodeservice	边缘服务
workspace	工作空间
dataset	数据集
dataAnnotation	数据集的标注信息
aiAlgorithm	训练算法
image	镜像
devserver	弹性裸金属

ModelArts 资源权限项

参考《 ModelArts API参考 》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**
- **工作空间管理权限**

11.2.2 依赖和委托

功能依赖

功能依赖策略项

您在使用ModelArts的过程中，需要和其他云服务交互，比如需要在提交训练作业时选择指定数据集OBS路径和日志存储OBS路径。因此管理员在为用户配置细粒度授权策略时，需要同时配置依赖的权限项，用户才能使用完整的功能。

说明

- 如果您使用根用户（与账户同名的缺省子用户）使用ModelArts，根用户默认拥有所有权限，不再需要单独授权。
- 请用户确保当前用户具备委托授权中包含的依赖策略项权限。例如，用户给ModelArts的委托需要授权SWR Admin权限，需要保证用户本身具备SWR Admin权限。

表 11-6 基本配置

业务场景	依赖的服务	依赖策略项	支持的功能
全局配置	IAM	iam:users:listUsers	查询用户列表（仅管理员需要）
基本功能	IAM	iam:tokens:assume	使用委托获取用户临时认证凭据（必须）
基本功能	BSS	bss:balance:view	在ModelArts控制台创建资源后，页面展示账号当前余额

表 11-7 管理工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
工作空间	IAM	iam:users:listUsers	按用户进行工作空间授权
	ModelArts	modelarts:*:delete*	删除工作空间时，同时清理空间内的资源

表 11-8 管理开发环境 Notebook

业务场景	依赖的服务	依赖策略项	支持的功能
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get	实例的启动、停止、创建、删除、更新等依赖的权限。
	AOM	aom:metric:get aom:metric:list aom:alarm:list	
	VPC	vpc:securityGroups:get vpc:vpcs:list	
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list kps:domainKeypairs:createkmskey	
	KMS	kms:cmk:list	
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子用户对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子用户创建的。
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。
	IAM	iam:users:listUsers	

业务场景	依赖的服务	依赖策略项	支持的功能
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。

表 11-9 弹性节点 Server

业务场景	依赖的服务	依赖策略项	支持的功能
弹性节点Server实例生命周期管理	ModelArts	modelarts:devserver:create modelarts:devserver:listByUser modelarts:devserver:list modelarts:devserver:get modelarts:devserver:delete modelarts:devserver:start modelarts:devserver:stop modelarts:devserver:sync	创建实例、查询实例列表、查询租户所有实例列表、查询实例详情、删除实例、启动实例、停止实例、同步实例状态。
	ECS	ecs:serverKeypairs:createecs:*:get	
	IAM	iam:users:getUser iam:users:listUsers iam:projects:listProjects	
	VPC	vpc.*.list	
	EPS	eps.*.list	
	EVS	evs.*.list	
	IMS	ims.*.list ims.*.get	

表 11-10 管理训练作业

业务场景	依赖的服务	依赖策略项	支持的功能
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。

业务场景	依赖的服务	依赖策略项	支持的功能
		modelarts:workspace:getQuota	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。
	SWR	SWR Admin	使用自定义镜像运行训练作业。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。

表 11-11 使用 Workflow

业务场景	依赖的服务	依赖策略项	支持的功能
使用数据集	ModelArts	modelarts:dataset:getDataset modelarts:dataset:createDataset modelarts:dataset:createDatasetVersion modelarts:dataset:createImportTask modelarts:dataset:updateDataset modelarts:processTask:createProcessTask modelarts:processTask:getProcessTask modelarts:dataset:listDatasets	在工作流中使用ModelArts数据集
管理模型	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	在工作流中管理ModelArts模型
部署上线	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	在工作流中管理ModelArts在线服务
训练作业	ModelArts	modelarts:trainJob:get modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJobVersion:list modelarts:trainJobVersion:create modelarts:trainJob:delete modelarts:trainJobVersion:delete modelarts:trainJobVersion:stop	在工作流中管理ModelArts训练作业
工作空间	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	在工作流中使用ModelArts工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	在工作流中使用OBS数据
工作流运行	IAM	iam:users:listUsers（查询用户列表） iam:agencies:getAgency（查询指定委托详情） iam:tokens:assume（获取委托Token）	在工作流运行时，调用ModelArts其他服务
集成DLI	DLI	dli:jobs:get（查询作业详情） dli:jobs:listAll（查询作业列表） dli:jobs:create（创建新作业）	在工作流中集成DLI

业务场景	依赖的服务	依赖策略项	支持的功能
集成MRS	MRS	mrs:job:get（查询作业详情） mrs:job:submit（创建并执行作业） mrs:job:list（查询作业列表） mrs:job:stop（停止作业） mrs:job:batchDelete（批量删除作业） mrs:file:list（查询文件列表）	在工作流中集成MRS

表 11-12 管理模型

业务场景	依赖的服务	依赖策略项	支持的功能
管理模型	SWR	SWR Admin	从自定义镜像导入、从OBS导入时使用自定义引擎。 SWR共享版不支持细粒度权限项，因此需要配置Admin权限。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	从OBS导入模型 模型转换指定OBS路径

表 11-13 管理部署上线

业务场景	依赖的服务	依赖策略项	支持的功能
在线服务	LTS	lts:logs:list（查询日志列表）	查询和展示LTS日志。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:GetBucketPolicy （获取桶策略） obs:bucket:HeadBucket （获取桶元数据） obs:bucket:ListAllMyBuckets （获取桶列表） obs:bucket:PutBucketPolicy （设置桶策略） obs:bucket:DeleteBucketPolicy （删除桶策略）	服务运行时容器挂载外部存储卷。
批量服务	OBS	obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:bucket:CreateBucket（创建桶） obs:bucket:ListBucket（列举桶内对象） obs:bucket:ListAllMyBuckets（获取桶列表）	创建批量服务，批量推理。
边缘服务	CES	ces:metricData:list（查询指标数据）	查看服务的监控指标
	IEF	ief:deployment:delete（删除应用部署）	管理边缘服务
AOM指标告警事件	AOM	aom:alarm:list	查看AOM监控相关信息。

表 11-14 管理数据集

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据集和标注	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUploadParts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	管理OBS中的数据集 标注OBS数据 创建数据管理作业
管理表格数据集	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describeTable	在数据集中管理DLI数据
管理表格数据集	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	在数据集中管理DWS数据
管理表格数据集	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	在数据集中管理MRS数据
智能标注	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	使用智能标注

业务场景	依赖的服务	依赖策略项	支持的功能
团队标注	IAM	iam:projects:listProjects（查询租户项目） iam:users:listUsers（查询用户列表） iam:agencies:createAgency（创建委托） iam:quotas:listQuotasForProject（查询指定项目的配额）	管理标注团队

表 11-15 资源管理

业务场景	依赖的服务	依赖策略项	支持的功能
资源池管理	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	资源池的创建、续费、退订等与计费相关的功能。
	CCE	cce:cluster:list cce:cluster:get	获取CCE集群列表、集群详情、集群证书等信息。
	KMS	kms:cmk:list kms:cmk:getMaterial	获取用户创建的密钥对列表信息。
	AOM	aom:metric:get	获取资源池的监控数据。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion	获取AI诊断日志。
	ECS	ecs:availabilityZones:list ecs:cloudServerFlavors:get ecs:cloudServerQuotas:get ecs:quotas:get ecs:serverKeypairs:list	查询可用区列表、规格、配额，配置密钥匙。
	EVS	evs:types:get evs:quotas:get	查询云硬盘类型列表、配额。
	BMS	bms:serverFlavors:get	查询裸金属规格。依赖权限需要配置在IAM项目视图中。
	DEW	kps:domainKeypairs:list	配置密钥对。依赖权限需要配置在IAM项目视图中。

业务场景	依赖的服务	依赖策略项	支持的功能
网络管理	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete vpc:securityGroups:get	ModelArts网络资源创建和删除、VPC网络打通。
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	用户的网络和SFS Turbo资源打通。

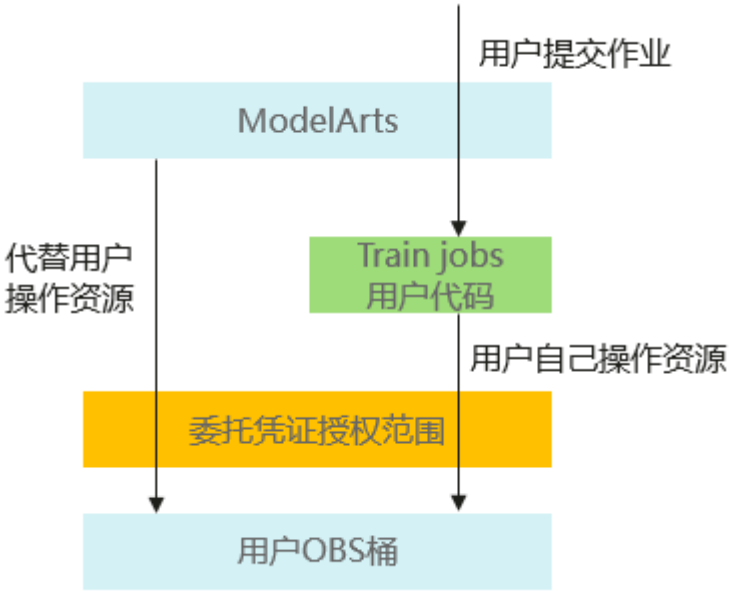
业务场景	依赖的服务	依赖策略项	支持的功能
边缘资源池	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	边缘池增删改查管理。

委托授权

用户在使用ModelArts服务的过程中，为了简化用户的操作，ModelArts后台可以代替用户完成一些工作，如训练作业启动前自动下载用户OBS桶中的数据集到作业空间、自动转储训练作业日志到用户OBS桶中。

ModelArts服务不会保存用户的Token认证凭据，在后台异步作业中操作用户的资源（如OBS桶）前，需要用户通过IAM委托向ModelArts显式授权，ModelArts在需要时使用用户的委托获取临时认证凭据用于操作用户资源，见“**添加授权**”。

图 11-4 委托授权



如图11-4所示，用户向ModelArts授权后，ModelArts使用委托授权的临时凭证访问和操作用户资源，协助用户自动化一些繁琐和耗时的操作。同时，委托凭证会同步到用户的作业中（Notebook实例和训练作业），用户在作业中可以使用委托凭证自行访问自己的资源。

在ModelArts服务中委托授权有两种方式：

1、一键式委托授权

ModelArts提供了一键式自动授权功能，用户可以在ModelArts的权限管理功能中，快速完成委托授权，由ModelArts为用户自动创建委托并配置到ModelArts服务中。

这种方式为保证使用业务过程中有足够的权限，基于依赖服务的预置系统策略指定授权范围，创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果您需要对委托授权的权限范围进行精确控制，请使用第二种方式。

2、定制化委托授权

管理员在IAM中为不同用户创建不同的委托授权策略，再到ModelArts中为用户配置已创建好的委托。管理员在IAM中为用户创建委托时，根据用户的实际权限范围为委托指定最小权限范围，控制用户在使用ModelArts过程中可访问的资源内容。具体参考**配置ModelArts基本使用权限**。

委托授权的越权风险

可以看到用户的委托授权是独立的，理论上用户的委托授权范围是可以超出用户自身用户组的授权策略的授权范围，如果配置不当就会出现用户越权的问题。

为了控制委托授权的越权风险，ModelArts服务的权限管理功能要求只有租户管理员才能为用户配置委托，由管理员保证委托授权的安全性。

委托授权的最小化

管理员在配置委托授权时，应严格控制授权的范围。

ModelArts为用户异步自动化完成作业的准备、清理等操作，所需的委托授权内容是基础授权范围。如果用户只使用ModelArts的部分功能，管理员可以依据委托授权表格的说明屏蔽不使用的基础权限项。相反地，如果用户需要在作业中使用基础授权范围外的资源权限，管理员也可以为用户在委托授权中增加新的权限项。总之，委托授权的范围应该基于实际业务场景所需权限范围来进行定制，保持委托授权范围的最小化。

委托基础授权范围

当您需要定制委托授权的权限列表时，请参考下面表格，根据实际业务选择授权项。

表 11-16 开发环境基础委托授权

业务场景	依赖的服务	委托授权项	说明
Notebook实例中操作OBS数据。	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	您可通过以下方式在Notebook实例中操作OBS中的数据： • 通过ModelArts SDK操作OBS数据。 • 通过Notebook文件上传功能操作OBS数据。 • 通过在Console页面添加OBS桶到Notebook实例的/data目录下，以文件方式操作OBS数据。
Notebook实例事件上报。	AOM	aom:alarm:put	在Notebook实例的生命周期中，部分事件会上报到用户AOM账号下，事件列表详见**Notebook实例事件**。
VPC与Notebook实例网络互联。	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnets:get	Notebook实例中新增一个可以与用户指定VPC的子网的网卡，用于与用户VPC下的服务进行网络互联。
VS Code一键连接Notebook。	ModelArts	modelarts:notebook:get	用于管理Notebook实例信息，单击VS Code插件时，获取实例详情信息，以方便将SSH配置信息写入本地VS Code。
停止Notebook实例。	ModelArts	modelarts:notebook:stop	用于停止运行中的Notebook实例。
更新Notebook实例自动停止时间。	ModelArts	modelarts:notebook:updateStopPolicy	用于更新Notebook实例的自动停止时间。

业务场景	依赖的服务	委托授权项	说明
OBS并行文件系统场景下使用MindInsight/TensorBoard可视化工具。	ModelArts	modelarts:notebook:umountStorage modelarts:notebook:getMountedStorage modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage	在开发环境Notebook实例中开启MindInsight/TensorBoard可视化工具，且需要访问的是OBS并行文件系统时，需要配置左侧的权限。

表 11-17 训练作业基础委托授权

业务场景	依赖的服务	委托授权项	说明
训练作业访问OBS文件。	OBS	obs:bucket:HeadBucket obs:bucket:GetBucketLocation obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:GetObject obs:object:GetObjectVersion obs:object:GetObjectAcl obs:object:GetObjectVersionAcl	训练作业配置代码目录、输入、输出和日志的OBS桶路径时，需要OBS服务相关操作权限，用于OBS对象路径的合法性校验。
训练作业以自定义容器镜像方式启动。	SWR	SWR Admin	训练作业以自定义容器镜像方式启动时，需要获取用户SWR容器镜像的临时登录指令，用于下载容器镜像。SWR共享版不支持细粒度权限项，因此需要配置Admin权限。
训练作业状态变化通知。	SMN	smn:template:list smn:template:create smn:topic:list smn:topic:publish	若要配置训练作业状态变化通知，需要SMN服务相关操作权限，用于发送模板化的消息通知。
训练作业配置挂载SFS Turbo。	SFS Turbo	SFS Turbo ReadOnlyAccess	训练作业配置挂载SFS Turbo时，需要SFS Turbo读权限，以通过SFS Turbo ID获取其详情。

业务场景	依赖的服务	委托授权项	说明
审计日志上报。	CTS	CTS Administrator	配置CTS服务权限，用于上报事件。CTS服务当前上报事件功能未支持细粒度权限项，因此需要配置Administrator权限（**CTS细粒度权限说明**）。

表 11-18 推理部署基础委托授权

业务场景	依赖的服务	委托授权项	说明
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	建议配置，在线服务配置LTS日志上报。
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	使用批量服务时必须配置。
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	使用边缘服务时必须配置，通过IEF部署边缘服务。
从OBS导入模型。	OBS	obs:object:DeleteObject obs:object:GetObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	必须配置。若有使用并行文件系统，则需额外配置obs:bucket:HeadBucket。
从容器镜像中导入模型。	SWR	SWR Admin	必须配置。SWR共享版不支持细粒度权限项，因此需要配置Admin权限。

业务场景	依赖的服务	委托授权项	说明
使用ModelArts Edge功能。	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	可选配置，如果使用ModelArts Edge功能需要配置。
AOM指标告警事件	AOM	aom:log:get aom:alarm:get aom:metric:put aom:alarm:put aom:event:put aom:event:list aom:event:get	建议配置，若需要AOM查看告警事件则需要配置。
监控指标上报CES	CES	ces:metricMeta:create	建议配置，监控指标上报CES。
消息订阅推送	SMN	smn:topic:list smn:topic:publish smn:application:publish	可选配置，如果使用消息订阅推送功能需要配置。

表 11-19 数据管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
使用数据标注、数据处理功能。	ModelArts	modelarts:trainJob:create modelarts:trainJob:update modelarts:trainJob:delete modelarts:trainJob:get modelarts:trainJob:list modelarts:trainJob:logExport modelarts:aiAlgorithm:get modelarts:model:get modelarts:service:list modelarts:model:create modelarts:workspace:list modelarts:workspace:get modelarts:trainJobInnerModel:list	必须配置，使用数据标注、数据处理功能时会进行创建训练作业、查询训练作业、算法查询等操作。
访问OBS数据	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUploadParts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	必须配置，在OBS中存储、查询、删除数据。

业务场景	依赖的服务	委托授权项	说明
访问DLI数据。	DLI	dli:queue:createQueue dli:queue:dropQueue dli:queue:scaleQueue dli:queue:submitJob dli:database:displayDatabase dli:database:displayAllTables dli:table:describeTable dli:table:showPrivileges dli:table:dropTable	可选配置，如果访问DLI数据需要配置。
访问MRS数据。	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:file:list	可选配置，如果访问MRS数据需要配置。
访问DWS数据。	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	可选配置，如果访问DWS数据需要配置。

表 11-20 专属资源池管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
通过关联sfsturbo功能实现专属资源池和SFS Turbo资源打通。	SFS Turbo	sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic	使用该特性时需要配置。
用户的ModelArts网络和VPC进行打通，同时添加相关路由。	VPC	vpc:vpcs:get vpc:subnets:get vpc:peerings:accept vpc:routes:create vpc:routes:delete vpc:routes:get vpc:routeTables:update vpc:routeTables:get	使用该特性时需要配置。

业务场景	依赖的服务	委托授权项	说明
使用ModelArts Lite Cluster资源池。	CCE APM	cce:cluster:get cce:node:get cce:node:list cce:job:get cce:node:create cce:node:delete cce:node:remove cce:addonInstance:get cce:addonInstance:list cce:addonInstance:create cce:addonInstance:update cce:addonInstance:delete apm:icmgr:create	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式管理用户的CCE集群，同步集群信息、纳管节点等。
	ECS BMS EVS DEW	ecs:cloudServers:create ecs:cloudServers:delete ecs:cloudServers:get ecs:cloudServers:start ecs:cloudServers:stop ecs:cloudServers:reboot ecs:cloudServers:redeploy ecs:cloudServers:listServerInterfaces ecs:cloudServers:changeVpc ecs:cloudServerFlavors:get ecs:quotas:get ecs:cloudServers:batchSetServerTags ecs:cloudServers:list bms:servers:create bms:serverFlavors:get evs:types:get evs:volumes:list evs:quotas:get evs:volumes:get kps:domainKeypairs:get	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式对用户的BMS/ECS节点进行生命周期的管理。

业务场景	依赖的服务	委托授权项	说明
	IMS	ims:images:get ims:images:share	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts Lite Cluster专属资源池节点创建在用户账号下，创建前需要将节点系统镜像共享给用户账号。

表 11-21 Workflow 基础委托授权

业务场景	依赖的服务	委托授权项	说明
对ModelArts数据管理、训练、推理等服务操作。	ModelArts	modelarts:workspace:getQuotas modelarts:service:get modelarts:app:get modelarts:pool:get modelarts:model:get modelarts:trainJob:get modelarts:dataset:get modelarts:dataAnnotation:get modelarts:workspace:get modelarts:aiAlgorithm:get modelarts:trainJobVersion:delete modelarts:dataset:delete modelarts:workspace:delete modelarts:trainJobVersion:create modelarts:dataset:import modelarts:trainJob:delete modelarts:service:delete modelarts:dataset:publishVersion modelarts:app:create modelarts:service:create modelarts:service:update modelarts:aiAlgorithm:create modelarts:trainJobVersion:stop modelarts:workspace:update modelarts:dataset:deleteVersion modelarts:trainJob:create modelarts:model:delete modelarts:model:create modelarts:dataset:create modelarts:app:delete modelarts:aiAlgorithm:delete modelarts:service:action modelarts:app:update modelarts:trainJobVersion:list modelarts:model:list modelarts:app:list modelarts:service:list	用于在Workflow中调用MA的数据管理、训练、推理等服务，创建相应的实例。 建议在配置委托权限时，直接配置ModelArts CommonOperations权限即可。

业务场景	依赖的服务	委托授权项	说明
		modelarts:dataset:list modelarts:workspace:list modelarts:trainJob:list modelarts:aiAlgorithm:list	
Workflow中对OBS数据操作。	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:PutObjectVersionAcl obs:object:PutObjectAcl obs:object:GetObjectAcl obs:object:DeleteObject obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketPolicy obs:bucket:DeleteBucketPolicy obs:bucket:GetBucketPolicy	可选配置，在工作流中使用OBS数据时需要配置。
Workflow中操作DLI相关的任务。	DLI	dli:jobs:create dli:jobs:delete dli:jobs:get dli:jobs:listAll dli:jobs:stop	可选配置，如果在Workflow中涉及DLI相关节点需要配置。
Workflow中操作MRS相关的任务	MRS	mrs:job:get mrs:cluster:get mrs:file:list mrs:cluster:list mrs:job:stop mrs:job:submit mrs:job:delete	可选配置，如果在Workflow中涉及MRS相关节点需要配置。
Workflow中使用SMN消息订阅功能。	SMN	smn:topic:list smn:topic:publish	可选配置，使用SMN消息订阅功能时必须配置。

11.2.3 工作空间

ModelArts的用户需要为不同的业务目标开发算法、管理和部署模型，此时可以创建多个工作空间，把不同应用开发过程的输出内容划分到不同工作空间中，便于管理和使用。

工作空间支持3种访问控制：

- PUBLIC：租户（主账号和所有子账号）内部公开访问。
- PRIVATE：仅创建者和主账号可访问。
- INTERNAL：创建者、主账号、指定IAM子账号可访问当授权类型为INTERNAL时需要指定可访问的子账号的账号名，可选择多个。

每个账号每个IAM项目都会分配1个默认工作空间，默认工作空间的访问控制为PUBLIC。

通过工作空间的访问控制能力，可限制仅允许部分人访问对应的工作空间。通过此功能可实现类似如下场景：

- **教育场景**：老师可给每个学生分配1个INTERNAL的工作空间并且限制该工作空间被指定学生访问，这样可使得学生可独立完成在ModelArts上的实验。
- **企业场景**：管理者可创建用于生产任务的工作空间并限制仅让运维人员使用，用于日常调试的工作空间并限制仅让开发人员使用。通过这种方式让不同的企业角色只能在指定工作空间下使用资源。

目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持申请开通。

11.3 典型场景配置实践

11.3.1 个人用户快速配置 ModelArts 访问权限

ModelArts使用过程中涉及到OBS、SWR等服务交互，需要用户配置委托授权，允许ModelArts访问这些依赖服务。如果没有授权，ModelArts的部分功能将不能正常使用。

约束与限制

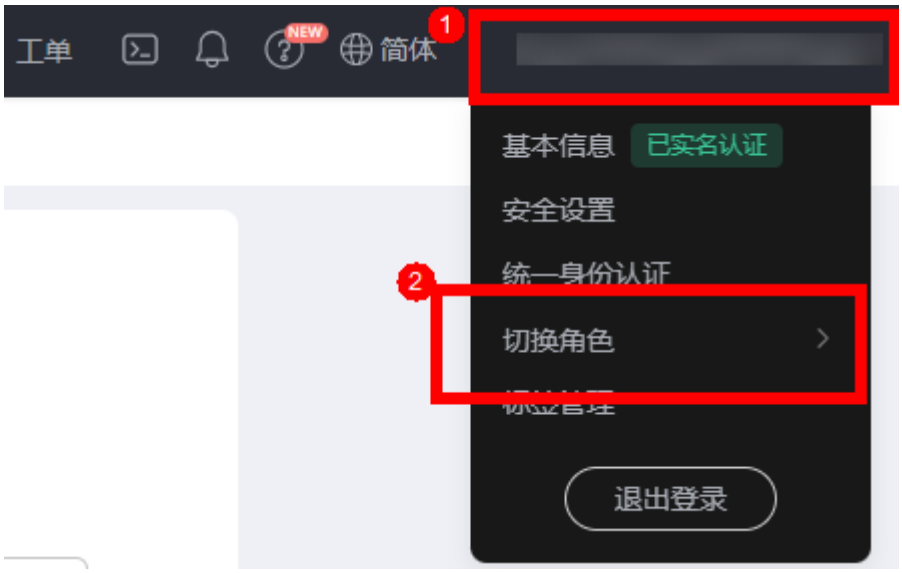
- 只有主账号可以使用委托授权，可以为当前账号授权，也可以为当前账号下的所有IAM用户授权。
- 多个IAM用户或账号，可使用同一个委托。
- 一个账号下，最多可创建50个委托。
- 对于首次使用ModelArts新用户，请直接新增委托即可。一般用户新增普通用户权限即可满足使用要求。如果有精细化权限管理的需求，可以自定义权限按需设置。
- 如果未获得委托授权，当打开“访问授权”页面时，ModelArts会提醒您当前用户未配置授权，需联系此IAM用户的管理员账号进行委托授权。

添加授权

1. 登录ModelArts管理控制台，在左侧导航栏选择“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。

表 11-22 参数说明

参数	说明
“授权对象类型”	包括IAM子用户、联邦用户、委托用户和所有用户。 • IAM子用户：由主账号在IAM中创建的用户，是服务的使用人员，具有独立的身份凭证（密码和访问密钥），根据账号授予的权限使用资源。IAM子用户相关介绍请参见**IAM用户介绍**。 • 联邦用户：又称企业虚拟用户。联邦用户相关介绍请参见**联邦身份认证**。 • 委托用户：IAM中创建的一个委托。IAM创建委托相关介绍请参见**创建委托**。 • 所有用户：该选项表示会将委托的权限授权到当前账号下的所有子账号、包括未来创建的子账号，授权范围较大，需谨慎使用。个人用户选择“所有用户”即可。

参数	说明
“授权对象”	“授权对象类型”选择“所有用户”时不涉及此参数。 • IAM子用户：选择指定的IAM子用户，给指定的IAM子用户配置委托授权。 **图 11-5** 选择 IAM 子用户  • 联邦用户：输入联邦用户的用户名或用户ID。 **图 11-6** 选择联邦用户  • 委托用户：选择委托名称。使用账号A创建一个权限委托，在此处将该委托授权给账号B拥有的委托。在使用账号B登录控制台时，可以在控制台右上角的个人账号切换角色到账号A，使用账号A的委托权限。 **图 11-7** 委托用户切换角色  **说明** ModelArts暂不支持创建身份策略权限的委托。
“委托选择”	• 已有委托：列表中如果已有委托选项，则直接选择一个可用的委托为上述选择的用户授权。单击委托名称查看该委托的权限详情。 • 新增委托：如果没有委托可选，可以在新增委托中创建委托权限。对于首次使用ModelArts的用户，需要新增委托。

参数	说明
“新增委托 > 委托名称”	系统自动创建委托名称，用户可以手动修改。
“新增委托 > 权限配置 > 普通用户”	普通用户包括用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。 可以单击“查看权限列表”，查看普通用户权限。
“新增委托 > 权限配置 > 自定义”	如用户有精细化权限管理的需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需要在权限列表中勾选要配置的权限。

3. 然后勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，即可完成委托配置。

查看授权的权限列表

用户可以在“权限管理”页面的授权列表中，查看已经配置的委托授权内容。单击授权内容列的“查看权限”，可以查看该授权的权限详情。

图 11-8 查看权限

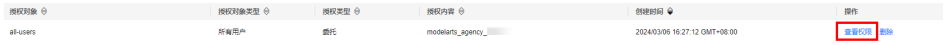
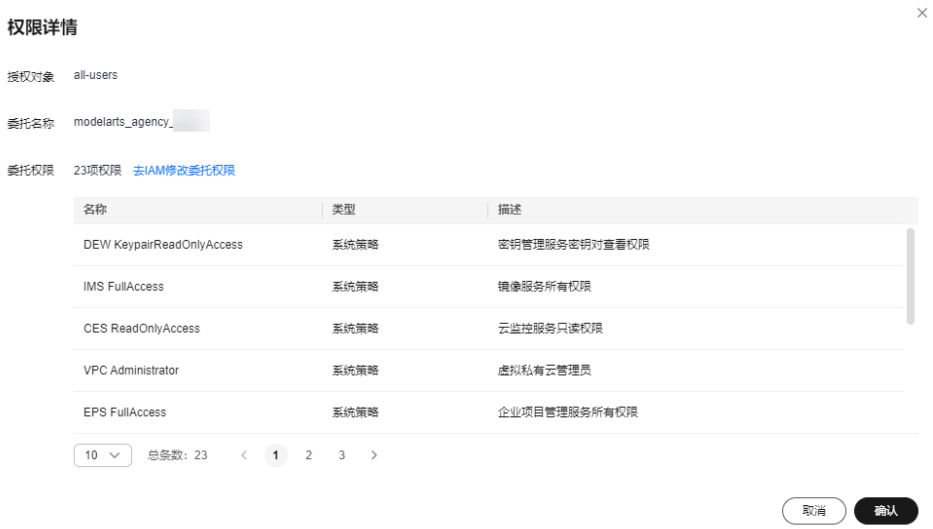


图 11-9 普通用户权限列表



11.3.2 配置 ModelArts 基本使用权限

11.3.2.1 场景描述

ModelArts作为顶层服务，其部分功能依赖于其他服务的访问权限。本章节主要介绍对于IAM子账号使用ModelArts时，如何根据需要开通的功能配置子账号相应权限。

权限列表

子账号的权限，由主用户来控制，主用户通过IAM的权限配置功能设置用户组的权限，从而控制用户组内的子账号的权限。此处的授权列表均按照ModelArts和其他服务的系统预置策略来举例。

表 11-23 服务授权列表

待授权的服务	授权说明	IAM权限设置	是否必选
ModelArts	授予子账号使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子账号配置此权限。	ModelArts CommonOperations	必选
	如果需要给子账号开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。	ModelArts FullAccess	可选 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
OBS对象存储服务	授予子账号使用OBS服务的权限。ModelArts的数据管理、开发环境、训练作业、模型推理部署均需要**通过OBS进行数据中转**。	OBS OperateAccess	必选
SWR容器镜像仓库	授予子账号使用SWR服务权限。ModelArts的**自定义镜像功能**依赖镜像服务SWR FullAccess权限。	SWR OperateAccess	必选
密钥管理服务	当子账号使用ModelArts **Notebook的SSH远程功能**时，需要配置子账号密钥管理服务的使用权限。	KMS CMKFullAccess	可选
IEF智能边缘平台	授予子账号智能边缘平台使用权限，ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。	Tenant Administrator	可选

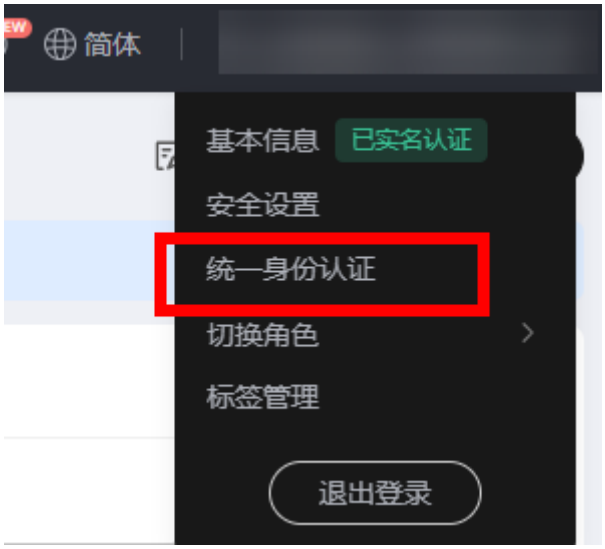
待授权的服务	授权说明	IAM权限设置	是否必选
CES云监控	授予子账号使用CES云监控服务的权限。通过CES云监控可以查看ModelArts的在线服务和对应模型负载运行状态的整体情况，并设置监控告警。	CES FullAccess	可选
SMN消息服务	授予子账号使用SMN消息服务的权限。SMN消息通知服务配合CES监控告警功能一起使用。	SMN FullAccess	可选
VPC虚拟私有云	子账号在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。	VPC FullAccess	可选
SFS弹性文件服务	授予子账号使用SFS服务的权限，ModelArts的专属资源池中可以挂载SFS系统作为开发环境或训练的存储。	SFS Turbo FullAccess SFS FullAccess	可选

11.3.2.2 Step1 创建用户组并加入用户

主用户账号下面可以创建多个子账号，并对子账号的权限进行分组管理。此步骤介绍如何创建用户组、子账号、并将子账号加入用户组中。

1. 主用户登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

   图 11-10 统一身份认证

2. 创建用户组。在左侧菜单栏中，选择“用户组”。单击右上角“创建用户组”，在“用户组名称”中填入“用户组02”，然后单击“确定”完成用户组创建。

创建完成后，返回用户组列表。通过用户组管理，将已有子账号加入到用户组中。如果没有子用户账号，可以创建子账号并加入用户组。

3. 创建子用户账号并加入用户组。在IAM左侧菜单栏中，选择“用户”，单击右上角“创建用户”，在“创建用户”页面中，添加多个用户。

   请根据界面提示，填写必选参数，然后单击“下一步”。

4. 在“加入用户组”步骤中，选择“用户组02”，然后单击“创建用户”。

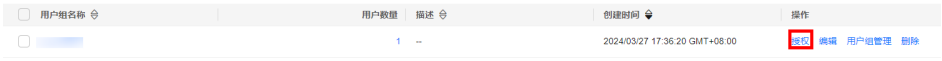
   系统将逐步创建好前面设置的2个用户。

11.3.2.3 Step2 为用户配置云服务使用权限

主用户为子账号授予ModelArts、OBS等云服务的使用权限后，子账号才可以使用这些云服务。此步骤介绍如何为用户组中的所有子账号授予使用ModelArts、OBS、SWR等各类云服务的权限。

1. 主用户在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。

   图 11-11 为用户组授权

2. 配置授权前，请先了解ModelArts各模块使用到的最小权限要求，如**表11-23**所示。
3. 配置ModelArts使用权限。在搜索框搜索ModelArts。ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。

   选择说明如下：

   - ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子账号配置此权限。
   - 如果需要给子账号开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。

4. 配置OBS使用权限。搜索OBS，勾选“OBS Administrator”。ModelArts训练作业中需要依赖OBS作为数据中转站，需要配置OBS的使用权限。
5. 配置SWR使用权限。搜索SWR，勾选“SWR FullAccess”。ModelArts的自定义镜像功能依赖镜像服务SWR FullAccess权限。
6. （可选）配置密钥管理权限。如果需要使用ModelArts Notebook的SSH访问功能，依赖密钥管理权限。搜索DEW，勾选“DEW KeypairFullAccess”。

   此处需要注意以下Region配置的是DEW密钥管理权限：华北-北京一、华北-北京四、华东-上海一、华东-上海二、华南-广州、西南-贵阳一、中国-香港、亚太-新加坡。其他Region配置的是KMS密钥管理权限。本示例中使用“华南-广州”Region举例，所以需要配置DEW密钥管理权限。

7. （可选）配置智能边缘平台使用权限。ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。

   注意：Tenant Administrator权限比较大，包含全部云服务的管理权限，而不仅是使用ModelArts服务。请谨慎配置。

8. （可选）配置CES云监控和SMN消息通知使用权限。ModelArts推理部署的在线服务详情页面内有调用次数详情，单击可查看该在线服务的调用次数随时间详细分布的情况。如果想进一步通过CES云监控查看ModelArts的在线服务和对应模型负载运行状态的整体情况，需要给子账号授予CES权限。

   如果只是查看监控，给子账号授予CES ReadOnlyAccess权限即可。

如果还需要在CES上设置监控告警，则需要再加上CES FullAccess权限，以及SMN消息通知权限。

9. （可选）配置VPC权限。如果用户在创建专属资源池过程中，需要开启自定义网络配置，此处需要授予用户VPC权限。
10. （可选）配置SFS和SFS Turbo权限。如果用户在专属资源池中挂载SFS系统作为开发环境或训练的存储时，需要授予使用权限。
11. 单击左上角的“查看已选”，确认已勾选的权限。
12. 再单击“下一步”，设置最小授权范围。单击“指定区域项目资源”，勾选待授权使用的区域，单击“确定”。
13. 提示授权成功，查看授权信息，单击“完成”。此处的授权生效需要15-30分钟。

11.3.2.4 Step3 为用户配置 ModelArts 的委托访问授权

配置完IAM权限之后，需要在ModelArts页面为子账号设置ModelArts访问授权，允许ModelArts访问OBS、SWR、IEF等依赖服务。

此方式只允许主用户为子账号进行配置。因此，本示例中，管理员账号需为所有用户完成访问授权的配置。

1. 使用主用户的账号登录ModelArts服务管理控制台。请注意选择左上角的区域，例如“华南-广州”。
2. 在左侧导航栏单击“权限管理”，进入“权限管理”页面。
3. 单击“添加授权”。在“授权”页面，在“授权对象类型”下面选择“所有用户”，选择“新增委托 ”，为该主用户下面的所有子账号配置委托访问授权。
   - 普通用户：普通用户的委托权限包括了用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。
   - 自定义：如果对用户有更精细化的权限管理需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需在权限列表中勾选要配置的权限。
4. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”，完成委托授权配置。

11.3.2.5 Step4 测试用户权限

由于4中的权限需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子账号登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 验证ModelArts权限。
   a. 在左上角选择区域，区域需与授权配置中的区域相同。
   b. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，界面未提示权限不足，表明ModelArts的使用权限和委托授权配置成功。

      如果提示“需获取依赖服务的授权”，说明未配置ModelArts委托访问授权，请参考**Step3 为用户配置ModelArts的委托访问授权**，使用主用户为子账号配置ModelArts委托访问授权。
   c. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，单击“创建”，如果可以正常打开创建页面，说明具备ModelArts的操作权限。

您也可以尝试其他功能，例如“训练管理>训练作业”等，如能正常打开创建页面，即可正常使用ModelArts。

3. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
4. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
5. 依次验证其他可选权限。
6. 验证结束，当前用户同时具备ModelArts部分功能的操作权限，可正常开始使用ModelArts服务。

11.3.3 给子账号配置开发环境基本使用权限

场景描述

本文介绍开发环境场景下子账号所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子账号使用Notebook进行调试，数据和代码存放在并行文件系统。以下内容需使用管理账号进行配置。

权限清单

- 权限

表 11-24 开发环境所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get aom:metric:get aom:metric:list aom:alarm:list	实例的启动、停止、创建、删除、更新等依赖的权限。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。	按需配置。
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket		
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。	按需配置。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。	按需配置。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。	按需配置。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list kps:domainKeypairs:createkmskey		

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	KMS	kms:cmk:list		
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子账号对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子账号创建的。	按需配置。
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。	按需配置。
	IAM	iam:users:listUsers		
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData		
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。	
VPC接入	VPC	VPC ReadOnlyAccess	实例能够挂载在用户的VPC下，实现多网络平面接入。	按需配置。

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。

- 委托

表 11-25 开发环境所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
JupyterLab	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	通过ModelArts的Notebook，在JupyterLab中使用OBS上传下载数据。	建议配置。
开发环境监控功能	AOM	aom:alarm:put	调用AOM的接口，获取Notebook相关的监控数据和事件，展示在ModelArts的Notebook中。	建议配置。
VPC接入	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnets:get	实例能够挂载在用户的VPC下，实现多网络平面接入。	按需配置。

操作步骤

本案例场景为**单机单卡场景下使用Notebook进行代码调试**，数据和代码存储在OBS服务的并行文件系统下，调试完成过后可保存镜像。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加开发环境使用权限和依赖服务SWR权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

1. 添加开发环境使用权限。
   - “策略名称”：设置自定义策略名称，例如：notebook。
   - “策略配置方式”：选择JSON视图。

- “策略内容”：填入如下内容。

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "modelarts:notebook:create",
            "modelarts:notebook:list",
            "modelarts:notebook:get",
            "modelarts:notebook:update",
            "modelarts:notebook:delete",
            "modelarts:notebook:start",
            "modelarts:notebook:stop",
            "modelarts:notebook:updateStopPolicy",
            "modelarts:notebook:listMountedStorages",
            "modelarts:notebook:mountStorage",
            "modelarts:notebook:getMountedStorage",
            "modelarts:notebook:umountStorage",
            "modelarts:image:delete",
            "modelarts:image:list",
            "modelarts:image:create",
            "modelarts:image:get",
            "modelarts:pool:list",
            "modelarts:tag:list",
            "modelarts:network:get",
            "aom:metric:get",
            "aom:metric:list",
            "aom:alarm:list"
         ]
      }
   ]
}
```

步骤3 添加依赖服务OBS权限。

在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：notebook-obs。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "obs:bucket:ListAllMyBuckets",
            "obs:bucket:ListBucket"
         ]
      }
   ]
}
```

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“trainJob”为项目级云服务、“trainJobobs”为全局级云服务。**了解更多**

步骤4 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤5 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。勾选“notebook”、“notebook-obs”、“SWR Admin”策略。单击“下一步”和“确定”。

图 11-12 给用户组授权策略

步骤6 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：ma_agency_obs。
   - “策略配置方式”：JSON视图。
   - “策略内容”：填入如下内容。

   ```
   {
      "Version": "1.1",
      "Statement": [
         {
            "Effect": "Allow",
            "Action": [
               "obs:object:GetObject",
               "obs:object:DeleteObject",
               "obs:bucket:PutBucketAcl",
               "obs:object:PutObject",
               "obs:bucket:CreateBucket",
               "obs:bucket:GetBucketAcl",
               "obs:bucket:PutBucketCORS",
               "obs:bucket:ListAllMyBuckets",
               "obs:bucket:ListBucket",
               "obs:object:GetObjectVersion"
            ]
         }
      ]
   }
   ```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_notebook。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子账号配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子账号，在已有委托中选择新建的委托，然后单击“创建”。

步骤7 验证权限是否配置成功。

登录子账号，如果用户能在控制台上成功**创建Notebook实例**、**挂载OBS文件系统**（OBS桶需由管理员创建）、**保存镜像**，则表示权限配置成功。

----结束

11.3.4 给子账号配置训练作业基本使用权限

场景描述

本文介绍训练作业场景下子账号所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子账号使用自定义镜像训练，数据和代码存放在OBS桶中。以下内容需使用管理账号进行配置。

权限清单

- 权限

表 11-26 训练作业所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
		modelarts:workspace:getQuotas	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。	按需配置。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。	按需配置。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。	按需配置。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	SWR	swr:repository:listTags swr:repository:getRepository swr:repository:listRepositories 若为企业SWR用户，还需要增加以下权限： swr:repository:getTag swr:instance:createTempCredential swr:repository:listTags swr:repository:getRepository swr:repository:listRepositories	使用自定义镜像运行训练作业。	按需配置。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。	按需配置。

- 委托

表 11-27 训练作业所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
训练作业	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	训练作业启动前下载数据、模型、代码。 训练作业运行中上传日志、模型。	建议配置。

操作步骤

本案例场景为**单机单卡场景下创建训练作业**，数据和代码存储在OBS服务的并行文件系统下，创建自定义镜像训练作业。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加训练作业使用权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：trainJob。
- “策略配置方式”：选择JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:trainJob:*",
        "modelarts:trainJobLog:*",
        "modelarts:aiAlgorithm:*",
        "modelarts:image:list",
        "modelarts:pool:list",
        "swr:repository:listTags",
        "swr:repository:getRepository",
        "swr:repository:listRepositories"
      ]
    }
  ]
}
```

步骤3 添加依赖服务OBS权限。

在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：trainJob-obs。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "obs:object:GetObject",
        "obs:object:DeleteObjectVersion",
        "obs:bucket:GetBucketLocation",
        "obs:object:AbortMultipartUpload",
        "obs:object:PutObjectAcl",
        "obs:object:DeleteObject",
        "obs:bucket:HeadBucket",
        "obs:bucket:PutBucketAcl",
        "obs:object:PutObject",
        "obs:object:GetObjectVersionAcl",
        "obs:bucket:ListAllMyBuckets",
        "obs:object:ListMultipartUploadParts",
        "obs:object:ModifyObjectMetaData",
        "obs:bucket:ListBucket",
        "obs:object:GetObjectVersion",
        "obs:object:GetObjectAcl"
      ]
    }
  ]
}
```


> **说明**
>
> 创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“trainJob”为项目级云服务、“trainJobobs”为全局级云服务。**了解更多**

步骤4 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤5 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。勾选“trainJob”和“trainJob-obs”策略。单击“下一步”和“确定”。

步骤6 为子账号添加镜像组织管理授权。

登录容器镜像服务控制台。在左侧菜单栏选择“组织管理”，单击组织名称。在“用户”页签下单击“添加授权”，在弹出的窗口中为子账号添加“编辑”权限，然后单击“确定”。

步骤7 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：ma_agency_obs。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：填入如下内容。

   ```
   {
     "Version": "1.1",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "obs:object:GetObject",
           "obs:object:PutObject",
           "obs:bucket:ListBucket"
         ]
       }
     ]
   }
   ```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_trainJob。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子账号配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子账号，在已有委托中选择新建的委托，然后单击“创建”。

步骤8 验证权限是否配置成功。

登录子用户账号，如果用户能在控制台上成功创建使用自定义镜像创建训练作业（如**单机单卡场景下创建训练作业**），则表示权限配置成功。

----结束

11.3.5 给子账号配置部署上线基本使用权限

场景描述

本文介绍部署上线场景下子账号所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子账号权限，使其能够在开发环境Notebook中使用基础镜像构建一个新的推理镜像，并完成模型的创建，部署为在线服务。

权限清单

- 权限

表 11-28 管理模型所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
管理模型	ModelArts	modelarts:model:*	创建、删除、查看、导入AI模型。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 从自定义镜像导入。 • 从OBS导入时使用自定义引擎。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl	从OBS导入模型。 模型转换指定OBS路径。	按需配置。

表 11-29 部署上线所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
部署服务	ModelArts	modelarts:service:*	部署、启动、查新、更新模型服务。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
	LTS	lts:logs:list	查询和展示LTS日志。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
批量服务	OBS	obs:object:GetObject obs:object:PutObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets	创建批量服务。	按需配置。
边缘服务	CES	ces:metricData:list	查看服务的监控指标。	按需配置。
	IEF	IEF Administrator	管理边缘服务。	按需配置。

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。

- 委托

表 11-30 部署上线所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	在线服务配置LTS日志上报。	按需配置。
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	运行批量服务。	按需配置。
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	通过IEF部署边缘服务。	按需配置。

操作步骤

本案例场景为**在开发环境中构建并调试推理镜像**，在Notebook中制作自定义镜像，然后将调试完成的镜像导入ModelArts的模型管理中，并部署上线。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加部署上线使用权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

添加部署上线使用权限。

- “策略名称”：设置自定义策略名称，例如：service。
- “策略配置方式”：选择JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:service:*",
        "modelarts:model:*",
        "modelarts:notebook:create",
        "modelarts:notebook:list",
        "modelarts:notebook:get",
        "modelarts:notebook:update",
        "modelarts:notebook:delete",
        "modelarts:notebook:start",
        "modelarts:notebook:stop",
        "modelarts:notebook:updateStopPolicy",
        "modelarts:image:delete",
        "modelarts:image:list",
        "modelarts:image:create",
        "modelarts:image:get",
        "modelarts:image:register",
        "modelarts:image:listGroup",
        "modelarts:pool:list",
        "modelarts:tag:list",
        "aom:metric:get",
        "aom:metric:list",
        "aom:alarm:list"
      ]
    }
  ]
}
```

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。勾选“service”、“SWR Admin”策略。单击“下一步”和“确定”。

步骤5 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：service_agency。
   - “策略配置方式”：JSON视图。

- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lts:groups:create",
        "lts:groups:list",
        "lts:topics:create",
        "lts:topics:delete",
        "lts:topics:list"
      ]
    }
  ]
}
```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_service。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子账号配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子账号，在已有委托中选择新建的委托，然后单击“创建”。

步骤6 验证权限是否配置成功。

登录子账号，如果用户能跑通**在开发环境中构建并调试推理镜像**的案例，在Notebook中制作自定义镜像，然后将调试完成的镜像导入ModelArts的模型中，并部署上线，则表示权限配置成功。

----结束

11.3.6 给子账号配置查看所有 Notebook 实例的权限

查找实例

Notebook页面展示了所有创建的实例。如果需要查找特定的实例，可根据筛选条件快速查找。

- 参考**给子账号配置查看所有Notebook实例的权限**后，进入“开发空间>Notebook”页面，打开“查看所有”开关，可以看到IAM项目下所有子账号创建的Notebook实例。
- 按实例名称、实例ID、实例状态、使用的镜像、实例规格、实例描述、创建时间等单个筛选或组合筛选。

给子账号配置查看所有 Notebook 实例的权限

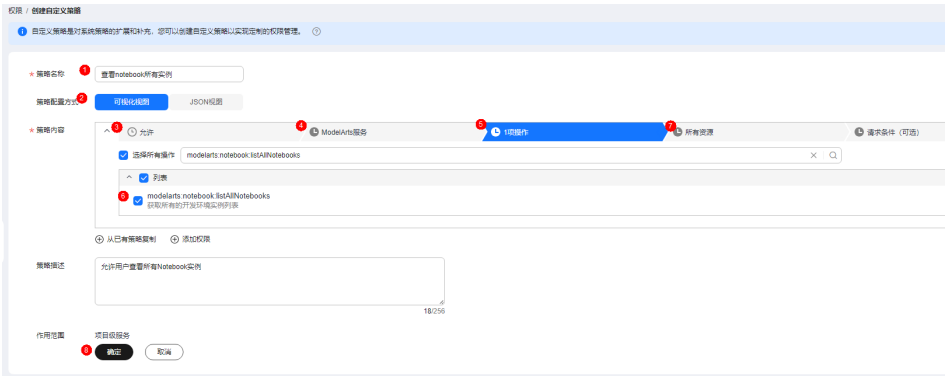
当子账号被授予“listAllNotebooks”和“listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到IAM项目下所有子账号创建的Notebook实例 。配置该权限后，也可以在Notebook中访问子账号的OBS、SWR等。

1. 使用主用户账号登录ModelArts管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，需要设置两条策略。

   策略1：设置查看Notebook所有实例，如图11-13所示，单击“确定”。

   - “策略名称”：设置自定义策略名称，例如：查看Notebook所有实例。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:notebook:listAllNotebooks并选中，所有资源选择默认值。

   图 11-13 创建自定义策略

   策略2：设置查看Notebook实例创建者信息的策略。

   - “策略名称”：设置自定义策略名称，例如：查看所有子账号信息。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索IAM服务并选中，操作列中搜索关键词iam:users:listUsers并选中，所有资源选择默认值。
3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子账号没有在用户组中，也可以通过“用户组管理”功能增加用户。

子账号启动其他用户的 SSH 实例

子账号可以看到所有用户的Notebook实例后，如果要通过SSH方式远程连接其他用户的Notebook实例，需要将SSH密钥对更新成自己的，否则会报错ModelArts.6786。更新密钥对具体操作请参见修改Notebook SSH远程连接配置。具体的错误信息提示：ModelArts.6789: 在ECS密钥对管理中找不到指定的ssh密钥对xxx，请更新密钥对并重试。

11.3.7 管理员和开发者权限分离

对于中小规模团队，管理员希望对ModelArts资源进行主导分配，全局控制，而对于普通开发者只需关注自己实例的生命周期控制。对于开发者账号，一般不会具有

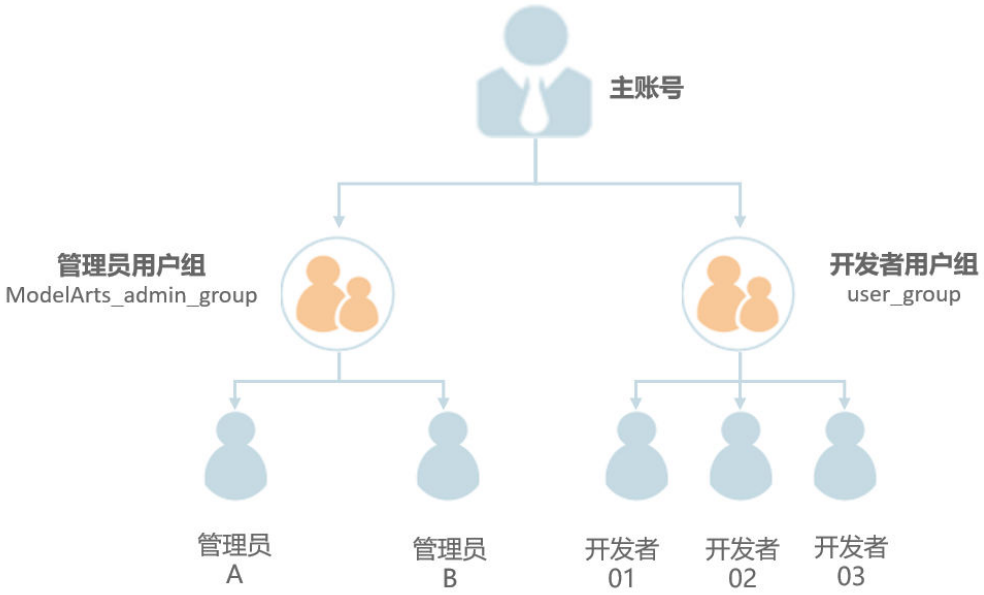
te_admin的权限，相应的权限也需要主账号进行统一配置。本章节以使用Notebook进行项目开发为例，通过自定义策略配置实现管理员和开发者分离。

场景描述

以使用Notebook进行项目开发为例，管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。

普通开发者使用开发环境，只需关注对自己Notebook实例的操作权限，包括对自己实例的创建、启动、停止、删除等权限以及周边依赖服务的权限。普通开发者不需要ModelArts专属资源池的操作权限，也不需要查看其他用户的Notebook实例。

图 11-14 账号关系示意图



配置管理员权限

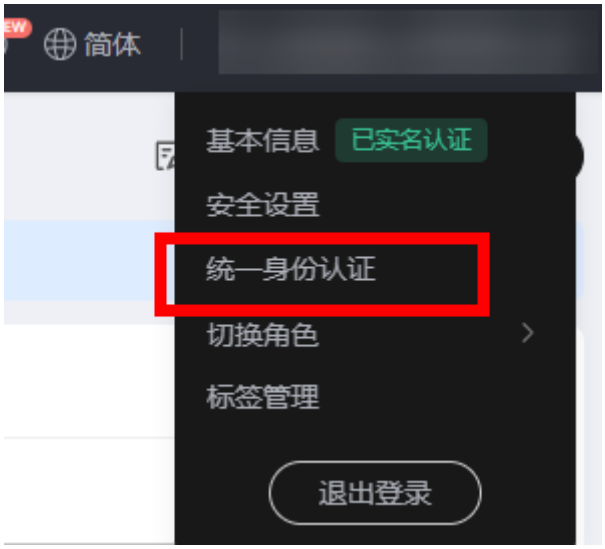
管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。可以通过以下配置流程实现管理员权限配置。

步骤1 使用主账号创建一个管理员用户组ModelArts_admin_group，将管理员账号加入用户组ModelArts_admin_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用管理员账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 11-15 登录控制台



2. 创建自定义策略1，赋予用户IAM和OBS服务权限。在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，在“策略名称”中填入“Policy1_IAM_OBS”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy1_IAM_OBS”的具体内容如下，赋予用户IAM和OBS操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:users:listUsers",
                "iam:projects:listProjects",
                "obs:object:PutObject",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:bucket:HeadBucket",
                "obs:object:DeleteObject",
                "obs:bucket:CreateBucket",
                "obs:bucket:ListBucket"
                ]
        }
    ]
}
```

3. 重复**步骤2.2**创建自定义策略2，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。“策略名称”为“Policy2_AllowOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy2_AllowOperation”的具体内容如下，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:serverKeypairs:list",
                "ecs:serverKeypairs:get",
                "ecs:serverKeypairs:delete",
                "ecs:serverKeypairs:create",
```

```
            "swr:repository:getNamespace",
            "swr:repository:listNamespaces",
            "swr:repository:deleteTag",
            "swr:repository:getRepository",
            "swr:repository:listTags",
            "swr:instance:createTempCredential",
            "mrs:cluster:get",
            "modelarts:*:*"
        ]
    }
  ]
}
```

步骤3 将步骤2创建的自定义策略授权给管理员用户组ModelArts_admin_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称ModelArts_admin_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”和“Policy2_AllowOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给管理员用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择管理员的账号，选择“新增委托”，“权限配置”选择“普通用户”。管理员不做权限控制，此处默认使用普通用户委托即可。
3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。

步骤5 测试管理员用户权限。

1. 使用管理员用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts控制台的左侧导航栏中，选择“专属资源池”，单击创建，未提示权限不足，表明管理员用户的权限配置成功。

----结束

配置开发者权限

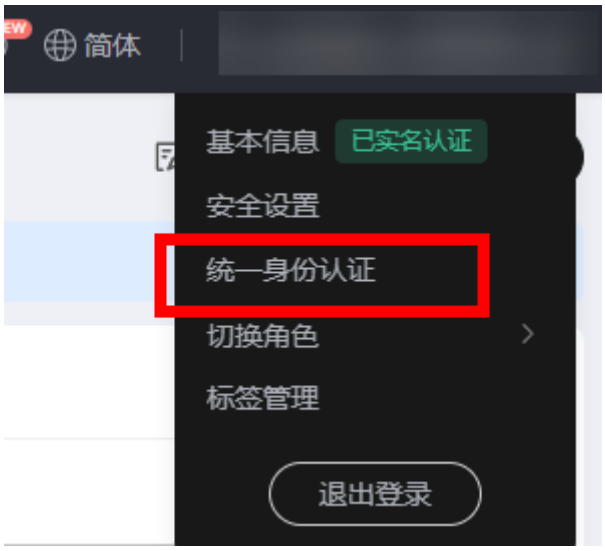
开发者权限需要通过IAM的细粒度授权控制实现，可以通过以下配置流程实现开发者权限配置。

步骤1 使用主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 11-16 登录控制台



2. 创建自定义策略3，拒绝用户操作ModelArts专属资源池并拒用户查看其他用户的Notebook。

   在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy3_DenyOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy3_DenyOperation”的具体内容如下，可以直接复制粘贴。

```
{
   "Version": "1.1",
   "Statement": [
         {
          "Effect": "deny",
          "Action": [
              "modelarts:pool:create",
              "modelarts:pool:update",
              "modelarts:pool:delete",
       "modelarts:notebook:listAllNotebooks"
            ]

         }
   ]
}
```

步骤3 将自定义策略授权给开发者用户组user_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”、“Policy2_AllowOperation”和“Policy3_DenyOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给开发者用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“新增委托”，“委托名称”设置为“ma_agency_develop_user”，“权限配置”选择“自定义”，“权限名称”勾选“OBS Administrator”。开发者用户只需要配置OBS的委托授权即可，允许开发者用户在使用Notebook时，与OBS服务交互。

3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。
4. 在“权限管理”页面，再次单击“添加授权”，进入“访问授权”页面，为其他开发者用户配置委托。

   “授权对象类型”选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“已有委托”，“委托名称”勾选上一步创建的“ma_agency_develop_user”，

步骤5 测试开发者用户权限。

1. 使用user_group用户组中任意一个子账号登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts左侧菜单栏中，选择“专属资源池”，单击创建，界面未提示权限不足，表明开发者用户的权限配置成功。

----结束

11.3.8 不允许子账号使用公共资源池创建作业

本章节介绍如何控制ModelArts用户权限，限制用户使用ModelArts公共资源池的资源创建训练作业、创建开发环境实例，部署推理服务等。

场景介绍

对于ModelArts专属资源池的用户，不允许使用公共资源池创建训练作业、创建Notebook实例或者部署推理服务时，可以通过权限控制限制用户使用公共资源池。

涉及配置的自定义权限策略项如下;

- modelarts:notebook:create：此策略项表示创建Notebook实例。
- modelarts:trainJob:create：此策略项表示创建训练作业。
- modelarts:service:create：此策略项表示创建推理服务。

给子账号配置权限：限制使用公共资源池

1. 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：不允许用户使用公共资源池创建。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：拒绝，云服务中搜索“ModelArts”服务并选中，“操作”中查找**写**操作“modelarts:trainJob:create”、“modelarts:notebook:create”和“modelarts:service:create”并选中。“所有资源”选择“默认值”。“请求条件”中单击“添加条件”，设置“条件键”为“modelarts:poolType”，“运算符”为“StringEquals”，“值”为“public”。

     JSON视图的策略内容如下：

     ```
     {
       "Version": "1.1",
       "Statement": [
     ```

```
        {
            "Effect": "Deny",
            "Action": [
                "modelarts:trainJob:create",
                "modelarts:notebook:create",
                "modelarts:service:create"
            ],
            "Condition": {
                "StringEquals": {
                    "modelarts:poolType": [
                        "public"
                    ]
                }
            }
        }
    ]
}
```

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

4. 在用户的委托授权中同步增加此策略，避免在租户面通过委托token突破限制。

   在统一身份认证服务页面的左侧导航中选择委托，找到该用户组在ModelArts上使用的委托名称，单击右侧的“修改”操作，选择“授权记录”页签，单击“授权”，选中上一步创建的自定义策略“不允许用户使用公共资源池”，单击“下一步”，选择允许使用的资源区域，单击“确定”。

验证

使用子账号用户登录ModelArts控制台，选择“模型训练 > 训练作业”，单击“创建训练作业”，在创建训练页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“开发空间 > Notebook”，单击“创建”，在创建Notebook页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“模型部署 > 在线服务”，单击“部署”，在部署服务页面，资源池规格只能选择专属资源池。

11.3.9 委托授权 ModelArts 云服务使用 SFS Turbo

本章节介绍如何配置ModelArts委托权限，允许用户使用专属资源池的网络中的“关联sfsturbo”和“解除关联”功能。

- 当用户新增委托并授权操作SFS Turbo时，请参考**新增委托授权操作SFS Turbo**。
- 当用户为已有的委托新增权限，授权操作SFS Turbo，请参考**已有委托新增授权操作SFS Turbo**。

场景介绍

对于使用ModelArts专属资源池的用户，在控制台创建完网络后，在网络列表页“操作 > 更多”下拉框中可见“关联sfsturbo”和“解除关联”。其中，“关联sfsturbo”用于将此网络与某个选定的SFS Turbo资源做关联操作，关联完成后，表示SFS Turbo与网络已进行打通，可在训练和开发环境等功能时使用此SFS Turbo。

关联与解除关联操作需要用户委托授权ModelArts云服务操作SFS Turbo的部分权限。

涉及配置的自定义权限策略项如下：

- sfsturbo:shares:addShareNic：此策略项表示sfsturbo创建网卡的权限。
- sfsturbo:shares:deleteShareNic：此策略项表示sfsturbo删除网卡的权限。
- sfsturbo:shares:showShareNic：此策略项表示sfsturbo显示网卡详情的权限。
- sfsturbo:shares:listShareNics：此策略项表示sfsturbo显示网卡列表的权限。

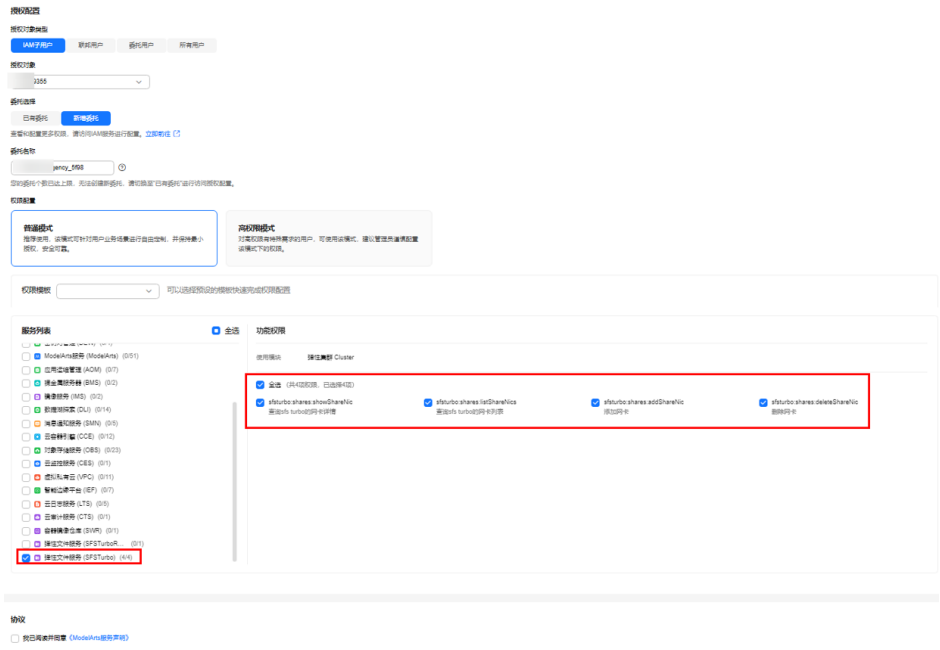
约束限制

相应region区域开放此功能。

新增委托授权操作 SFS Turbo

1. 登录ModelArts管理控制台，在左侧导航栏选择“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。
   - “授权对象类型”：根据需要选择"IAM子用户"、"联邦用户"、"委托用户"、"所有用户"
   - “授权对象”：选择授权对象
   - “委托选择”：新增委托
   - “权限配置”：普通模式，选中弹性文件服务(SFSTurbo)下的"sfsturbo:shares:addShareNic"、"sfsturbo:shares:deleteShareNic"、"sfsturbo:shares:showShareNic"、"sfsturbo:shares:listShareNics"
3. 单击“创建”。此时，拥有该委托的所有用户均有权限进行关联与解除关联操作。

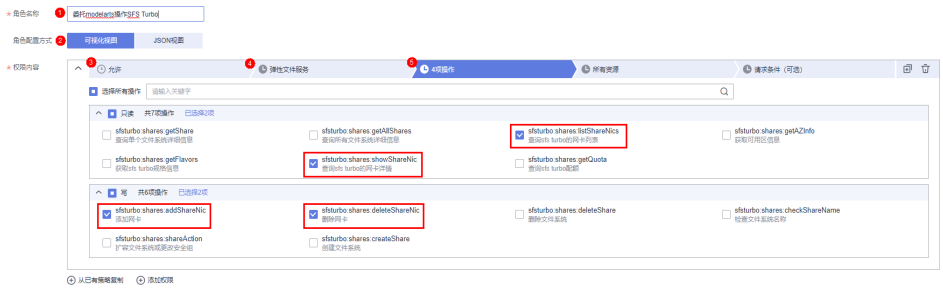
图 11-17 创建授权 ModelArts 云服务操作 SFS Turbo 的部分权限



已有委托新增授权操作 SFS Turbo

1. 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：委托modelarts操作SFS Turbo。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：允许，云服务中搜索“SFSTurbo”服务并选中，“操作”中查找**只读**操作“sfsturbo:shares:showShareNic”、“sfsturbo:shares:listShareNics”并选中，查找**写**操作“sfsturbo:shares:addShareNic”、“sfsturbo:shares:deleteShareNic”并选中。“所有资源”选择“所有资源”。

   图 11-18 创建自定义策略（可视化视图）

   JSON视图的策略内容如下：

   ```
   {
       "Version": "1.1",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "sfsturbo:shares:addShareNic",
                   "sfsturbo:shares:listShareNics",
                   "sfsturbo:shares:deleteShareNic",
                   "sfsturbo:shares:showShareNic"
               ]
           }
       ]
   }
   ```

3. 以委托“modelarts_agency”为例，在统一身份认证服务页面的左侧导航选择“委托”，在委托页面查找“modelarts_agency”，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略“委托modelarts操作SFS Turbo”，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，拥有该委托的所有用户均有权限进行关联与解除关联操作。

验证

登录ModelArts控制台，选择“专属资源池 > 网络”，单击“更多”，选择“关联sfsturbo”，关联成功。

登录ModelArts控制台，选择“专属资源池 > 网络”，单击“更多”，选择“解除关联”，解除成功。

11.3.10 给子账号配置文件夹级的 SFS Turbo 访问权限

场景描述

本文介绍如何配置文件夹级的SFS Turbo访问权限，实现在ModelArts中访问挂载的SFS Turbo时，只允许子账号访问特定的SFS Turbo文件夹内容。

说明

给子账号配置文件夹级的SFS Turbo访问权限为白名单功能，如果有试用需求，请提工单申请权限。

前提条件

- 需要在ModelArts控制台打开严格授权模式，单击“权限管理 > 启用严格模式”。
- 如果打开严格模式前没有为子账号配置过ModelArts权限，开启严格授权模式后可能会导致子账号无法使用ModelArts功能，请根据您的业务需求配置需要的ModelArts服务的权限（参见**依赖和委托**中ModelArts服务对应的依赖策略项）。

操作步骤

步骤1 使用主用户账号登录管理控制台，鼠标放在右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

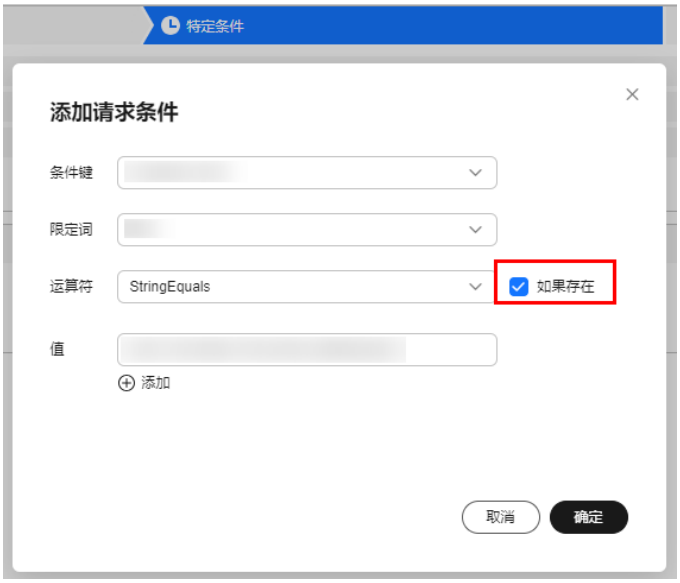
- “策略名称”：设置自定义策略名称，例如：ma_sfs_turbo。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "<modelarts_action>"
      ],
      "Condition": {
        "StringEqualsIfExists": {
          "modelarts:sfsId": [
            "<your_ssf_id>"
          ],
          "modelarts:sfsPath": [
            "<sfs_path>"
          ],
          "modelarts:sfsOption": [
            "<sfs_option>"
          ]
        }
      }
    }
  ]
}
```

说明

- 未创建以上权限策略前，所有子账号默认可以挂载SFS Turbo。当您创建了以上SFS权限管控策略后，没有被授予以上权限的子账号，默认在ModelArts Console上创建训练作业时无法挂载SFS Turbo（具有Tenant Administrator权限的子账号除外）。
- 当前仅支持配置允许策略的权限（即以上“策略内容”中的“Effect”只能配置为“Allow”），请勿配置拒绝策略的权限。
- Condition参数必须使用“StringEqualsIfExists”字段，对应可视化视图为勾选“如果存在”的开关。

图 11-19 “如果存在”的开关



以上代码中的"*<modelarts_action>*"、"*<your_ssf_id>*"、"*<sfs_path>*"、"*<sfs_option>*"，需要根据您的业务需求替换为实际的参数，各参数含义如下。

表 11-31 参数解释

参数	参数解释
Action	表示在何种场景下授予SFS Turbo文件夹访问权限。 ● 创建开发环境实例：modelarts:notebook:create ● 创建训练作业：modelarts:trainJob:create 支持填写多种Action，例如： "Action": ["modelarts:trainJob:create", "modelarts:notebook:create"],
modelarts:sfsId	SFS Turbo的ID，在SFS Turbo详情页查看。支持填写多个ID，例如： "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"],

参数	参数解释
modelarts:sfsPath	需要进行权限配置的SFS Turbo文件夹路径。支持填写多个路径，例如： "modelarts:sfsPath": ["/path1", "/path2/path2-1"], 如果sfsId中填写了多个ID，则sfsPath会应用于所有sfsId。例如以下代码含义为：为"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"的"/path1"和"/path2/path2-1"配置访问权限，同时也为"2a70da1e-ea87-4ee4-ae1e-55df846e7f41"的"/path1"和"/path2/path2-1"配置访问权限。 "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"], "modelarts:sfsPath": ["/path1", "/path2/path2-1"],

参数	参数解释

modelarts:sfsOption	设置用户对于SFS Turbo文件夹的权限类型，支持填写以下参数：
	• 仅读权限：readonly
	• 读写权限：readwrite（创建开发环境实例modelarts:notebook:create仅支持配置readwrite）
	如果需要在一个自定义策略中添加多个不同的sfsOption，需要“Statement”中新增JSON结构体，例如：
	{
	"Version": "1.1",
	"Statement": [
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:trainJob:create"
	],
	"Condition": {
	"StringEqualsIfExists": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path1"
	],
	"modelarts:sfsOption": [
	"readonly"
	]
	}
	}
	},
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:trainJob:create"
	],
	"Condition": {
	"StringEqualsIfExists": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path2"
	],
	"modelarts:sfsOption": [
	"readwrite"
	]
	}
	}
	}
	]
	}

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子账号配置权限。勾选**步骤2**中创建的“ma_sfs_turbo”策略。单击“下一步”和“确定”。

步骤5 在已有的ModelArts委托权限中，追加IAM ReadOnlyAccess权限。

1. 在ModelArts管理控制台，单击“权限管理”，在对应委托的操作列，单击“查看权限 > 去IAM修改委托权限”。

2. 在新页面中，单击“授权记录 > 授权”，搜索“IAM ReadOnlyAccess”，勾选后单击“下一步”并单击“确认”。

步骤6 验证权限是否配置成功。

登录子用户账号，在创建训练作业/创建Notebook时，仅能看到配置的SFS Turbo文件夹，则表示权限配置成功。

----结束

11.4 FAQ

11.4.1 使用 ModelArts 时提示“权限不足”，如何解决?

当您使用ModelArts时如果提示权限不足，请您按照如下指导对相关服务和用户进行授权，并对用户权限进行检查操作。

本案例中以OBS权限不足为例，介绍如何为用户授予OBS服务权限。其它权限不足的场景也可以参考本案例操作，只是授权范围不同。不同业务场景下的授权范围请参考**权限依赖和委托**章节。

由于ModelArts的使用权限依赖OBS服务的授权，您需要为用户授予OBS的系统权限。

- 如果您需要授予用户关于OBS的所有权限和ModelArts的基础操作权限，请参见**配置基础操作权限**。
- 如果您需要对用户使用OBS和ModelArts的权限进行精细化管理，进行自定义策略配置，请参见**创建ModelArts自定义策略**。

配置基础操作权限

使用ModelArts的基本功能，您需要为用户配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，由于ModelArts依赖OBS权限，您还需要登录IAM管理控制台为用户授予“作用范围”为“全局级服务”的“OBS Administrator”策略。

具体操作步骤如下：

步骤1 创建用户组。

登录IAM管理控制台，单击“用户组>创建用户组”。在“创建用户组”界面，输入“用户组名称”单击“确定”。

步骤2 配置用户组权限。

在用户组列表中，单击步骤1新建的用户组右侧的“授权”，在用户组“授权”页面，您需要配置的权限如下：

1. 配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，如下图所示，然后单击“确定”完成授权。

   说明

   区域级项目授权后只在授权区域生效，如果需要所有区域都生效，则所有区域都需要进行授权操作。

2. 配置“作用范围”为“全局级服务”的“OBS Administrator”权限，然后单击“确定”完成授权。

步骤3 创建用户并加入用户组。

在IAM控制台创建用户，并将其加入步骤1中创建的用户组。

步骤4 用户登录并验证权限。

新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择ModelArts，进入ModelArts主界面，选择不同类型的专属资源池，在页面单击“创建”，如果无法进行创建（当前权限仅包含ModelArts CommonOperations），表“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择除ModelArts外（假设当前策略仅包含ModelArts CommonOperations）的任一服务，如果提示权限不足，表示“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理>数据集>创建数据 > 集”，如果可以成功访问对应的OBS路径，表示全局级服务的“OBS Administrator”已生效。

----结束

创建 ModelArts 自定义策略

如果系统预置的ModelArts权限不满足您的授权要求，或者您需要管理用户操作OBS的操作权限，可以创建自定义策略。更多关于创建自定义策略操作和参数说明请参见**创建自定义策略**。

目前华为云支持可视化视图创建自定义策略和JSON视图创建自定义策略，本章节将使用JSON视图方式的策略，以为ModelArts用户授予开发环境的使用权限并且配置ModelArts用户OBS相关的最小化权限项为例，指导您进行自定义策略配置。

说明

如果一个自定义策略中包含多个服务的授权语句，这些服务必须是同一属性，即都是全局级服务或者项目级服务。

由于OBS为全局服务，ModelArts为项目级服务，所以需要创建两条“作用范围”别为“全局级服务”以及“项目级服务”的自定义策略，然后将两条策略同时授予用户。

1. 创建ModelArts相关OBS的最小化权限的自定义策略。

   登录IAM控制台，在“权限管理>权限”页面，单击“创建自定义策略”。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts依赖的OBS权限自定义策略样例**，如果您需要了解更多关于OBS的系统权限，请参见**OBS权限管理**。

2. 创建ModelArts开发环境的使用权限的自定义策略。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts开发环境使用权限的自定义策略样例**，ModelArts自定义策略中可以添加的授权项（Action）请参见**《ModelArts API参考》>权限策略和授权项**。
   - 如果您需要对除ModelArts和OBS之外的其它服务授权，IAM支持服务的所有策略请参见**权限策略**。

3. 在IAM控制台**创建用户组并授权**。

   在IAM控制台创建用户组之后，将步骤**1**中创建的自定义策略授权给该用户组。

4. **创建用户并加入用户组**。

   在IAM控制台创建用户，并将其加入**3**中创建的用户组。

5. **用户登录**并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理 > 数据集”，如果无法进行创建（当前仅包含开发环境的使用权限），表示仅为ModelArts用户授予开发环境的使用权限已生效。
   - 在“服务列表”中选择除ModelArts，进入ModelArts主界面，单击“开发环境>Notebook>创建”，如果可以成功访问“存储配置”项对应的OBS路径，表示为用户配置的OBS相关权限已生效。

ModelArts 依赖的 OBS 权限自定义策略样例

如下示例为ModelArts依赖OBS服务的最小化权限项，包含OBS桶和OBS对象的权限。授予示例中的权限您可以通过ModelArts正常访问OBS不受限制。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "obs:bucket:ListAllMybuckets",
        "obs:bucket:HeadBucket",
        "obs:bucket:ListBucket",
        "obs:bucket:GetBucketLocation",
        "obs:object:GetObject",
        "obs:object:GetObjectVersion",
        "obs:object:PutObject",
        "obs:object:DeleteObject",
        "obs:object:DeleteObjectVersion",
        "obs:object:ListMultipartUploadParts",
        "obs:object:AbortMultipartUpload",
        "obs:object:GetObjectAcl",
        "obs:object:GetObjectVersionAcl",
        "obs:bucket:PutBucketAcl",
        "obs:object:PutObjectAcl"
      ],
      "Effect": "Allow"
    }
  ]
}
```

ModelArts 开发环境使用权限的自定义策略样例

```
{
  "Version": "1.1",
  "Statement": [

    {
      "Effect": "Allow",
      "Action": [
        "modelarts:notebook:list",
        "modelarts:notebook:create" ,
        "modelarts:notebook:get" ,
        "modelarts:notebook:update" ,
        "modelarts:notebook:delete" ,
        "modelarts:notebook:action" ,
        "modelarts:notebook:access"
      ]
```



```
    }
  ]
}
```

12 Standard 自动学习

12.1 使用 ModelArts Standard 自动学习实现口罩检测

该案例是使用华为云一站式AI开发平台ModelArts的新版“自动学习”功能，基于华为云AI开发者社区AI Gallery中的数据集资产，让零AI基础的开发者完成“物体检测”的AI模型的训练和部署。依据开发者提供的标注数据及选择的场景，无需任何代码开发，自动生成满足用户精度要求的模型。可支持图片分类、物体检测、预测分析、声音分类等场景。可根据最终部署环境和开发者需求的推理速度，自动调优并生成满足要求的模型。

说明

费用说明：本案例使用过程中，从AI Gallery下载数据集免费，但是数据集存储在OBS桶中会收取少量费用，具体计费请参见**OBS价格详情页**。

在ModelArts上运行训练作业、将模型部署为在线服务会收取计算资源费用。案例使用完成后请参考**后续操作：清除相应资源**及时清除资源和数据。

步骤一：准备工作

- 注册华为账号并开通华为云、实名认证
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。具体配置操作请参见**配置ModelArts Standard访问授权**。

步骤二：创建训练数据集

1. 单击**口罩检测小数据集**进入数据集详情页，单击右侧“下载”。
2. 在弹出的窗口中选择云服务区域，例如该案例选择云服务区域为“华北-北京四”，单击“确定”进入下载详情页。
3. 在“下载详情”页面，填写参数。
   - 下载方式：ModelArts数据集。

- 目标区域：华北-北京四，目标区域须与上一步中选择的云服务区域保持一致。
- 数据类型：图片。
- 数据集输入位置：用来存放源数据集信息，例如本案例中从Gallery下载的数据集。单击图标选择您的OBS桶下的任意一处目录，但不能与输出位置为同一目录。
- 数据集输出位置：用来存放输出的数据标注的相关信息，或版本发布生成的Manifest文件等。单击图标选择OBS桶下的空目录，且此目录不能与输入位置一致，也不能为输入位置的子目录。
- 名称：创建数据集名称，为方便后续创建物体检测项目选择对应的数据集，建议您的数据集名称具有可识别性。
- 描述：描述数据集详细信息。

图 12-1 下载详情

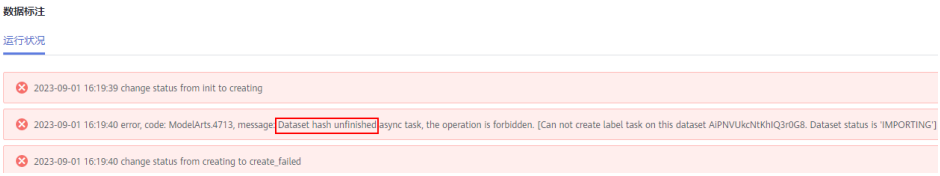
4. 确认无误后单击右下角“确定”。
5. 系统会跳转到我的下载页面，单击 按钮，查看下载进度，等待数据集下载完成（下载完成大约需要5分钟，请耐心等待）。单击 展开下载详情，可以查看该数据集的“目标位置”。
6. 查看数据集是否已导入ModelArts。

   返回ModelArts管理控制台，在左侧导航栏选择“数据集”，默认进入数据集新版页面。在新版数据集列表页，单击数据集名称左侧的 ，展开数据集，查看“导入状态”，导入状态为“导入完成”代表示数据集导入成功，且数据集正常。

说明

数据集下载完成后，请务必先检查数据集是否已经导入成功，如果数据集还未成功导入，创建自动学习物体检测项目后数据标注节点会报错。

图 12-2 数据标注节点报错



步骤三：创建自动学习物体检测项目

1. 确保数据集创建完成且可正常使用后，在ModelArts控制台，左侧导航栏选择“自动学习”默认进入新版自动学习页面，选择物体检测项目，单击“创建项目”。
2. 进入“创建物体检测”页面后，填写相关参数。
   - 计费模式：默认按需计费。
   - 名称：自行创建项目名称。
   - 描述：自行描述项目详情，例如口罩检测。
   - 数据集：下拉选择已下载的数据集（**步骤2**中已成功导入的数据集，默认为下拉数据集列表中的第一个数据集）。
   - 输出路径：选择**步骤2的3**中的数据集输出位置。
   - 训练规格：根据您的实际需要选择对应的训练规格。
3. 确认无误后单击右下角“创建项目”可自动跳转至自动学习的运行总览页面。

步骤四：运行工作流

在自动学习的运行总览页面，会产生一条工作流。工作流会自动从数据标注节点开始，依次运行数据集版本发布、数据校验、物体检测、模型注册、服务部署等节点，直至工作流全部运行完成。您需要做的是：

1. 在数据标注节点，待数据标注节点变为橘色即为“等待操作”状态，双击数据标注节点，打开数据标注节点的运行详情页面。前往实例详情页确认所有图片是否都标注完成，确认无误后，回到工作流页面单击“继续运行”。
2. 在“确认是否继续允许”的弹窗中，单击“确定”，工作流会继续从数据标注节点依次运行到服务部署节点。该段时间不需要用户做任何操作。
3. 当工作流运行到“服务部署”节点，“服务部署”节点会变成橙色，双击“服务部署”节点。在服务部署页签中，可以看到状态变为了“等待输入”。
4. 需要选择填写以下两个参数，其他参数均为默认值，保持不变。
   - 计算节点规格：根据您的实际需求选择相应的规格。
   - 是否自动停止：为避免资源浪费，建议打开自动停止开关，根据您的实际需要，选择自动停止时间，也可以自定义自动停止的时间。

图 12-3 选择计算节点规格

图 12-4 设置自动停止

5. 参数填写完毕之后，单击运行状况右边的“继续运行”，单击确认弹窗中的“确定”即可继续完成工作流的运行。

步骤五：预测分析

运行完成的工作流会自动部署为相应的在线服务，您只需要在相应的服务详情页面进行预测即可。

1. 在服务部署节点单击“实例详情”直接跳转进入在线服务详情页，或者在ModelArts管理控制台，选择“模型部署 > 在线服务”，单击生成的在线服务名称，即可进入在线服务详情页。
2. 在服务详情页，选择“预测”页签。

图 12-5 上传预测图片

3. 单击“上传”选择上传一张需要预测的图片，单击“预测”，即可在右边的预测结果显示区查看您的预测结果。

图 12-6 查看预测结果（1）--没戴口罩

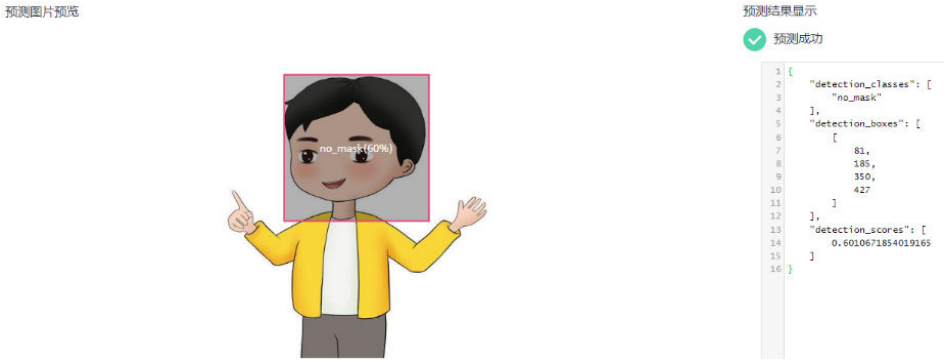
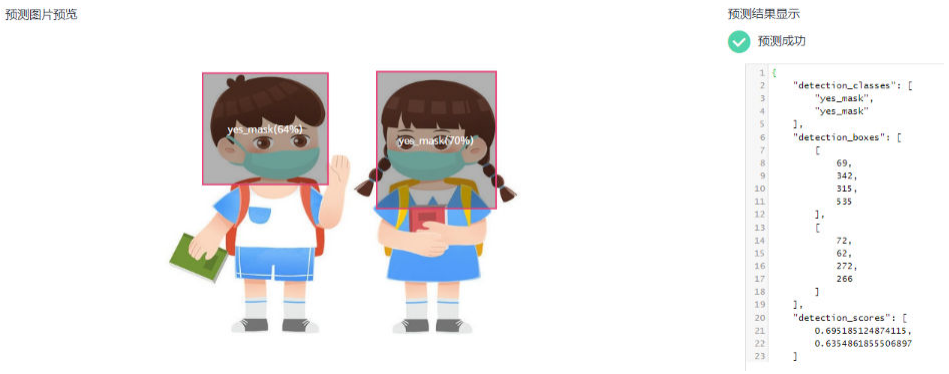


图 12-7 查看预测结果（2）--戴口罩



后续操作：清除相应资源

在完成预测之后，建议关闭服务，以免产生不必要的计费。

1. 停止运行服务
   - 预测完成后，单击页面右上角的“停止”，即可停止该服务。
   - 单击左上角 返回在线服务，在对应的服务名称所在行，单击选择操作列的“更多>停止”，停止该服务。

     图 12-8 停止服务

2. 清除OBS中的数据。
   a. 在控制台左侧导航栏的服务列表 ，选择“对象存储服务OBS”，进入OBS服务详情页面。
   b. 在左侧导航栏选择“桶列表”，在列表详情，找到自己创建的OBS桶，单击桶名称，进入OBS桶详情。
   c. 在桶的详情页，左侧导航栏选择“对象”，在右侧“名称”列选中不需要的存储对象，单击操作列的“更多>删除”，即可删除相应的存储对象。

常见问题

- 创建数据集时找不到创建的OBS桶，请**查看OBS桶与ModelArts是否在同一个区域**。
- 数据校验节点失败。
  请查看您的数据集是否符合规范，数据集规范请参考**数据集要求与上传规范**。

12.2 使用 ModelArts Standard 自动学习实现垃圾分类

随着科技发展与人们生活质量的快速提升，生活垃圾分类成为当下越来越热门的话题，常见的生活垃圾分为厨余垃圾蛋壳、厨余垃圾水果果皮、可回收物塑料玩具、可回收物纸板箱、其他垃圾烟蒂、其他垃圾一次性餐盒、有害垃圾干电池、有害垃圾过期药物等。人工识别效率低下、费时费力，AI技术显然可以为此贡献一份力量。

该案例介绍了华为云一站式开发平台ModelArts的自动学习功能实现的常见生活垃圾分类，让您不用编写代码也可以实现生活垃圾分类。

说明

本案例只适用于新版自动学习功能。

步骤一：准备工作

- 注册华为账号并开通华为云、实名认证
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。具体配置操作请参见**配置ModelArts Standard访问授权**。

步骤二：创建 OBS 桶

1. 登录**OBS管理控制台**，在桶列表页面右上角单击“创建桶”，创建OBS桶。例如，创建名称为“dataset-exeml”的OBS桶。

   说明

   - 创建桶的区域需要与ModelArts所在的区域一致。例如：当前ModelArts在华北-北京四区域，在对象存储服务创建桶时，请选择华北-北京四。请参考**查看OBS桶与ModelArts是否在同一区域**检查您的OBS桶区域与ModelArts区域是否一致。
   - 请勿开启桶加密，ModelArts不支持加密的OBS桶，会导致ModelArts读取OBS中的数据失败。
2. 在桶列表页面，单击桶名称，进入该桶的概览页面。
3. 单击左侧导航的“对象”，在对象页面单击“新建文件夹”，创建OBS文件夹。具体请参见**新建文件夹**章节。

步骤三：准备训练数据集

1. 单击**8类常见生活垃圾图片数据集**，进入AI Gallery数据集详情页，单击右侧“下载”。
2. 选择对应的云服务区域例如：华北-北京四，需要确保您选择的区域与您的管理控制台所在的区域一致。
3. 进入“下载详情”页面，填写以下参数。
   - 下载方式：ModelArts数据集。
   - 目标区域：华北-北京四。
   - 数据类型：系统会根据您的数据集，匹配到相应的数据类型。例如本案例使用的数据集，系统匹配为“图片”类型。
   - 数据集输入位置：用来存放源数据集信息，例如本案例中从Gallery下载的数据集。单击图标选择您的OBS桶下的任意一处目录，但不能与输出位置为同一目录。
   - 数据集输出位置：用来存放输出的数据标注的相关信息，或版本发布生成的Manifest文件等。单击图标选择OBS桶下的空目录，且此目录不能与输入位置一致，也不能为输入位置的子目录。

图 12-9 下载详情

4. 完成参数填写，单击“确定”，自动跳转至AI Gallery个人中心“我的下载”页签，单击 按钮，查看下载进度，等待5分钟左右下载完成，单击 展开下载详情，可以查看该数据集的“目标位置”。

步骤四：创建新版自动学习图像分类项目

1. 确保数据集创建完成且可正常使用后，在**ModelArts控制台**，左侧导航栏选择“开发空间 > 自动学习”，进入自动学习总览页面。
2. 单击选择“图像分类”创建项目。完成参数填写。
   - 名称：自定义您的项目名称。
   - 描述：自定义描述您的项目详情，例如垃圾分类。
   - 数据集：下拉选择已下载的数据集（**步骤2**中已成功导入的数据集，默认为下拉数据集列表中的第一个数据集）。
   - 输出路径：选择您**步骤1**创建好的OBS文件夹下的路径，用来存储训练模型等相关文件。
   - 训练规格：根据您的实际需要选择对应的训练规格。
3. 参数填写完成，单击“创建项目”。

步骤五：运行工作流

项目完成创建之后，会自动跳转到新版自动学习的运行总览页面。同时您的工作流会自动从数据标注节点开始运行。您需要做的是：

1. 观察数据标注节点，待数据标注节点变为橙色即为“等待操作”状态。双击数据标注节点，打开数据标注节点的运行详情页面，单击“继续运行”。
2. 在弹出的窗口中，单击“确定”，工作流会开始继续运行。当工作流运行到“服务部署”节点，状态会变为“等待输入”，您需要填写以下两个输入参数，其他参数保持默认。
   - 计算节点规格：根据您的实际需求选择相应的规格，不同规格的配置费用不同，选择好规格后，配置费用处会显示相应的费用。

- 是否自动停止：为了避免资源浪费，建议您打开该开关，根据您的需求，选择自动停止时间，也可以自定义自动停止的时间。

图 12-10 选择计算节点规格

图 12-11 设置自动停止

3. 参数填写完毕之后，单击运行状况右边的“继续运行”，单击确认弹窗中的“确定”即可继续完成工作流的运行。

步骤六：预测分析

运行完成的工作流会自动部署相应的在线服务，您只需要在相应的服务详情页面进行预测即可。

1. 在服务部署节点单击“实例详情”或者在ModelArts管理控制台，选择“模型部署 > 在线服务”，单击生成的在线服务名称，即可进入在线服务详情页。
2. 在服务详情页，单击选择“预测”页签。

图 12-12 上传预测图片

3. 单击“上传”，选择一张需要预测的图片，单击“预测”，即可在右边的预测结果显示区查看您的预测结果。

图 12-13 预测样例图

图 12-14 查看预测结果

说明

本案例中数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练数据集中的图片相似才可能预测准确。

ModelArts的AI Gallery中提供了常见的精度较高的算法和相应的训练数据集，用户可以在**AI Gallery的资产集市**中获取。

后续操作：清除相应资源

在完成预测之后，建议关闭服务，以免产生不必要的计费。

1. 停止运行服务
   - 预测完成后，单击页面右上角的“停止”，即可停止该服务。
   - 单击左上角 < 返回在线服务，在对应的服务名称所在行，单击选择操作列的“更多>停止”，停止该服务。

图 12-15 停止服务

2. 清除OBS中的数据。

   a. 在控制台左侧导航栏的服务列表 ☰，选择“对象存储服务OBS”，进入OBS服务详情页面。

   b. 在左侧导航栏选择“桶列表”，在列表详情，找到自己创建的OBS桶，单击桶名称，进入OBS桶详情。

   c. 在桶的详情页，左侧导航栏选择“对象”，在右侧“名称”列选中不需要的存储对象，单击“操作”列的“更多>删除”，即可删除相应的存储对象。

常见问题

- 创建数据集时找不到创建的OBS桶，请**查看OBS桶与ModelArts是否在同一个区域**。
- 数据校验节点失败。

  请查看您的数据集是否符合规范，数据集规范请参考**数据集要求与上传规范**。

13 Standard 开发环境

13.1 将 Notebook 的 Conda 环境迁移到 SFS 磁盘

本文介绍了如何将Notebook的Conda环境迁移到SFS磁盘上。这样重启Notebook实例后，Conda环境不会丢失。

步骤如下:

1. **创建新的虚拟环境并保存到SFS目录**
2. **克隆原有的虚拟环境到SFS盘**
3. **重新启动镜像激活SFS盘中的虚拟环境**
4. **保存并共享虚拟环境**

前提条件

创建一个Notebook，“资源类型”选择“专属资源池”，“存储配置”选择“SFS弹性文件服务器”，打开terminal。

创建新的虚拟环境并保存到 SFS 目录

创建新的conda虚拟环境。

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

查看现有的conda虚拟环境，此时可能出现新创建的虚拟环境的名称为空的情况。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
                      /home/ma-user/work/envs/user_conda/sfs-new-env
```

添加新创建的虚拟环境到conda env。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

查看现有的conda虚拟环境，此时新的虚拟环境已经能够正常显示，可以直接通过名称进行虚拟环境的切换。

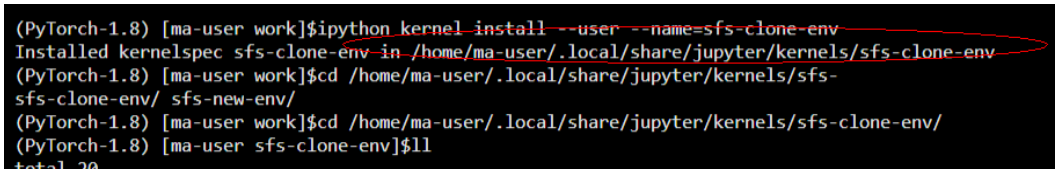
```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）。

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-new-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-new-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-new-env”为举例，请以用户实际的安装路径为准。

图 13-1 安装路径回显



刷新JupyterLab页面，可以看到新的kernel。

说明

重启Notebook后kernel需要重新注册。

克隆原有的虚拟环境到 SFS 盘

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-clone-env --clone PyTorch-1.8 -
y
Source:      /home/ma-user/anaconda3/envs/PyTorch-1.8
Destination: /home/ma-user/work/envs/user_conda/sfs-clone-env
Packages: 20
Files: 39687
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/ma-user/work/envs/user_conda/sfs-clone-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

查看新创建的clone虚拟环境，如果出现新创建的虚拟环境的名称为空的情况，可以参考**添加新创建到虚拟环境到conda env**。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10             /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env             /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env           *  /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-clone-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-clone-env”为举例，请以用户实际的安装路径为准。

刷新JupyterLab页面，可以看到新的kernel。

重新启动镜像激活 SFS 盘中的虚拟环境

方法一，直接使用完整conda env路径。

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

方法二，先添加虚拟环境到conda env，然后使用名称激活。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

方法三，直接使用完成虚拟环境中的python或者pip。

```
# shell
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/pip list
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/python -V
```

保存并共享虚拟环境

将要迁移的虚拟环境打包。

```
# shell
pip install conda-pack
conda pack -n sfs-clone-env -o sfs-clone-env.tar.gz --ignore-editable-packages
Collecting packages...
Packing environment at '/home/ma-user/work/envs/user_conda/sfs-clone-env' to 'sfs-clone-env.tar.gz'
[########################################] | 100% Completed |  3min 33.9s
```

解压到SFS目录。

```
# shell

mkdir /home/ma-user/work/envs/user_conda/sfs-tar-env
tar -zxvf sfs-clone-env.tar.gz -C /home/ma-user/work/envs/user_conda/sfs-tar-env
```

查看现有的conda虚拟环境。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8           *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
sfs-tar-env             /home/ma-user/work/envs/user_conda/sfs-tar-env
test-env                /home/ma-user/work/envs/user_conda/test-env
```

13.2 使用 ModelArts VSCode 插件调试训练 ResNet50 图像分类模型

应用场景

Notebook等线上开发工具工程化开发体验不如IDE，但是本地开发服务器等资源有限，运行和调试环境大多使用团队公共搭建的CPU或GPU服务器，并且是多人共用，这带来一定的环境搭建和维护成本。因此使用本地IDE+远程Notebook结合的方式，可以同时享受IDE工程化开发和云上资源的即开即用，优势互补，满足开发者需求。

VS Code在Python项目开发中提供了优秀的代码编辑、调试、远程连接和同步能力，在开发者中广受欢迎。本文以Ascend Model Zoo为例，介绍如何通过VS Code插件及ModelArts Notebook进行云端数据调试及模型开发。

方案优势

云端开发调试优势：

- 环境保持一致
- 配置一键完成
- 代码远程调试
- 资源按需使用

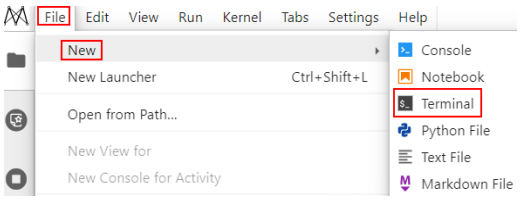
准备工作

1. 下载VS Code IDE，下载路径：**开源Visual Studio Code**。根据不同的操作系统选择不同的安装包。
2. 创建Notebook实例。
   a. **登录ModelArts控制台**，单击左侧导航“开发环境 > Notebook”，然后单击“创建”。

      镜像选择“mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3”，类型选择“ASCEND”，并打开“SSH远程开发”开关，密钥对选择已有的或单击“立即创建”。

   b. Notebook创建后，“状态”为“运行中”。单击“操作”列的“打开”，进入JupyterLab，然后参考下图打开Terminal。

图 13-2 打开 Terminal

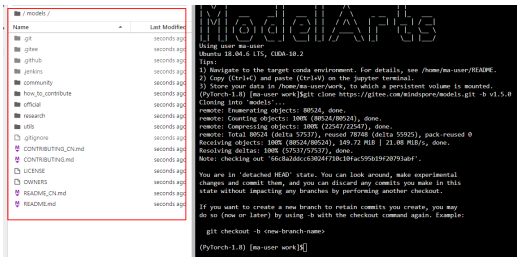


3. 下载项目代码。

   在Terminal执行如下命令下载项目代码。本例中，以图像分类模型resnet50模型为例。下载后的文件如图13-3所示，代码所在路径为“./models/official/cv/resnet/”。

   ```
   # 下载代码
   git clone https://gitee.com/mindspore/models.git -b v1.5.0
   ```

   图 13-3 下载后的模型包文件

4. 下载花卉识别数据集。

   本样例使用的数据集为类别数为五类的花卉识别数据集。

   在Terminal里执行如下命令下载并解压数据集，将数据集保存在“./models/dataset/flower_photos”文件夹。

   ```
   cd models
   mkdir dataset
   cd dataset
   wget http://download.tensorflow.org/example_images/flower_photos.tgz
   tar zxvf flower_photos.tgz
   ```

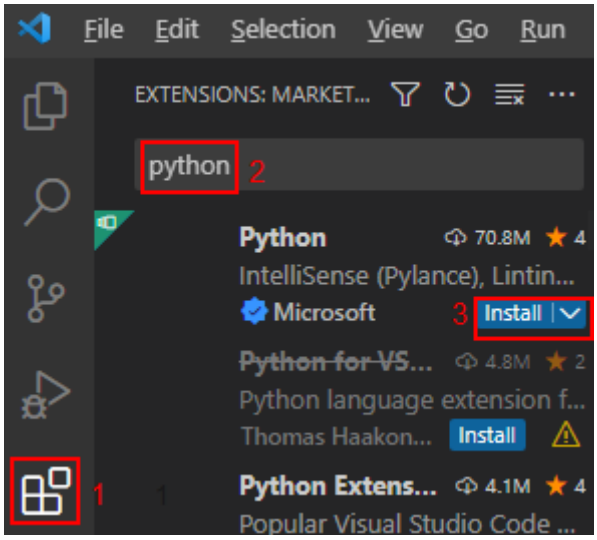
步骤 1：通过 VS Code 插件连接云端 Notebook

通过VS Code插件连接**准备工作**里创建的云端Notebook，详细操作请参考**VS Code一键连接Notebook**。

步骤 2：安装 Python 插件以及配置入参

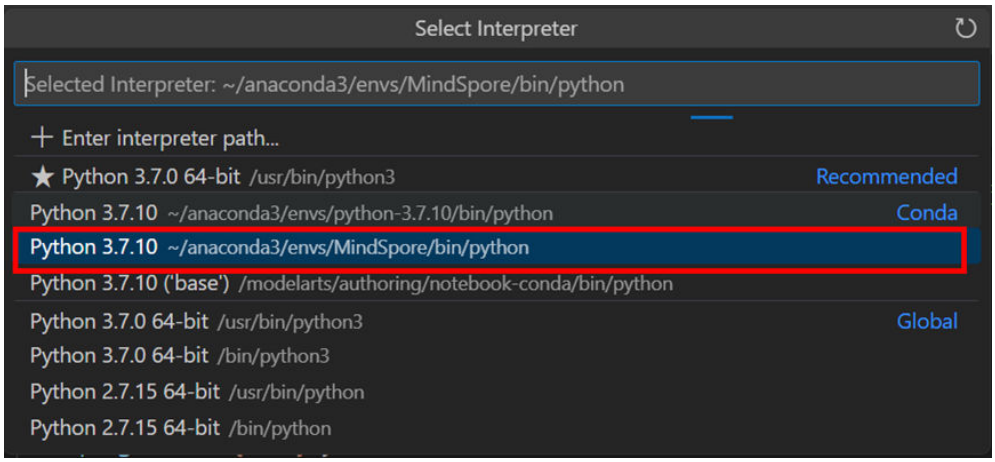
1. 打开VS Code工具，单击“Extensions”，搜索python，然后单击“Install”。

图 13-4 安装 Python



2. 输入Ctrl+Shirt+P，搜索“python:select interpreter”，选择Python解释器。

图 13-5 选择 python 解释器



3. 单击“RUN > Add Configuration...” 选择Python > Python File，填入如下代码。

   如果文件已创建，单击“RUN > Open Configurations”，填入如下代码。

   # 根据README说明文档，配置的Parameter入参如下，其中device_target="CPU"表示CPU环境运行，device_target="Ascend"表示在Ascend环境运行

```
"configurations": [
    {
        "name": "Python: Django Debug Single Test",
        "type": "python",
        "request": "launch",
        "program": "${file}",
        "args":[
            "--net_name", "resnet50",
            "--dataset", "imagenet2012",
            "--data_path", "/home/ma-user/work/models/dataset/flower_photos/",
            "--class_num", "5",
            "--config_path", "/home/ma-user/work/models/official/cv/resnet/config/
```



```
resnet50_imagenet2012_config.yaml",
                "--epoch_size", "1",
                "--device_target","Ascend"
            ]
        }
    ]
```

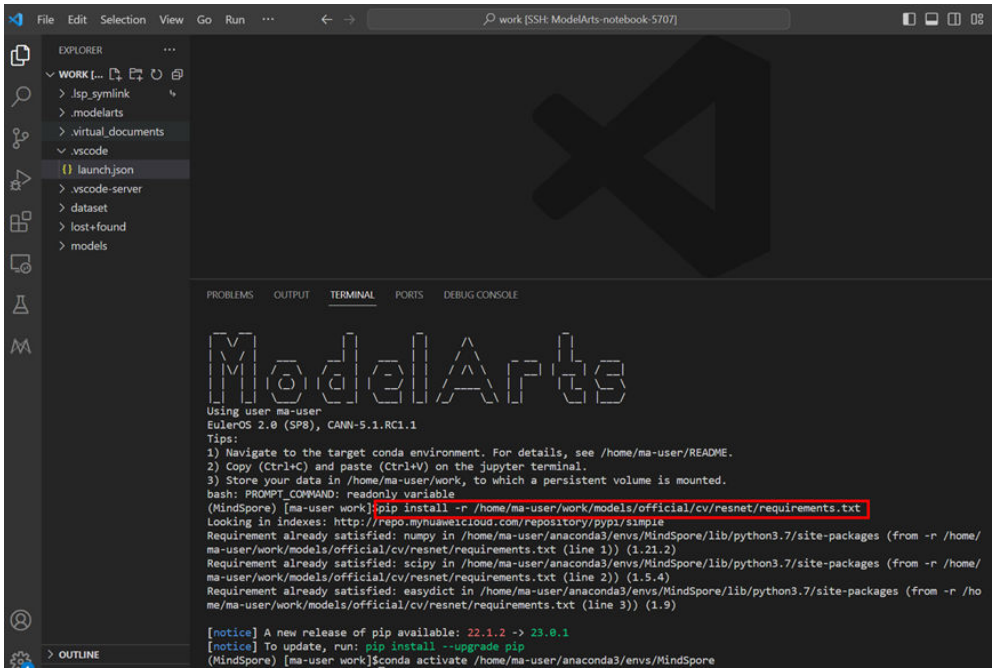
步骤 3：在 VS Code 中远程调试代码

1. 参考**准备工作**上传本地代码和数据至云端Notebook。
2. 云端Notebook安装依赖。

   在本地IDE中打开“Terminal > New Terminal”，执行如下命令。

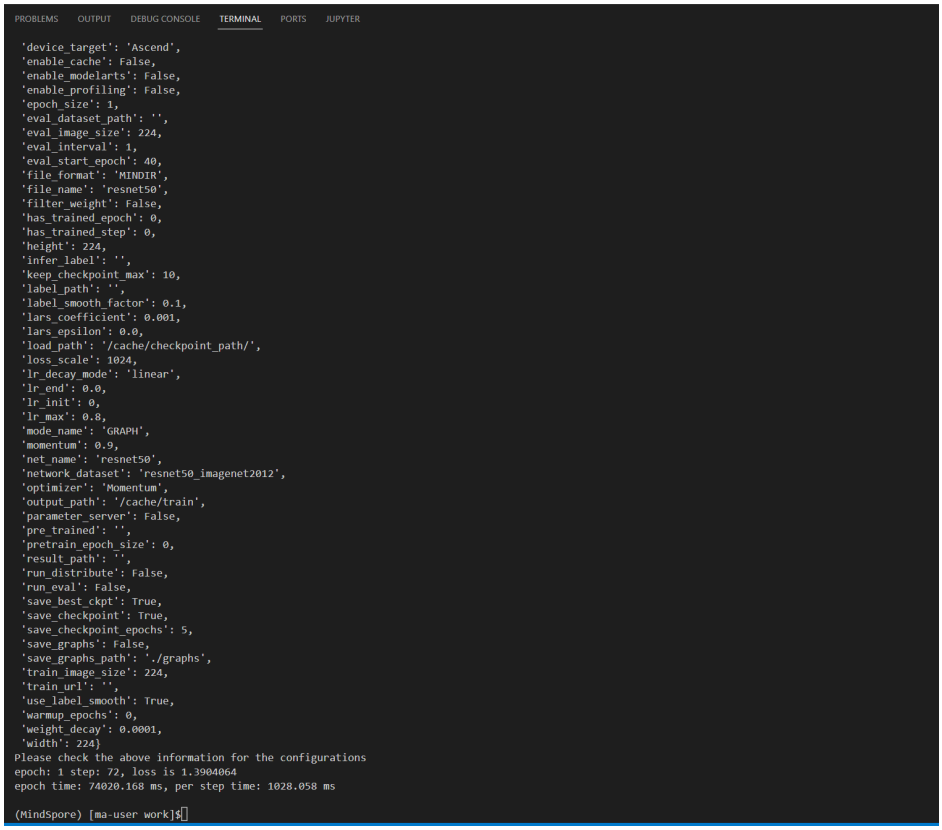
   ```
   pip install -r /home/ma-user/work/models/official/cv/resnet/requirements.txt
   ```

   图 13-6 执行命令

3. 云端调试与运行。
   a. 打开训练文件。文件所在路径为“/home/ma-user/work/models/official/cv/resnet/train.py”
   b. 代码调测：在需要调测点打断点，然后单击“RUN > Start Debugging”。
   c. 代码运行：单击“RUN > Run Without Debugging”，运行结果如下：

图 13-7 代码运行结果



步骤 4：保存开发环境镜像

完成Notebook调测后，此时的Notebook已经包含了模型训练所有的依赖环境，因此可以将已经调测完成的开发环境保存成一个镜像。

- 方式一：保存镜像需要指定镜像名称、镜像标签、SWR服务的组织等信息，保存镜像需要等待几分钟时间，期间不能对Notebook有额外操作。

  SWR服务的组织可以在SWR服务中进行创建，也可以使用SDK创建默认的SWR组织，默认最多只能创建5个组织。

  a. 在“/home/ma-user/work/models/official/cv/resnet/”下创建save_image.py，

  b. 复制代码至save_image.py，

  c. 运行save_image.py，进行保存镜像。

  save_image.py代码如下：

```
# save_image.py
# 导入ModelArts SDK的依赖，并初始化Session，此处的ak、sk、project_id、region_name请替换成用户自己的信息
from modelarts.session import Session
# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')
```

```
# 保存notebook镜像
from modelarts.image_mgmt import ImageSave
from modelarts.service import SWRManagement
# 创建一个镜像组织。如果组织数量已超过阈值，则会报错“namespace is invalid”，需要删除一个组织或手动指定一个已有的组织信息（使用image_organization = “your-swr-namespace-name”指定）
image_organization = SWRManagement(session).get_default_namespace()
# image_organization = “your-swr-namespace-name”
print("Default image_organization:", image_organization)
image_name = "mindspore-image-models-image" #@param {type:"string"}
image_tag = "1.0.0" #@param {type:"string"}
image_save = ImageSave(session=session, name=image_name, tag=image_tag, organization=image_organization)
image_save.save()
```

- 方式二：在ModelArts控制台单击“保存镜像”。

  在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列的“更多 > 保存镜像”，进入“保存镜像”页面，设置组织、镜像名称、镜像版本和描述信息后单击“确认”保存镜像。此时Notebook会冻结，需要等待几分钟。详细操作请参考**保存Notebook镜像环境**。

  图 13-8 保存镜像

查看所保存的镜像

保存后的镜像可以在ModelArts控制台“镜像管理”页面查看到该镜像详情。单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

步骤 5：使用 SDK 提交训练作业

本地调测完成后可以提交训练作业。因为数据在Notebook中，设置InputData中“is_local_source”的参数为“True”，会自动将本地数据同步上传到OBS中。

步骤如下：

1. 在“/home/ma-user/work/models/official/cv/resnet/”下创建train_notebook.py，
2. 复制代码至train_notebook.py，
3. 运行train_notebook.py，进行训练作业提交。

```
# train_notebook.py
# 导入ModelArts SDK的依赖，并初始化Session，此处的ak、sk、project_id、region_name请替换成用户自己的信息
from modelarts.train_params import TrainingFiles
```

```
from modelarts.train_params import OutputData
from modelarts.train_params import InputData
from modelarts.estimatorV2 import Estimator
from modelarts.session import Session

# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')

# 样例中为了方便默认创建一个OBS桶，推荐将调测所需要传输的文件统一放到`${default_bucket}/intermidiate`目录下，也可以按照注释代码自行指定

obs_bucket = session.obs.get_default_bucket()
print("Default bucket name: ", obs_bucket)
default_obs_dir = f"{obs_bucket}/intermidiate"
#default_obs_dir = "obs://your-bucket-name/folder-name"

# 本地的工程代码文件夹路径
code_dir_local = "/home/ma-user/work/models/official/cv/resnet/" #@param {type:"string"}

# 代码的启动文件名称
boot_file = "train.py" #@param {type:"string"}
train_file = TrainingFiles(code_dir=code_dir_local, boot_file=boot_file)

# 本地数据集路径
local_data_path = "/home/ma-user/work/models/dataset/flower_photos" #@param {type:"string"}

# 模型输出保存路径
output_local = "/home/ma-user/work/models/official/cv/resnet/output" #@param {type:"string"}
# 模拟训练过程中模型输出回传至指定OBS的路径，需要以"/"结尾
obs_output_path = f"{default_obs_dir}/mindspore_model/output/"

# 指定一个obs路径用于存储输出结果
output = [OutputData(local_path=output_local, obs_path=obs_output_path, name="output")]

# 模拟训练过程中模训练日志回传至指定OBS的路径，需要以"/"结尾
log_obs_path = f"{default_obs_dir}/mindspore_model/logs/"

# 训练所需的代码路径，代码会自动从本地上传至OBS
code_obs_path = f"{default_obs_dir}/mindspore_model/"
data_obs_path = f"{default_obs_dir}/dataset/flower_photos/"

# sdk会将代码自动上传至OBS，并同步到训练环境
train_file = TrainingFiles(code_dir=code_dir_local, boot_file=boot_file, obs_path=code_obs_path)

# 指定OBS中的数据集路径，会自动将local_path数据上传至obs_path，用户可以在代码中通过 --data_url接收这个数据集路径
input_data = InputData(local_path=local_data_path, obs_path=data_obs_path,
is_local_source=True, name="data_url")


from modelarts.service import SWRManagement
image_organization = SWRManagement(session).get_default_namespace()
# image_organization = "your-swr-namespace-name"
print("Default image_organization:", image_organization)
```

```
image_name = "mindspore-image-models-image" #@param {type:"string"}
image_tag = "1.0.0" #@param {type:"string"}

import os
ENV_NAME=os.getenv('ENV_NAME')

# 启动训练作业：使用user_command（shell命令）方式启动训练作业
# 注意：训练启动默认的工作路径为"/home/ma-user/modelarts/user-job-dir"，而代码上传路径为"./resnet/${code_dir}"下
# --enable_modelarts=True 该代码仓已适配ModelArts
estimator = Estimator(session=session,
                      training_files=train_file,
                      outputs=output,
                      user_image_url=f"{image_organization}/{image_name}:{image_tag}", # 自定义镜像swr地址，由镜像仓库组织/镜像名称:镜像tag组成
                      user_command=f'cd /home/ma-user/modelarts/user-job-dir/ && /home/ma-user/anaconda3/envs/MindSpore/bin/python ./resnet/train.py --net_name=resnet50 --dataset=imagenet2012 --enable_modelarts=True --class_num=5 --config_path=./resnet/config/resnet50_imagenet2012_config.yaml --epoch_size=10 --device_target="Ascend" --enable_modelarts=True', # 执行训练命令
                      train_instance_type="modelarts.p3.large.public", # 虚拟资源规格，不同region的资源规格可能不同，请参考“Estimator参数说明”表下的说明查询修改
                      train_instance_count=1, # 节点数，适用于多机分布式训练，默认是1
                      #pool_id='若指定专属池，替换为页面上查到的poolId'，同时修改资源规格为专属池专用的虚拟子规格
                      log_url=log_obs_path
                     )
# job_name是可选参数，可不填随机生成工作名
job_instance = estimator.fit(inputs=[input_data],
                             job_name="modelarts_training_job_with_sdk_by_command_v01")
```

表 13-1 Estimator 参数说明

参数名称	参数说明
session	modelarts session
training_files	训练代码的路径和启动文件
user_image_url	自定义镜像swr地址，由镜像仓库组织/镜像名称:镜像tag组成
user_command	执行训练命令
train_instance_type	本地调测'local'或云端资源规格。每个region的资源规格可能是不同的，可以通过下述说明查询对应的资源规格信息。
train_instance_count	节点数
log_url	日志输出路径
job_name	作业名称，不可以重复

说明

train_instance_type表示训练的资源规格，每个region的资源规格可能是不同的。通过如下方法查询资源规格：

- 公共资源池执行如下命令查询

```
from modelarts.session import Session
from modelarts.estimatorV2 import Estimator
from pprint import pprint

# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')

info = Estimator.get_train_instance_types(session=session)
pprint(info)
```

- 专属池规格

  ModelArts专属资源池统一使用虚拟子规格，不区分GPU和Ascend。资源规格参考**表13-2**查询。

表 13-2 专属资源池虚拟规格的说明

train_instance_type	说明
modelarts.pool.visual.xlarge	1卡
modelarts.pool.visual.2xlarge	2卡
modelarts.pool.visual.4xlarge	4卡
modelarts.pool.visual.8xlarge	8卡

步骤 6：清除资源

Notebook在代码调试完成及提交训练作业后就可以关闭了，减少资源扣费。

当调测完成且实例处于运行状态时，单击停止；

当下次调测且实例处于停止状态时，单击启动实例，随开随用。

训练输出保存结构说明

ModelArts训练作业的模型输出和日志信息会定时同步到指定的OBS中，本示例中模型输出路径和日志输出路径分别为f"{default_obs_dir}/mindspore_model/output/"和f"{default_obs_dir}/mindspore_model/logs/"，用户可以在OBS中查看训练输出信息。

本示例中训练输出保存在OBS的目录结构如下所示：

```
${your_bucket}
└── intermidiate
├── dataset
```

```
│   └── flower_photos
│   └── flower_photos.zip
└── mindspore_model
├── logs
│   └── xxx-xxx-xxx--0.log
├── output
│   └── 20220627-105226-resnet50-224
└── mindspore-image-models.zip
```

提交训练作业常见问题

- **报错信息：Exception: You have attempted to create more buckets than allowed**

  原因分析：由于桶的数量多于限额，无法自动创建。

  解决方法：用户可以删除一个桶，或者直接指定一个已存在的桶（修改变量obs_bucket的值）。

- **报错信息："errorMessage":"The number of namespaces exceeds the upper limit"或"namespace is invalid"**

  原因分析：SWR组织数限额，SWR组织默认最多只能创建5个组织。

  解决方法：用户可以删除一个SWR组织，或者直接指定一个已存在的SWR组织（修改变量image_organization的值）。

- **报错信息：standard_init_linux.go:224: exec user process caused "exet format error"**

  原因分析：可能由于训练规格错误导致训练作业卡死。

  解决方法：请参考**说明**查询资源规格。

- **报错信息：报错镜像失败，报错：401，'Unauthorized'，b'{errors": [{"errorCode":"SVCSTG.SWR.4010000",errorMessage":"Authenticate Error",……}]**

  原因分析：远程连接Notebook时需要输入鉴权信息。

  解决方法：传入AK，SK信息。

  ```
  # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
  # 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
  __AK = os.environ["HUAWEICLOUD_SDK_AK"]
  __SK = os.environ["HUAWEICLOUD_SDK_SK"]
  # 如果进行了加密还需要进行解密操作
  session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')
  ```

14 Standard 模型训练

14.1 使用 ModelArts Standard 自定义算法实现手写数字识别

本文为用户提供如何将本地的自定义算法通过简单的代码适配，实现在ModelArts上进行模型训练与部署的全流程指导。

场景描述

本案例用于指导用户使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。

通过学习本案例，您可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

操作流程

开始使用如下样例前，请务必按**准备工作**指导完成必要操作。

1. **Step1 准备训练数据**：下载MNIST数据集。
2. **Step2 准备训练文件和推理文件**：编写训练与推理代码。
3. **Step3 创建OBS桶并上传文件**：创建OBS桶和文件夹，并将数据集和训练脚本，推理脚本，推理配置文件上传到OBS中。
4. **Step4 创建训练作业**：进行模型训练。
5. **Step5 推理部署**：训练结束后，将生成的模型导入ModelArts用于创建模型，并将模型部署为在线服务。
6. **Step6 预测结果**：上传一张手写数字图片，发起预测请求获取预测结果。
7. **Step7 清除资源**：运行完成后，停止服务并删除OBS中的数据，避免不必要的扣费。

准备工作

- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在弹出的“添加授权”窗口中，选择：

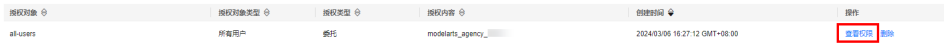
  - **授权对象类型：所有用户**
  - **委托选择：新增委托**
  - **权限配置：普通用户**

  选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

  图 14-1 配置委托访问授权

  c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

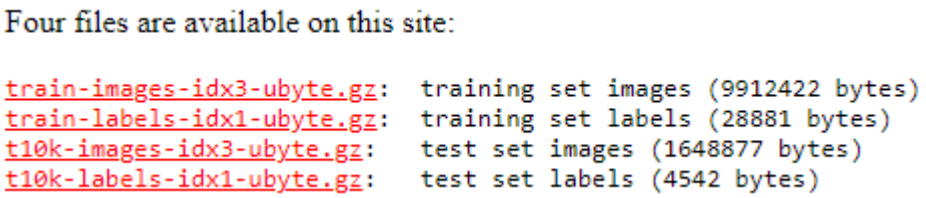
  图 14-2 查看委托配置信息



Step1 准备训练数据

本案例使用的数据是MNIST数据集，您可以在浏览器中搜索“MNIST数据集”下载如图14-3所示的4个文件。

图 14-3 MNIST 数据集



- “train-images-idx3-ubyte.gz”：训练集的压缩包文件，共包含60000个样本。

- “train-labels-idx1-ubyte.gz”：训练集标签的压缩包文件，共包含60000个样本的类别标签。
- “t10k-images-idx3-ubyte.gz”：验证集的压缩包文件，共包含10000个样本。
- “t10k-labels-idx1-ubyte.gz”：验证集标签的压缩包文件，共包含10000个样本的类别标签。

Step2 准备训练文件和推理文件

针对此案例，ModelArts提供了需使用的训练脚本、推理脚本和推理配置文件。请参考如下文件内容。

说明

粘贴“.py”文件代码时，请直接新建“.py”文件，否则会可能出现“SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence”报错。

粘贴完代码后，建议检查代码文件是否出现中文注释变为乱码的情况，如果出现该情况请将编辑器改为utf-8格式后再粘贴代码。

在本地电脑中创建训练脚本“train.py”，内容如下：

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# 定义网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
```



```
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


# 模型训练，设置模型为训练模式，加载训练数据，计算损失函数，执行梯度下降
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


# 模型验证，设置模型为验证模式，加载验证数据，计算损失函数和准确率
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


# 以下为pytorch mnist
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)


def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
```

```
}


def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
    """Read an SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
       Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# --- 以上为pytorch mnist
# --- end


# raw mnist 数据处理
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
```

```
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
            test.pt
    """
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # 定义可以接收的训练作业运行参数
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')

    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True,
```

```
                    help='For Saving the current Model')
    args = parser.parse_args()

    use_cuda = not args.no_cuda and torch.cuda.is_available()

    torch.manual_seed(args.seed)

    # 设置使用 GPU 还是 CPU 来运行算法
    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # 定义数据预处理方法
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # 将 raw mnist 数据集转换为 pytorch mnist 数据集
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # 分别创建训练和验证数据集
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)

    # 分别构建训练和验证数据迭代器
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # 初始化神经网络模型并复制模型到计算设备上
    model = Net().to(device)
    # 定义训练优化器和学习率策略，用于梯度下降计算
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # 训练神经网络，每一轮进行一次验证
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # 保存模型与适配 ModelArts 推理模型包规范
    if args.save_model:

        # 在 train_url 训练参数对应的路径内创建 model 目录
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # 按 ModelArts 推理模型包规范，保存模型到 model 目录内
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # 复制推理代码与配置文件到 model 目录内
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

if __name__ == '__main__':
    main()
```

在本地电脑中创建推理脚本“customize_service.py”，内容如下：

```
import os
import log
import json

import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 模型推理服务
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)

        # 调用自定义函数加载模型
        self.model = Mnist(model_path)

        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # 接收request数据，并转换为模型可以接受的输入格式
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    # 将推理的结果进行后处理，得到预期的输出格式，该结果就是最终的返回值
    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    # 对于输入数据进行前向推理，得到推理结果
    def _inference(self, data):
```

```
        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()

    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))

    # CPU 或者 GPU 映射
    model.to(device)

    # 声明为推理模式
    model.eval()

    return model
```

在本地电脑中推理配置文件“config.json”，内容如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

Step3 创建 OBS 桶并上传文件

将上一步中的数据和代码文件、推理代码文件与推理配置文件，从本地上传到OBS桶中。在ModelArts上运行训练作业时，需要从OBS桶中读取数据和代码文件。

1. 登录OBS管理控制台，按照如下示例创建OBS桶和文件夹。

   ```
   {OBS桶}                  # OBS对象桶，用户可以自定义名称，例如：test-modelarts-xx
       -{OBS文件夹}         # OBS文件夹，自定义名称，此处举例为pytorch
   ```

```
      - mnist-data      # OBS文件夹，用于存放训练数据集，可以自定义名称，此处举例为mnist-data
      - mnist-code      # OBS文件夹，用于存放训练脚本train.py，可以自定义名称，此处举例为mnist-code
          - infer       # OBS文件夹，用于存放推理脚本customize_service.py和配置文件config.json
      - mnist-output    # OBS文件夹，用于存放训练输出模型，可以自定义名称，此处举例为mnist-output
```

注意

- 创建的OBS桶所在区域和后续使用ModelArts必须在同一个区域Region，否则会导致训练时找不到OBS桶。具体操作可参见**查看OBS桶与ModelArts是否在同一区域**。
- 创建OBS桶时，桶的存储类别请勿选择“归档存储”，归档存储的OBS桶会导致模型训练失败。

2. 上传**Step1 准备训练数据**中下载的MNIST数据集压缩包文件到OBS的“mnist-data”文件夹中。

注意

- 上传数据到OBS中时，请不要加密，否则会导致训练失败。
- 文件无需解压，直接上传压缩包至OBS中即可。

3. 上传训练脚本“train.py”到“mnist-code”文件夹中。
4. 上传推理脚本“customize_service.py”和推理配置文件“config.json”到“mnist-code”的“infer”文件中。

Step4 创建训练作业

1. 登录ModelArts管理控制台，选择和OBS桶相同的区域。
2. 在“权限管理”中检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
3. 在左侧导航栏选择“模型训练 > 训练作业”进入训练作业页面，单击“创建训练作业”。
4. 填写创建训练作业相关信息。
   - “创建方式”：选择“自定义算法”。
   - “启动方式”：选择“预置框架”，下拉框中选择PyTorch，pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64。
   - “代码目录”：选择已创建的OBS代码目录路径，例如“/test-modelarts-xx/pytorch/mnist-code/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “启动文件”：选择代码目录下上传的训练脚本“train.py”。
   - “输入”：单击“增加训练输入”，设置训练输入的“参数名称”为“data_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-data/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “输出”：单击“增加训练输出”，设置训练输出的“参数名称”为“train_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-output/”（test-modelarts-xx需替换为您的OBS桶名称）。预下载至本地目录选择“不下载”。

- “资源类型”：选择GPU单卡的规格。如果有免费GPU规格，可以选择免费规格进行训练。
- 其他参数保持默认即可。

说明

本样例代码为单机单卡场景，选择GPU多卡规格会导致训练失败。

5. 单击“提交”，确认训练作业的参数信息，确认无误后单击“确定”。
   页面自动返回“训练作业”列表页，当训练作业状态变为“已完成”时，即完成了模型训练过程。

说明

本案例的训练作业预计运行十分钟。

6. 单击训练作业名称，进入作业详情界面查看训练作业日志信息，观察日志是否有明显的Error信息，如果有则表示训练失败，请根据日志提示定位原因并解决。
7. 在训练详情页左下方单击训练输出路径，如图14-4所示，跳转到OBS目录，查看是否存在model文件夹，且model文件夹中是否有生成训练模型。如果未生成model文件夹或者训练模型，可能是训练输入数据不完整导致，请检查训练数据上传是否完整，并重新训练。

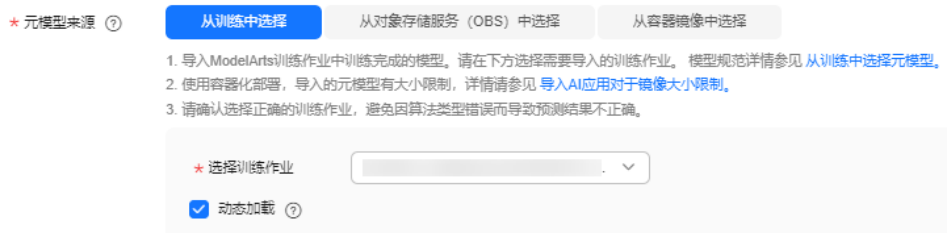
图 14-4 训练输出路径

Step5 推理部署

模型训练完成后，可以创建模型，将模型部署为在线服务。

1. 在ModelArts管理控制台，单击左侧导航栏中的“模型管理（AI应用）”，进入“自定义模型”页面，单击“创建模型”。
2. 在“创建模型”页面，填写相关参数，然后单击“立即创建”。
   在“元模型来源”中，选择“从训练中选择”页签，选择**Step4 创建训练作业**中完成的训练作业，勾选“动态加载”。AI引擎的值是系统自动写入的，无需设置。

图 14-5 设置元模型来源



3. 在模型列表页面，当模型状态变为“正常”时，表示模型创建成功。单击模型操作列的“部署”，弹出“版本列表”，单击操作列“部署>在线服务”，将模型部署为在线服务。

图 14-6 部署在线服务

4. 在“部署”页面，参考下图填写参数，然后根据界面提示完成在线服务创建。本案例适用于CPU规格，节点规格需选择CPU。如果有免费CPU规格，可选择免费规格进行部署（每名用户限部署一个免费的在线服务，如果您已经部署了一个免费在线服务，需要先将其删除才能部署新的免费在线服务）。

图 14-7 部署模型

完成服务部署后，返回在线服务页面列表页，等待服务部署完成，当服务状态显示为“运行中”，表示服务已部署成功。

Step6 预测结果

1. 在“在线服务”页面，单击在线服务名称，进入服务详情页面。
2. 单击“预测”页签，请求类型选择“multipart/form-data”，请求参数填写“image”，单击“上传”按钮上传示例图片，然后单击“预测”。

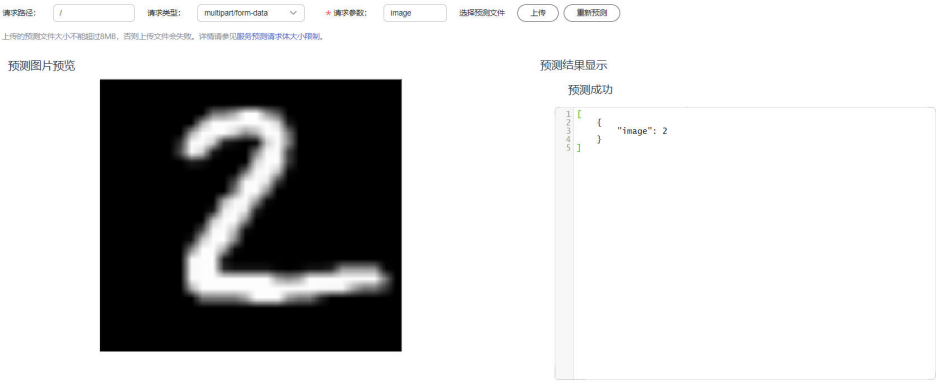
   预测完成后，预测结果显示区域将展示预测结果，根据预测结果内容，可识别出此图片的数字是“2”。

说明

本案例中使用的MNIST是比较简单的用做demo的数据集，配套算法也是比较简单的用于教学的神经网络算法。这样的数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练集中的图片相似（黑底白字）才可能预测准确。

图 14-8 示例图片

图 14-9 预测结果展示



Step7 清除资源

如果不再需要使用此模型及在线服务，建议清除相关资源，避免产生不必要的费用。

- 在“在线服务”页面，“停止”或“删除”刚创建的在线服务。
- 在“自定义模型”页面，“删除”刚创建的模型。
- 在“训练作业”页面，“删除”运行结束的训练作业。
- 进入OBS，删除本示例使用的OBS桶及文件夹，以及文件夹的文件。

常见问题

- 训练作业一直在等待中（排队）？

  训练作业状态一直在等待中状态表示当前所选的资源池规格资源紧张，作业需要进行排队，请耐心等待。请参考**训练作业一直在等待中（排队）？** 。
- 在ModelArts中选择OBS路径时，找不到已创建的OBS桶?

  请确保创建的桶和ModelArts服务在同一区域，详细操作请参考**查看OBS桶与ModelArts是否在同一个区域**。

14.2 基于 ModelArts Standard 运行 GPU 训练作业

14.2.1 在 ModelArts Standard 上运行 GPU 训练作业的场景介绍

不同AI模型训练所需要的数据量和算力不同，在训练时选择合适的存储及训练方案可提升模型训练效率与资源性价比。ModelArts Standard支持单机单卡、单机多卡和多机多卡的训练场景，满足不同AI模型训练的要求。

ModelArts Standard提供了公共资源池和专属资源池，专属资源池不与其他用户共享资源，更加高效。针对企业多用户场景，推荐使用专属资源池开展AI模型训练。

本文提供了端到端案例指导，帮助您快速了解如何在ModelArts Standard上选择合适的训练方案并进行模型训练。

针对不同的数据量和算法情况，推荐以下训练方案：

- 单机单卡：小数据量（1G训练数据）、低算力场景（1卡Vnt1），存储方案推荐使用“OBS的并行文件系统（存放数据和代码）”。
- 单机多卡：中等数据量（50G左右训练数据）、中等算力场景（8卡Vnt1），存储方案推荐使用“SFS（存放数据和代码）”。
- 多机多卡：大数据量（1T训练数据）、高算力场景（4台8卡Vnt1），存储方案推荐使用“SFS（存放数据）+普通OBS桶（存放代码）”，采用分布式训练。

说明

当使用SFS+OBS的存储方案可以实现存储加速，该方案的端到端实践案例请参见**面向AI场景使用OBS+SFS Turbo的存储加速实践**。

表 14-1 不同场景所需服务及购买推荐

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
单机单卡	按需购买（并行文件系统）	×	免费	免费	包月购买	免费	×	按需购买
单机多卡	×	包月购买（HPC型 500G）	免费	免费	包月购买	免费	包月购买（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
多机多卡	按需购买（普通OBS桶）	包月购买（HPC型 500G）	免费	免费	包月购买	免费	包月购买（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

表 14-2 开源数据集训练效率参考

算法及数据	资源规格	Epoch数	预计运行时长（hh:mm:ss）
算法：PyTorch官方针对ImageNet的样例 数据：ImageNet分类数据子集	1机1卡Vnt1	10	0:05:03
算法：YOLOX 数据：COCO2017	1机1卡Vnt1	10	03:33:13
	1机8卡Vnt1	10	01:11:48
	4机8卡Vnt1	10	0:36:17
算法：Swin-Transformer 数据：ImageNet21K	1机1卡Vnt1	10	197:25:03
	1机8卡Vnt1	10	26:10:25
	4机8卡Vnt1	10	07:08:44

表 14-3 训练各步骤性能参考

步骤	说明	预计时长
镜像下载	首次下载镜像的时间（25G）。	8分钟
资源调度	点创建训练作业开始到变成运行中的时间（资源充足、镜像已缓存）。	20秒

步骤	说明	预计时长
训练列表页打开	已有50条训练作业，单击训练模块后的时间。	6秒
日志加载	作业运行中，已经输出1兆的日志文本，单击训练详情页面需要多久加载出日志。	2.5秒
训练详情页	作业运行中，没有用户日志情况下，在ModelArts控制台主页面单击训练详情页面后加载页面内容。	2.5秒
JupyterLab页面	进入JupyterLab页面后加载页面内容。	0.5秒
Notebook列表页	已有50个Notebook实例，在ModelArts控制台主页面单击开发环境后的时间。	4.5秒

说明

镜像下载时间受节点规格、节点硬盘类型（高IO/普通IO）、是否SSD等因素影响，以上数据仅供参考。

14.2.2 在 ModelArts Standard 运行 GPU 训练作业的准备工作

使用ModelArts Standard的专属资源池训练时，需要完成以下准备工作。

购买服务资源

表 14-4 购买服务资源

服务	使用说明	参考文档
弹性文件服务 SFS	弹性文件服务默认为按需计费，即按购买的存储容量和时长收费。您也可以购买包年包月套餐，提前规划资源的使用额度和时长。在欠费时，您需要及时（15天之内）续费以避免您的文件系统资源被清空。 购买的SFS可以用于存储数据和代码。	如何购买弹性文件服务?
容器镜像服务 SWR	容器镜像服务分为企业版和共享版。共享版计费项包括存储空间和流量费用，目前均免费提供给您。企业版支持按需计费模式。 购买的SWR可以用于上传自定义镜像。	上传镜像
对象存储服务 OBS	对象存储服务提供按需计费和包年包月两种计费模式，用户可以根据实际需求购买OBS服务。 OBS服务支持以下两种存储方式，单机单卡场景使用文件系统，多机多卡场景使用普通OBS桶。	• 创建普通OBS桶 • 创建并行文件系统

服务	使用说明	参考文档
虚拟私有云 VPC	虚拟私有云可以为您构建隔离的、用户自主配置和管理的虚拟网络环境。 通过打通专属资源池的VPC，可以方便用户跨VPC使用资源，提升资源利用率。	创建虚拟私有云和子网
弹性云服务器 ECS	如果您需要在服务器上部署相关业务，较之物理服务器，弹性云服务器的创建成本较低，并且可以在几分钟之内快速获得基于云服务平台的弹性云服务器设施，并且这些基础设施是弹性的，可以根据需求伸缩。 购买的ECS服务可以用于挂载SFS Turbo存储。 **说明** 购买时需注意，ECS需要和SFS买到同一个VPC才能挂载SFS存储。	自定义购买ECS
数据加密服务 DEW	在使用Notebook进行代码调试时，如果要开启“SSH远程开发”功能，需要选择密钥对，便于用户登录弹性云服务器时使用密钥对方式进行身份认证，提升通信安全。密钥对可免费创建。	如何创建密钥对?

配置权限

步骤1 配置IAM权限。

1. 使用华为云主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。
2. 创建自定义策略。
   a. 使用华为云主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。
   b. 在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy1”或“Policy2”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

      说明

      创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。**了解更多**。

      - 项目级云服务的自定义策略“Policy1”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "modelarts:pool:create",
                "modelarts:pool:update",
                "modelarts:pool:delete"
```

```
            ],
            "Effect": "Deny"
        },
        {
            "Action": [
                "sfsturbo:*:*",
                "vpc:*:*",
                "dss:*:get",
                "dss:*:list"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "ecs:*:*",
                "evs:*:get",
                "evs:*:list",
                "evs:volumes:create",
                "evs:volumes:delete",
                "evs:volumes:attach",
                "evs:volumes:detach",
                "evs:volumes:manage",
                "evs:volumes:update",
                "evs:volumes:use",
                "evs:volumes:uploadImage",
                "evs:snapshots:create",
                "vpc:*:get",
                "vpc:*:list",
                "vpc:networks:create",
                "vpc:networks:update",
                "vpc:subnets:update",
                "vpc:subnets:create",
                "vpc:ports:*",
                "vpc:routers:get",
                "vpc:routers:update",
                "vpc:securityGroups:*",
                "vpc:securityGroupRules:*",
                "vpc:floatingIps:*",
                "vpc:publicIps:*",
                "ims:images:create",
                "ims:images:delete",
                "ims:images:get",
                "ims:images:list",
                "ims:images:update",
                "ims:images:upload"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "vpc:*:*",
                "ecs:*:get*",
                "ecs:*:list*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:cmk:*",
                "kms:dek:*",
                "kms:grant:*",
                "kms:cmkTag:*",
                "kms:partition:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

- 全局级云服务的自定义策略“Policy2”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
                "obs:object:ListMultipartUploadParts",
                "obs:object:AbortMultipartUpload",
                "obs:object:GetObjectAcl",
                "obs:object:GetObjectVersionAcl"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. 将自定义策略授权给开发者用户组user_group。
   a. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1”、“Policy2”、“SWR Admin”。单击“下一步”。

      说明

      SWR的权限有SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess。但SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess仅限容器镜像服务企业版使用，目前企业版已暂停公测。非企业版用户暂不支持使用此权限。因此需要在此勾选“SWR Admin” 策略。

   b. 设置最小授权范围，选择授权范围方案为“所有资源”，单击“确定”。

更多权限管理的信息请参见**ModelArts权限管理基本概念**。

步骤2 配置ModelArts委托权限。

给用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

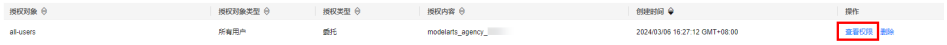
1. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。
2. 在弹出的“添加授权”窗口中，选择：
   - **授权对象类型**：**所有用户**
   - **委托选择**：**新增委托**
   - **权限配置**：**普通用户**

   选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 14-10 配置委托访问授权

3. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

图 14-11 查看委托配置信息



步骤3 配置SWR组织权限。

IAM用户创建后，需要管理员在组织中为用户添加授权，使IAM用户对组织内所有镜像享有读取/编辑/管理的权限。

只有具备“管理”权限的账号和IAM用户才能添加授权。

1. 登录容器镜像服务控制台。
2. 在左侧菜单栏选择“组织管理”，单击组织名称。
3. 在“用户”页签下单击“添加授权”，在弹出的窗口中为IAM用户选择权限，然后单击“确定”。

SWR授权管理详情可参考**授权管理**。

说明

如果给子账号的SWR授权不是SWR Admin权限，则需要继续配置SWR组织权限。

步骤4 测试用户权限。

由于权限配置需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子账号登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 验证ModelArts权限。
   a. 在左上角的服务列表中，选择ModelArts服务，进入ModelArts管理控制台。
   b. 在ModelArts管理控制台，可正常创建Notebook、训练作业、注册镜像。
3. 验证SFS权限。
   a. 在左上角的服务列表中，选择SFS服务，进入SFS管理控制台。
   b. 在SFS管理控制台的SFS Turbo中单击右上角的“创建文件系统”，如果能正常打开页面，表示当前用户具备SFS的操作权限。

4. 验证ECS权限。
   a. 在左上角的服务列表中，选择ECS服务，进入ECS管理控制台。
   b. 在ECS管理控制台，单击右上角的“购买弹性云服务器”，如果能正常打开页面，表示当前用户具备ECS的操作权限。
5. 验证VPC权限。
   a. 在左上角的服务列表中，选择VPC服务，进入VPC管理控制台。
   b. 在VPC管理控制台，单击右上角的“创建虚拟私有云”，如果能正常打开页面，表示当前用户具备VPC的操作权限。
6. 验证DEW权限。
   a. 在左上角的服务列表中，选择DEW服务，进入DEW管理控制台。
   b. 在DEW管理控制台，选择“密钥对管理 > 私有密钥对”，单击“创建密钥对”，如果能正常打开页面，表示当前用户具备DEW的操作权限。
7. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
8. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
   c. 单击右上角的“上传镜像”，如果能看到授权的组织，表示当前用户具备SWR组织权限。

----结束

创建专属资源池

ModelArts提供独享的计算资源，可用于Notebook、训练作业、部署模型。专属资源池不与其他用户共享，更加高效。在使用专属资源池之前，您需要先创建一个专属资源池，操作指导请参考**创建Standard专属资源池**。

- 配置“网络”时需要选择已打通VPC的网络。如果需要新建网络和打通VPC可以参考**配置Standard专属资源池可访问公网**。
- “规格类型”和“节点数量”根据训练计划使用的资源选择。

在 ECS 服务器挂载 SFS Turbo 存储

在ECS服务器挂载SFS Turbo存储后，支持将训练所需的数据通过ECS上传至SFS Turbo。

1. 检查云服务环境。
   - ECS服务器和SFS的共享硬盘在相同的VPC或者对应VPC能够互联。
   - ECS服务器基础镜像用的是Ubuntu 18.04。
   - ECS服务器和SFS Turbo在同一子网中。
2. 在ECS服务器中设置华为云镜像源。

```
sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
```

3. 安装NFS客户端，挂载对应盘。
```
sudo apt-get update
sudo apt-get install nfs-common
```
4. 获取SFS Turbo的挂载命令。
   a. 进入弹性文件服务SFS管理控制台。
   b. 选择“SFS Turbo”进入文件系统列表，单击文件系统名称，进入详情页面。
   c. 在“基本信息”页签获取并记录“Linux挂载命令”。
5. 在ECS服务器中挂载NFS存储。

   确认对应目录存在后，输入对应指令，命令如下。
```
mkdir -p /mnt/sfs_turbo
mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
```

在 ECS 中设置 ModelArts 用户可读权限

在ModelArts训练平台使用自定义镜像时，默认用户为ma-user、默认用户组为ma-group。如果在训练时调用ECS中的文件，需要修改文件权限改为ma-user可读，否则会出现Permission denied错误。

1. 在Terminal中执行以下命令，在ECS中提前创建好ma-user和ma-group。
```
default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
    groupdel -f ${default_group}; \
    groupadd -g 100 ma-group; \
fi && \
if [ -z ${default_group} ]; then \
    groupadd -g 100 ma-group; \
fi && \
if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
    userdel -r ${default_user}; \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
if [ -z ${default_user} ]; then \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
# set bash as default
rm /bin/sh && ln -s /bin/bash /bin/sh
```
2. 执行以下命令，查看创建的用户信息。
```
id ma-user
```
   如果出现以下信息则表示创建成功。
```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

安装和配置 OBS 命令行工具

obsutil是用于访问、管理对象存储服务OBS的命令行工具，使用该工具可以对OBS进行常用的配置管理操作，如创建桶、上传文件/文件夹、下载文件/文件夹、删除文件/文件夹等。

obsutil安装和配置的具体操作指导请参见**obsutils快速入门**。

须知

操作命令中的AK/SK和Endpoint要替换为用户实际获取的AK/SK和Endpoint。

（可选）工作空间配置

ModelArts支持设置子账号的细粒度权限、不同工作空间之间资源隔离。ModelArts工作空间帮您实现项目资源隔离、多项目分开结算等功能。

如果您开通了企业项目管理服务的权限，可以在创建工作空间的时候绑定企业项目ID，并在企业项目下添加用户组，为不同的用户组设置细粒度权限供组里的用户使用。

如果您未开通企业项目管理服务的权限，也可以在ModelArts创建自己独立的工作空间，但是无法使用跟企业项目相关的功能。

说明

工作空间为白名单功能，使用该功能需要提工单申请开通。

14.2.3 在 ModelArts Standard 上运行 GPU 单机单卡训练作业

操作流程

1. 准备工作
   a. **购买服务资源**（OBS和SWR）
   b. **配置权限**
   c. **创建专属资源池**（不需要打通VPC）
   d. **安装和配置OBS命令行工具**
   e. **（可选）工作空间配置**
2. 模型训练
   a. **本地构建镜像及调试**
   b. **上传镜像**
   c. **上传数据和算法到OBS**
   d. **使用Notebook进行代码调试**
   e. **创建单机单卡训练作业**
   f. **监控资源**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。
   a. 启动线下的容器镜像：
      ```
      # run on terminal
      docker run -ti ${your_image:tag}
      ```
   b. 在容器中输入如下命令，得到pytorch.tar.gz：
      ```
      # run on container

      # 基于想要迁移的base环境创建一个名为pytorch的conda环境
      conda create --name pytorch --clone base

      pip install conda-pack

      #将pytorch env打包生成pytorch.tar.gz
      conda pack -n pytorch -o pytorch.tar.gz
      ```
   c. 将打包好的压缩包传到本地：
      ```
      # run on terminal
      docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
      ```
   d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。
2. 构建新镜像。

   基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

   构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

   说明

   - 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
   - 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。
3. 构建参考样例

   Dockerfile样例：

   ```
   FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

   USER root

   # section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
   RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
       default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
       if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
           groupdel -f ${default_group}; \
           groupadd -g 100 ma-group; \
       fi && \
       if [ -z ${default_group} ]; then \
           groupadd -g 100 ma-group; \
       fi && \
       if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
           userdel -r ${default_user}; \
           useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
           chmod -R 750 /home/ma-user; \
       fi && \
       if [ -z ${default_user} ]; then \
           useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
           chmod -R 750 /home/ma-user; \
       fi && \
       # set bash as default
       rm /bin/sh && ln -s /bin/bash /bin/sh

   # section2: config apt source and install tools needed.
   ```

```
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

   如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

   训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

c. 测试训练启动脚本。

i. 优先使用手工进行数据复制的工作并验证

一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

docker和linux的文件交互命令如下：

```
docker cp data/ 39c9ceedb1f6:/cache/
```

数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

```
cd /cache/code/
python start_train.py
```

如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

ii. 预置脚本测试整体流程

一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**基于ModelArts Standard运行GPU训练作业**。

如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

iii. 针对专属池场景

由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

可以通过命令查看容器输出到stdout的所有日志：

```
docker logs -f 39c9ceedb1f6
```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。
      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。
   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]
   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```
3. 上传镜像至镜像仓库。
   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]
   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据和算法到 OBS

- 已经在OBS上创建好并行文件系统，请参见**创建并行文件系统**。
- 已经安装和配置obsutil，请参见**安装和配置OBS命令行工具**。

- OBS和训练容器间的数据传输原理可以参考**基于ModelArts Standard运行GPU训练作业**。

步骤1 准备数据

1. 单击下载**动物数据集**至本地，并解压。
2. 通过obsutil将数据集上传至OBS桶中。

```
./obsutil cp ./dog_cat_1w obs://${your_obs_buck}/demo/ -f -r
```

说明

OBS支持多种文件上传方式，当文件少于100个时，可以在OBS Console中上传，当文件大于100个时，推荐使用工具，推荐**OBS Browser+**（win）、**obsutil**（linux）。上述例子为obsutil使用方法。

步骤2 准备算法

main.py文件内容如下，并将其上传至OBS桶的demo文件夹中：

```
import argparse
import os
import random
import shutil
import time
import warnings
from enum import Enum
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.distributed as dist
import torch.optim
from torch.optim.lr_scheduler import StepLR
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))
parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
parser.add_argument('data', metavar='DIR', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                        'batch size of all GPUs on the current node when '
                        'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
```

```
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int,
                    help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--dist-backend', default='nccl', type=str,
                    help='distributed backend')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing-distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                         'N processes per node, which has N GPUs. This is the '
                         'fastest way to use PyTorch for either single node or '
                         'multi node data parallel training')
best_acc1 = 0


def main():
    args = parser.parse_args()
    if args.seed is not None:
        random.seed(args.seed)
        torch.manual_seed(args.seed)
        cudnn.deterministic = True
        warnings.warn('You have chosen to seed training. '
                      'This will turn on the CUDNN deterministic setting, '
                      'which can slow down your training considerably! '
                      'You may see unexpected behavior when restarting '
                      'from checkpoints.')
    if args.gpu is not None:
        warnings.warn('You have chosen a specific GPU. This will completely '
                      'disable data parallelism.')
    if args.dist_url == "env://" and args.world_size == -1:
        args.world_size = int(os.environ["WORLD_SIZE"])
    args.distributed = args.world_size > 1 or args.multiprocessing_distributed
    ngpus_per_node = torch.cuda.device_count()
    if args.multiprocessing_distributed:
        # Since we have ngpus_per_node processes per node, the total world_size
        # needs to be adjusted accordingly
        args.world_size = ngpus_per_node * args.world_size
        # Use torch.multiprocessing.spawn to launch distributed processes: the
        # main_worker process function
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
    else:
        # Simply call main_worker function
        main_worker(args.gpu, ngpus_per_node, args)
def main_worker(gpu, ngpus_per_node, args):
    global best_acc1
    args.gpu = gpu
    if args.gpu is not None:
        print("Use GPU: {} for training".format(args.gpu))
    if args.distributed:
        if args.dist_url == "env://" and args.rank == -1:
            args.rank = int(os.environ["RANK"])
        if args.multiprocessing_distributed:
            # For multiprocessing distributed training, rank needs to be the
            # global rank among all the processes
            args.rank = args.rank * ngpus_per_node + gpu
        dist.init_process_group(backend=args.dist_backend, init_method=args.dist_url,
```

```
                            world_size=args.world_size, rank=args.rank)
# create model
if args.pretrained:
    print("=> using pre-trained model '{}'".format(args.arch))
    model = models.__dict__[args.arch](pretrained=True)
else:
    print("=> creating model '{}'".format(args.arch))
    model = models.__dict__[args.arch]()
if not torch.cuda.is_available():
    print('using CPU, this will be slow')
elif args.distributed:
    # For multiprocessing distributed, DistributedDataParallel constructor
    # should always set the single device scope, otherwise,
    # DistributedDataParallel will use all available devices.
    if args.gpu is not None:
        torch.cuda.set_device(args.gpu)
        model.cuda(args.gpu)
        # When using a single GPU per process and per
        # DistributedDataParallel, we need to divide the batch size
        # ourselves based on the total number of GPUs of the current node.
        args.batch_size = int(args.batch_size / ngpus_per_node)
        args.workers = int((args.workers + ngpus_per_node - 1) / ngpus_per_node)
        model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
    else:
        model.cuda()
        # DistributedDataParallel will divide and allocate batch_size to all
        # available GPUs if device_ids are not set
        model = torch.nn.parallel.DistributedDataParallel(model)
elif args.gpu is not None:
    torch.cuda.set_device(args.gpu)
    model = model.cuda(args.gpu)
else:
    # DataParallel will divide and allocate batch_size to all available GPUs
    if args.arch.startswith('alexnet') or args.arch.startswith('vgg'):
        model.features = torch.nn.DataParallel(model.features)
        model.cuda()
    else:
        model = torch.nn.DataParallel(model).cuda()
# define loss function (criterion), optimizer, and learning rate scheduler
criterion = nn.CrossEntropyLoss().cuda(args.gpu)
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
"""Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
# optionally resume from a checkpoint
if args.resume:
    if os.path.isfile(args.resume):
        print("=> loading checkpoint '{}'".format(args.resume))
        if args.gpu is None:
            checkpoint = torch.load(args.resume)
        else:
            # Map model to be loaded to specified single gpu.
            loc = 'cuda:{}'.format(args.gpu)
            checkpoint = torch.load(args.resume, map_location=loc)
        args.start_epoch = checkpoint['epoch']
        best_acc1 = checkpoint['best_acc1']
        if args.gpu is not None:
            # best_acc1 may be from a checkpoint from a different GPU
            best_acc1 = best_acc1.to(args.gpu)
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        scheduler.load_state_dict(checkpoint['scheduler'])
        print("=> loaded checkpoint '{}' (epoch {})"
              .format(args.resume, checkpoint['epoch']))
    else:
        print("=> no checkpoint found at '{}'".format(args.resume))
cudnn.benchmark = True
```

```
    # Data loading code
    traindir = os.path.join(args.data, 'train')
    valdir = os.path.join(args.data, 'val')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    train_dataset = datasets.ImageFolder(
        traindir,
        transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            normalize,
        ]))
    if args.distributed:
        train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
    else:
        train_sampler = None

    train_loader = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size, shuffle=(train_sampler is None),
        num_workers=args.workers, pin_memory=True, sampler=train_sampler)
    val_loader = torch.utils.data.DataLoader(
        datasets.ImageFolder(valdir, transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize,
        ])),
        batch_size=args.batch_size, shuffle=False,
        num_workers=args.workers, pin_memory=True)
    if args.evaluate:
        validate(val_loader, model, criterion, args)
        return

    for epoch in range(args.start_epoch, args.epochs):
        if args.distributed:
            train_sampler.set_epoch(epoch)
        # train for one epoch
        train(train_loader, model, criterion, optimizer, epoch, args)
        # evaluate on validation set
        acc1 = validate(val_loader, model, criterion, args)
        scheduler.step()
        # remember best acc@1 and save checkpoint
        is_best = acc1 > best_acc1
        best_acc1 = max(acc1, best_acc1)
        if not args.multiprocessing_distributed or (args.multiprocessing_distributed
                                                    and args.rank % ngpus_per_node == 0):
            save_checkpoint({
                'epoch': epoch + 1,
                'arch': args.arch,
                'state_dict': model.state_dict(),
                'best_acc1': best_acc1,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict()
            }, is_best)
def train(train_loader, model, criterion, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses, top1, top5],
        prefix="Epoch: [{}]".format(epoch))
    # switch to train mode
    model.train()
    end = time.time()
    for i, (images, target) in enumerate(train_loader):
```

```
        # measure data loading time
        data_time.update(time.time() - end)
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        if torch.cuda.is_available():
            target = target.cuda(args.gpu, non_blocking=True)
        # compute output
        output = model(images)
        loss = criterion(output, target)
        # measure accuracy and record loss
        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        top5.update(acc5[0], images.size(0))
        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if i % args.print_freq == 0:
            progress.display(i)
def validate(val_loader, model, criterion, args):
    batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
    losses = AverageMeter('Loss', ':.4e', Summary.NONE)
    top1 = AverageMeter('Acc@1', ':6.2f', Summary.AVERAGE)
    top5 = AverageMeter('Acc@5', ':6.2f', Summary.AVERAGE)
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, losses, top1, top5],
        prefix='Test: ')
    # switch to evaluate mode
    model.eval()
    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            if args.gpu is not None:
                images = images.cuda(args.gpu, non_blocking=True)
            if torch.cuda.is_available():
                target = target.cuda(args.gpu, non_blocking=True)
            # compute output
            output = model(images)
            loss = criterion(output, target)
            # measure accuracy and record loss
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            losses.update(loss.item(), images.size(0))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))
            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()
            if i % args.print_freq == 0:
                progress.display(i)
        progress.display_summary()
    return top1.avg
def save_checkpoint(state, is_best, filename='checkpoint.pth.tar'):
    torch.save(state, filename)
    if is_best:
        shutil.copyfile(filename, 'model_best.pth.tar')
class Summary(Enum):
    NONE = 0
    AVERAGE = 1
    SUM = 2
    COUNT = 3


class AverageMeter(object):
    """Computes and stores the average and current value"""
```

```
    def __init__(self, name, fmt=':f', summary_type=Summary.AVERAGE):
        self.name = name
        self.fmt = fmt
        self.summary_type = summary_type
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)
    def summary(self):
        fmtstr = ''
        if self.summary_type is Summary.NONE:
            fmtstr = ''
        elif self.summary_type is Summary.AVERAGE:
            fmtstr = '{name} {avg:.3f}'
        elif self.summary_type is Summary.SUM:
            fmtstr = '{name} {sum:.3f}'
        elif self.summary_type is Summary.COUNT:
            fmtstr = '{name} {count:.3f}'
        else:
            raise ValueError('invalid summary type %r' % self.summary_type)
        return fmtstr.format(**self.__dict__)
class ProgressMeter(object):
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix
    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))
    def display_summary(self):
        entries = [" *"]
        entries += [meter.summary() for meter in self.meters]
        print(' '.join(entries))
    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """Computes the accuracy over the k top predictions for the specified values of k"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))
        res = []
        for k in topk:
            correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res
```

```
if __name__ == '__main__':
    main()
```

----结束

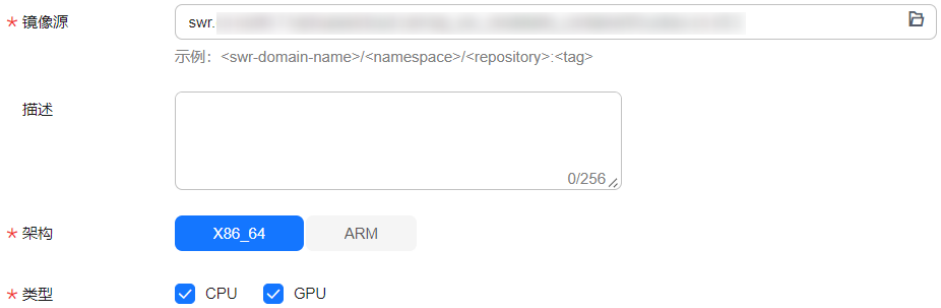
使用 Notebook 进行代码调试

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产生不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

操作步骤如下：

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如**图14-12**所示。

   图 14-12 注册镜像

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建Notebook”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表14-5**。

      表 14-5 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。

参数名称	说明
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择1 GPU规格。
- 存储配置：选择“云硬盘EVS”作为存储位置。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接Notebook方式介绍**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。

如果创建Notebook启动失败，建议参考**调试要点**进行检查。

6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 挂载OBS并行文件系统：在Notebook实例详情页面，选择“存储配置”页签，单击“添加数据存储”，设置挂载参数。

a. 设置本地挂载目录，在“/data/”目录下输入一个文件夹名称，例如：demo。挂载时，后台自动会在Notebook容器“的/data/”目录下创建该文件夹，用来挂载OBS文件系统。

b. 选择存放OBS并行文件系统下的文件夹，单击“确定”。

8. 挂载成功后，可以在Notebook实例详情页查看到挂载结果。
9. 代码调试。

打开Notebook，打开Terminal，进入步骤7中挂载的目录。

```
cd /data/demo
```

执行训练命令：

```
/home/ma-user/anaconda3/envs/pytorch/bin/python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
```

告警"RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.12) doesn't match a supported version!"不影响训练，可忽略。

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

创建单机单卡训练作业

说明

针对专属池场景，应注意挂载的目录设置和调试时一致。

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd ${MA_JOB_DIR}/demo && python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
     ```
     此处的“demo”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择单GPU规格。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

监控资源

用户可以通过资源占用情况窗口查看计算节点的资源使用情况，最多可显示最近三天的数据。在资源占用情况窗口打开时，会定期向后台获取最新的资源使用率数据并刷新。

操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

操作二：单击图例“cpuUsage”、“gpuMemUsage”、“gpuUtil”、“memUsage”“npuMemUsage”、“npuUtil”、可以添加或取消对应参数的使用情况图。

操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

表 14-6 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况。

参数	说明
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况。

14.2.4 在 ModelArts Standard 上运行 GPU 单机多卡训练作业

操作流程

1. 准备工作：
   a. **购买服务资源**（VPC、SFS、SWR和ECS）
   b. **配置权限**
   c. **创建专属资源池**（打通VPC）
   d. **在ECS服务器挂载SFS Turbo存储**
   e. **在ECS中设置ModelArts用户可读权限**
   f. **安装和配置OBS命令行工具**
   g. **（可选）工作空间配置**
2. 模型训练：
   a. **本地构建镜像及调试**
   b. **上传镜像**
   c. **上传数据和算法至SFS（首次使用时需要）**
   d. **使用Notebook进行代码调试**
   e. **创建单机多卡训练作业**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。
   a. 启动线下的容器镜像：

```
# run on terminal
docker run -ti ${your_image:tag}
```

b. 在容器中输入如下命令，得到pytorch.tar.gz：

```
# run on container

# 基于想要迁移的base环境创建一个名为pytorch的conda环境
conda create --name pytorch --clone base

pip install conda-pack

#将pytorch env打包生成pytorch.tar.gz
conda pack -n pytorch -o pytorch.tar.gz
```

c. 将打包好的压缩包传到本地：

```
# run on terminal
docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
```

d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。

2. 构建新镜像。

基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

说明

- 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
- 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。

3. 构建参考样例

Dockerfile样例：

```
FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

USER root

# section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
```

```
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

   如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

   训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

c. 测试训练启动脚本。

   i. 优先使用手工进行数据复制的工作并验证

      一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/

cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

docker和linux的文件交互命令如下：

```
docker cp data/ 39c9ceedb1f6:/cache/
```

数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

```
cd /cache/code/
python start_train.py
```

如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

ii. 预置脚本测试整体流程

一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**基于ModelArts Standard运行GPU训练作业**。

如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

iii. 针对专属池场景

由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

可以通过命令查看容器输出到stdout的所有日志：

```
docker logs -f 39c9ceedb1f6
```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。
      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。
   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]
   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```
3. 上传镜像至镜像仓库。
   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]
   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据和算法至 SFS

- ECS服务器已挂载SFS，请参考**在ECS服务器挂载SFS Turbo存储**。
- 已经在ECS中设置权限，请参考**在ECS中设置ModelArts用户可读权限**。
- 已经安装和配置obsutil，请参见**安装和配置OBS命令行工具**。

步骤1 准备数据

1. 登录coco数据集下载官网地址：**https://cocodataset.org/#download**
2. 下载coco2017数据集的Train（18GB）、Val images（1GB）、Train/Val annotations（241MB），分别解压后并放入coco文件夹中。
3. 下载完成后，将数据上传至SFS相应目录中。由于数据集过大，推荐先通过obsutil工具将数据集传到OBS桶后，再将数据集迁移至SFS。

   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

   ```
   # 将本地数据传至OBS中
   # ./obsutil cp ${数据集所在的本地文件夹路径} ${存放数据集的obs文件夹路径} -f -r
   # 例如
   ./obsutil cp ./coco obs://your_bucket/ -f -r
   ```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

   ```
   # 将OBS数据传至SFS中
   # ./obsutil cp ${数据集所在的obs文件夹路径} ${SFS文件夹路径} -f -r
   # 例如
   ./obsutil cp obs://your_bucket/coco/ /mnt/sfs_turbo/ -f -r
   ```

   /mnt/sfs_turbo/coco文件夹内目录结构如下：

   ```
   coco
   |---annotations
   |---train2017
   |---val2017
   ```

   更多obsutil的操作，可参考**obsutil简介**。

   c. 将文件设置归属为ma-user：

   ```
   chown -R ma-user:ma-group coco
   ```

步骤2 准备算法

1. 下载YOLOX代码。代码仓地址：**https://github.com/Megvii-BaseDetection/YOLOX.git**。

   ```
   git clone https://github.com/Megvii-BaseDetection/YOLOX.git
   cd YOLOX
   git checkout 4f8f1d79c8b8e530495b5f183280bab99869e845
   ```

2. 修改“requirements.txt”中的onnx版本，改为“onnx>=1.12.0”。
3. 将“yolox/data/datasets/coco.py”第59行的“data_dir = os.path.join(get_yolox_datadir(), "COCO")”改为“data_dir = '/home/ma-user/coco'”。

   ```
   # data_dir = os.path.join(get_yolox_datadir(), "COCO")
   data_dir = '/home/ma-user/coco'
   ```

4. 在“tools/train.py”的第13行前加两句代码。

   ```
   # 加上这两句代码，防止运行时找不到yolox module
   import sys
   sys.path.append(os.getcwd())

   # line13
   from yolox.core import launch
   from yolox.exp import Exp, get_exp
   ```

5. 将“yolox/layers/jit_ops.py”第122行的“fast_cocoeval”改为“fast_coco_eval_api”。

   ```
   # def __init__(self, name="fast_cocoeval"):
   def __init__(self, name="fast_coco_eval_api"):
   ```

6. 将“yolox\evaluators\coco_evaluator.py”第294行的“from yolox.layers import COCOeval_opt as COCOeval”改为“from pycocotools.cocoeval import COCOeval”。

   ```
   try:
     # from yolox.layers import COCOeval_opt as COCOeval
     from pycocotools.cocoeval import COCOeval
   except ImportError:
     from pycocotools.cocoeval import COCOeval
   ```



```
  logger.warning("Use standard COCOeval.")
```

7. 在tools目录下新建一个“run.sh”作为启动脚本，“run.sh”内容可参考：

```
#!/usr/bin/env sh
set -x
set -o pipefail

export NCCL_DEBUG=INFO

DEFAULT_ONE_GPU_BATCH_SIZE=32
BATCH_SIZE=$((${MA_NUM_GPUS:-8} * ${VC_WORKER_NUM:-1} * $
{DEFAULT_ONE_GPU_BATCH_SIZE}))
if [ ${VC_WORKER_HOSTS} ];then
    YOLOX_DIST_URL=tcp://$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1):6666
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0} \
                            --dist-url ${YOLOX_DIST_URL}
else
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0}
fi
```

说明

部分环境变量在Notebook环境中不存在，因此需要提供默认值。

8. 将代码放到OBS上，然后通过OBS将代码传至SFS相应目录中。

   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

   ```
   # 将本地代码传至OBS中
   ./obsutil cp ./YOLOX obs://your_bucket/ -f -r
   ```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

   ```
   # 将OBS的代码传到SFS中
   ./obsutil cp obs://your_bucket/YOLOX/ /mnt/sfs_turbo/code/ -f -r
   ```

说明

本案例中以obsutils方式上传文件，除此之外也可通过SCP方式上传文件，具体操作步骤可参考**本地Linux主机使用SCP上传文件到Linux云服务器**。

9. 在SFS中将文件设置归属为ma-user。

```
chown -R ma-user:ma-group YOLOX
```

10. 执行以下命令，去除Shell脚本的\r字符。

```
cd YOLOX
sed -i 's/\r//' run.sh
```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

----结束

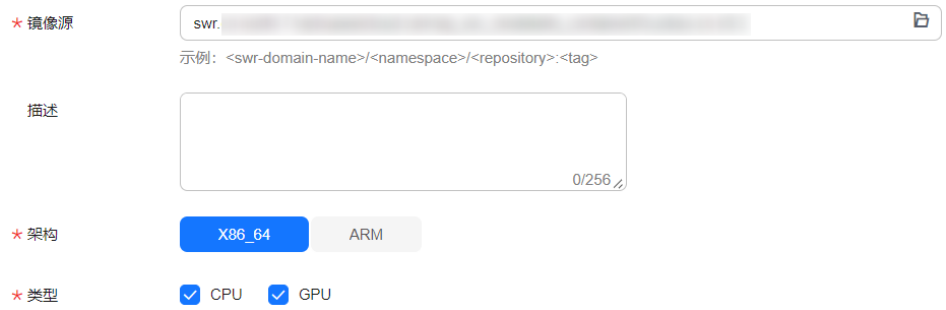
使用 Notebook 进行代码调试

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产生不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

操作步骤如下：

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如**图14-13**所示。

   图 14-13 注册镜像

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建Notebook”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表14-7**。

      表 14-7 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。

参数名称	说明
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择8卡GPU规格，“run.sh”文件中默认MA_NUM_GPUS为8卡，因此选择notebook规格时需要与MA_NUM_GPUS默认值相同。
- 存储配置：选择“弹性文件服务SFS”作为存储位置。子目录挂载可不填写，如果需挂载SFS指定目录，则在子目录挂载处填写具体路径。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接Notebook方式介绍**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 在Notebook中打开Terminal，输入启动命令调试代码。

```
# 建立数据集软链接
# ln -s /home/ma-user/work/${coco数据集在SFS上的路径} /home/ma-user/coco
# 进入到对应目录
# cd /home/ma-user/work/${YOLOX在SFS上的路径}
# 安装环境并执行脚本
# /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh

# 例如
ln -s /home/ma-user/work/coco /home/ma-user/coco
cd /home/ma-user/work/code/YOLOX/
/home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
```

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

创建单机多卡训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。

3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     ln -s /home/ma-user/work/coco /home/ma-user/coco && cd /home/ma-user/work/code/YOLOX/ && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择8卡GPU规格。
   - 计算节点：1。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，因此云上挂载路径需要填写为“/home/ma-user/work”。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

14.2.5 在 ModelArts Standard 上运行 GPU 多机多卡训练作业

操作流程

1. 准备工作：
   a. **购买服务资源**（VPC/SFS/OBS/SWR/ECS）
   b. **配置权限**
   c. **创建专属资源池**（打通VPC）
   d. **ECS服务器挂载SFS Turbo存储**
   e. **在ECS中设置ModelArts用户可读权限**
   f. **安装和配置OBS命令行工具**
   g. **（可选）工作空间配置**
2. 模型训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据至OBS（首次使用时需要）**
   d. **上传算法至SFS**
   e. **使用Notebook进行代码调试**
   f. **创建多机多卡训练作业**

本地构建镜像及调试

本节通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境依赖。

说明

- 容器镜像的大小建议小于15G，详细的自定义镜像规范要求请参见**训练作业自定义镜像规范**。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 如果训练数据和代码经常变动，则不建议把数据、代码放到容器镜像里，避免频繁地构建容器镜像。
- 容器已经能满足隔离需求，不建议在容器内再创建多个conda env。

1. 导出conda环境。
   a. 启动线下的容器镜像：
   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```
   b. 在容器中输入如下命令，得到pytorch.tar.gz：
   ```
   # run on container

   # 基于想要迁移的base环境创建一个名为pytorch的conda环境
   conda create --name pytorch --clone base

   pip install conda-pack

   #将pytorch env打包生成pytorch.tar.gz
   conda pack -n pytorch -o pytorch.tar.gz
   ```
   c. 将打包好的压缩包传到本地：
   ```
   # run on terminal
   docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
   ```
   d. 将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时使用**wget**命令获取、解压、清理。
2. 构建新镜像。

   基础镜像一般选用“ubuntu 18.04”的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

   构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

   说明

   - 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
   - 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。
3. 构建参考样例

   Dockerfile样例：
   ```
   FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

   USER root

   # section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there already exists 1000:100 but not ma-user:ma-group, below code will remove it
   RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
   ```



```
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
```

```
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. 调试镜像

   说明

   建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

   a. 确认对应的脚本、代码、流程在linux服务器上运行正常。

      如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

   b. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

      训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

   c. 测试训练启动脚本。

      i. 优先使用手工进行数据复制的工作并验证

         一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

         docker和linux的文件交互命令如下：

         ```
         docker cp data/ 39c9ceedb1f6:/cache/
         ```

         数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

         ```
         cd /cache/code/
         python start_train.py
         ```

         如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

      ii. 预制脚本测试整体流程

         一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**基于ModelArts Standard运行GPU训练作业**。

         如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

      iii. 针对专属池场景

         由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

         可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

         调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

         ```
         docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
         ```

         上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

   d. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

      可以通过命令查看容器输出到stdout的所有日志：

      ```
      docker logs -f 39c9ceedb1f6
      ```

一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

e. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

上传镜像

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

说明

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。

      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

示例：

```
docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

3. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

上传数据至 OBS

- 已经在OBS上创建好普通OBS桶，请参见**创建普通OBS桶**。
- 已经安装obsutil，请参考**安装和配置OBS命令行工具**。
- OBS和训练容器间的数据传输原理可以参考**基于ModelArts Standard运行GPU训练作业**。

1. 登录Imagenet数据集下载官网地址，下载Imagenet21k数据集：**http://image-net.org/**
2. 下载格式转换后的annotation文件：**ILSVRC2021winner21k_whole_map_train.txt**和**ILSVRC2021winner21k_whole_map_val.txt**。
3. 下载完成后将上述3个文件数据上传至OBS桶中的imagenet21k_whole文件夹中。上传方法请参考**上传数据和算法到OBS**。

上传算法到 SFS

1. 下载Swin-Transformer代码。
   ```
   git clone --recursive https://github.com/microsoft/Swin-Transformer.git
   ```
2. 修改lr_scheduler.py文件，把第27行：t_mul=1. 注释掉。
3. 修改data文件夹下imagenet22k_dataset.py，把第28行：print("ERROR IMG LOADED: ", path) 注释掉。
4. 修改data文件夹下的build.py文件，把第112行：prefix = 'ILSVRC2011fall_whole'，改为prefix = 'ILSVRC2021winner21k_whole'。
5. 在Swin-Transformer目录下创建requirements.txt指定python依赖库：
   ```
   # requirements.txt内容如下

   timm==0.4.12
   termcolor==1.1.0
   yacs==0.1.8
   ```
6. 准备run.sh文件中所需要的obs文件路径。

   a. 准备imagenet数据集的分享链接。

      勾选要分享的imagenet21k_whole数据集文件夹，单击分享按钮，选择分享链接有效期，自定义提取码，例如123456，单击“复制链接”，记录该链接。

   b. 准备“obsutil_linux_amd64.tar.gz”的分享链接。

      参考**下载和安装obsutil**下载“obsutil_linux_amd64.tar.gz”，将其上传至OBS桶中，设置为公共读。单击属性，单击复制链接。

      链接样例如下：

```
https://${bucketname_name}.obs.cn-north-4.myhuaweicloud.com/${folders_name}/pytorch.tar.gz
```

7. 在Swin-Transformer目录下，创建运行脚本run.sh。

   说明

   - 脚本中的"SRC_DATA_PATH=${imagenet数据集在obs中分享链接}"，需要替换为上一步中的imagenet21k_whole文件夹分享链接。
   - 脚本中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz"，需要替换为上一步中obsutil_linux_amd64.tar.gz在OBS上的路径（需将文件设置为公共读）。

   单机单卡运行脚本：

   ```
   # 在代码主目录下创建一个run.sh，内容如下

   #!/bin/bash

   # 从obs中下载数据到本地SSD盘
   DIS_DATA_PATH=/cache
   SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
   OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
   mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -

   IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
   MASTER_PORT="6061"

   /home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nproc_per_node=1 --master_addr localhost --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml --local_rank 0
   ```

   多机多卡运行脚本：

   ```
   # 创建run.sh

   #!/bin/bash

   # 从obs中下载数据到本地SSD盘
   DIS_DATA_PATH=/cache
   SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
   OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
   mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -
   IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
   MASTER_ADDR=$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1)

   MASTER_PORT="6060"
   NNODES="$VC_WORKER_NUM"
   NODE_RANK="$VC_TASK_INDEX"
   NGPUS_PER_NODE="$MA_NUM_GPUS"

   /home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nnodes=$NNODES --node_rank=$NODE_RANK --nproc_per_node=$NGPUS_PER_NODE --master_addr $MASTER_ADDR --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml
   ```

   说明

   - 推荐先使用单机单卡运行脚本，待正常运行后再改用多机多卡运行脚本。
   - 多机多卡run.sh中的“VC_WORKER_HOSTS”、“VC_WORKER_NUM”、“VC_TASK_INDEX”、“MA_NUM_GPUS”为ModelArts训练容器中预置的环境变量。训练容器环境变量详细介绍可参考**查看训练容器环境变量**。

8. 通过obsutils，将代码文件夹放到OBS上，然后通过OBS将代码传至SFS相应目录中。

9. 在SFS中将代码文件Swin-Transformer-main设置归属为ma-user。
```
chown -R ma-user:ma-group Swin-Transformer
```
10. 执行以下命令，去除Shell脚本的\r字符。
```
cd Swin-Transformer
sed -i 's/\r//' run.sh
```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

使用 notebook 进行代码调试

由于Notebook的/cache目录只能支持500G的存储，超过后会导致实例重启，ImageNet数据集大小超过该限制，因此建议用线下资源调试、或用小批量数据集在Notebook调试（Notebook调试方法与**使用Notebook进行代码调试**相同）。

创建多机多卡训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd /home/ma-user/work/code/Swin-Transformer && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择所需GPU规格。
   - 计算节点个数：选择需要的节点个数。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

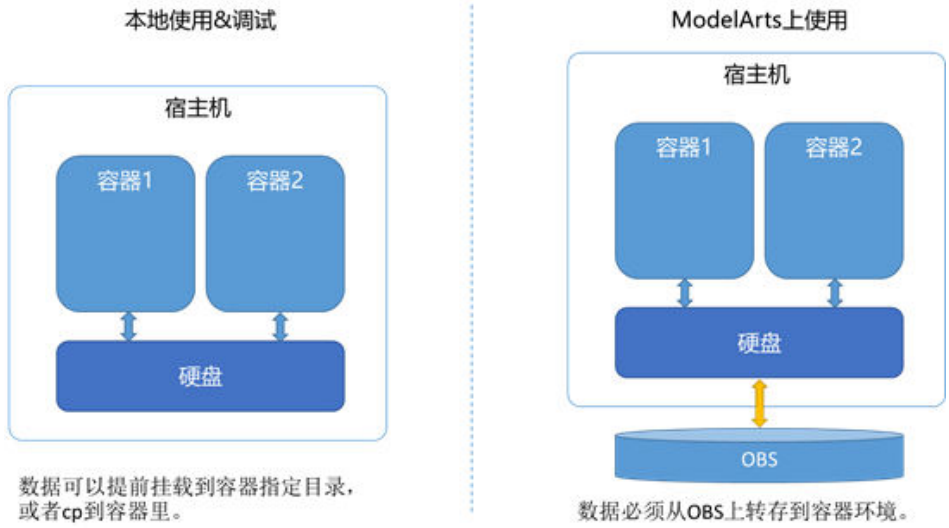
     为了和Notebook调试时代码路径一致，保持相同的启动命令，云上挂载路径需要填写为“/home/ma-user/work”。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

14.2.6 在 ModelArts Standard 使用 run.sh 脚本实现 OBS 和训练容器间的数据传输

自定义容器在ModelArts上训练和本地训练的区别如下图：

图 14-14 本地与 ModelArts 上训练对比



ModelArts上进行训练比本地训练多了一步OBS和容器环境的数据迁移工作。

增加了和OBS交互工作的整个训练流程如下：

说明

建议使用OBSutil作为和OBS交互的工具，如何在本机安装obsutil可以参考**安装和配置OBS命令行工具**。

1. 训练数据、代码、模型下载。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）
2. 启动脚本，用法无切换，一般就是到达执行目录，然后python xxx.py。
3. 训练结果、日志、checkpoints上传。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）

可以用一个run脚本把整个流程包起来。run.sh脚本的内容可以参考如下示例：

```
#!/bin/bash

##认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
##本示例以AK和SK保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

##安装obsutil，完成AKSK配置。建议在基础镜像里做好。
#mkdir -p /opt && cd /opt
#wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/obsutil_linux_amd64.tar.gz
#tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils
#alias obsutil='/opt/utils/obsutil'
#obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-north-4.myhuaweicloud.com

##训练输入复制到容器镜像本地。
#/cache目录的容量较大。

DATA_URL=`echo ${DLS_DATA_URL} | sed /s/s3/obs/`
mkdir –p /cache/data
/opt/utils/obsutil cp –r –f ${DATA_URL} /cache/data

##执行训练作业。
#涉及conda env切换时。
source /xxxxx/etc/profile.d/conda.sh
```

```
conda activate xxxenv
conda info --envs
#启动训练脚本。
cd xxxx
python xxx.py

##复制输出结果到OBS目录。
TRAIN_URL=`echo ${DLS_TRAIN_URL} | sed /s/s3/obs/`
/opt/utils/obsutil cp –r –f /cache/out ${TRAIN_URL}
```

把run.sh放到/opt目录，在实际启动任务的时候，使用以下命令启动任务即可：

```
bash –x /opt/run.sh
```

把run.sh放到/root目录，可以在原镜像里增加一层，这一层就只是COPY这个run脚本。在基础镜像里可以一起把obsutil安装、配置好。参考如下dockerfile：

```
FROM $your_docker_image_tag

RUN mkdir -p /opt && cd /opt && \
        wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/
obsutil_linux_amd64.tar.gz && \
        tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils && \
        /opt/utils/obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-
north-4.myhuaweicloud.com

COPY run.sh /opt/run.sh
```

须知

ModelArts的容器会有一个/cache目录，这个目录挂载的硬盘容量最大。建议下载数据和中间数据都存到这个目录中，防止因硬盘占满导致任务失败。

15 Standard 推理部署

15.1 ModelArts Standard 推理服务访问公网方案

本章节提供了推理服务访问公网的方法。

应用场景

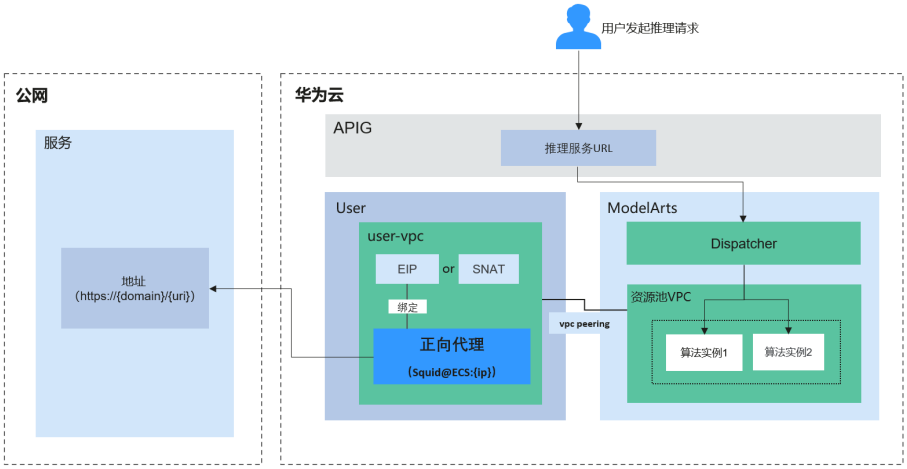
推理服务访问公网地址的场景，如：

- 输入图片，先进行公网OCR服务调用，然后进行NLP处理；
- 进行公网文件下载，然后进行分析；
- 分析结果回调给公网服务终端。

方案设计

从推理服务的算法实例内部，访问公网服务地址的方案。如下图所示：

图 15-1 推理服务访问公网



步骤一：ModelArts 专属资源池打通 VPC

1. 创建好VPC和子网，具体步骤请参考**创建虚拟私有云和子网**。
2. 创建Modelarts专属资源池网络。
   a. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”，默认进入“Standard资源池”页面。
   b. 切换到“网络”页签，单击“创建”，弹出“创建网络”页面。
   c. 在“创建网络”弹窗中填写网络信息。
   d. 确认无误后，单击“确定”。
3. Modelarts专属资源池网络打通VPC。
   a. 在控制台左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”。
   b. 切换到“网络”页签，选择上一步骤创建的网络，单击操作列的“打通VPC”。

   图 15-2 打通 VPC

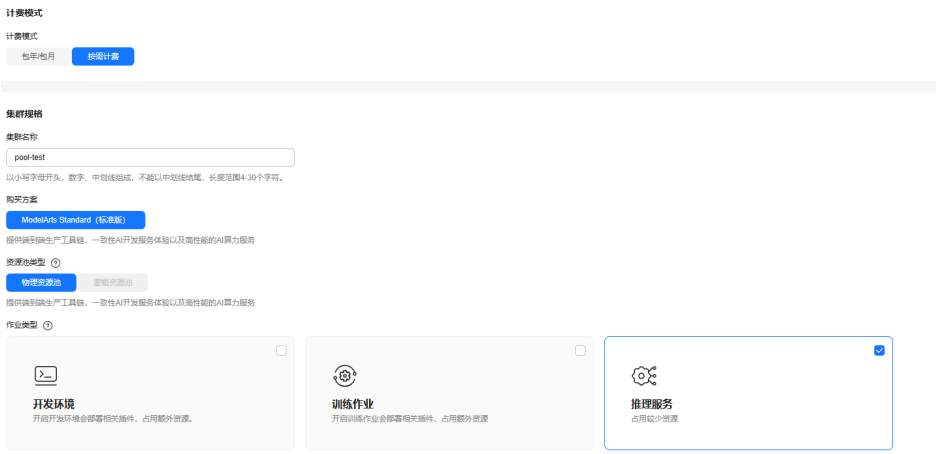
   

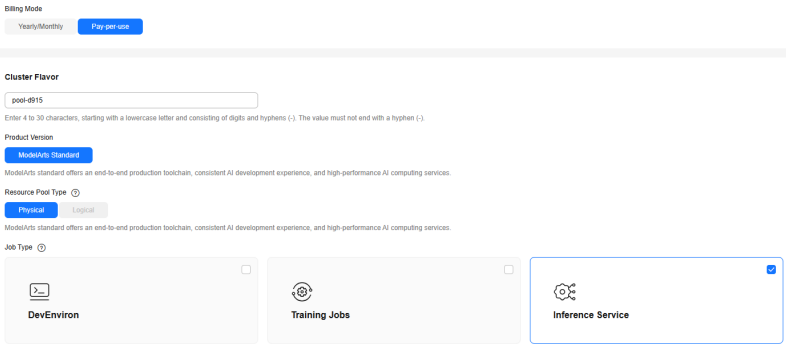
   c. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择提前创建好的VPC和子网。

   说明

   需要打通的对端网络不能和当前网段重叠。
4. 创建Modelarts专属资源池。
   a. 在控制台左侧导航栏中选择“AI专属资源池 > 弹性集群Cluster”。
   b. 在“Standard资源池”页签，单击“购买AI专属集群创建专属资源池”，进入购买AI专属集群创建专属资源池界面填写参数。
   “作业类型”包括推理服务。“网络”选择上文中已打通VPC的网络。

   图 15-3 作业类型



c. 单击“立即购买”确认规格。产品规格和协议许可确认无误后，单击“提交”，即可创建专属资源池。

步骤二：使用 Docker 安装和配置正向代理

1. 购买弹性云服务器ECS，详情请见**购买ECS**。镜像可选择Ubuntu最新版本。虚拟私有云选择提前创建好的VPC。
2. 申请弹性公网IP EIP，详情请见**申请弹性公网IP**。
3. 将弹性公网IP绑定到ECS，详情请见**将弹性公网IP绑定至实例**。
4. 登录ECS，执行如下命令进行Docker安装。如已安装，请直接进入下一步。
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
5. 执行如下命令安装Squid容器。
   ```
   docker pull ubuntu/squid
   ```
6. 创建主机目录。
   ```
   mkdir –p /etc/squid/
   ```
7. 打开并配置whitelist.conf文件。
   ```
   vim whitelist.conf
   ```
   配置内容为安全控制可访问的地址，支持配置通配符，例如：
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   说明

   如果地址访问不通，请在浏览器配置访问域名。
8. 打开并配置squid.conf文件。
   ```
   vim squid.conf
   ```
   配置内容如下。
   ```
   # An ACL named 'whitelist'
   acl whitelist dstdomain '/etc/squid/whitelist.conf'

   # Allow whitelisted URLs through
   http_access allow whitelist

   # Block the rest
   http_access deny all

   # Default port
   http_port 3128
   ```
9. 设置主机目录和配置文件权限如下。
   ```
   chmod 640 -R /etc/squid
   ```
10. 执行如下命令启动Squid实例。
    ```
    docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
    ```
11. 进入docker刷新Squid。
    ```
    docker exec –it squid bash
    root@{container_id}:/# squid -k reconfigure
    ```

步骤三：设置 DNS 代理和调用公网地址

1. 在自定义模型镜像时设置代理指向代理服务器私有IP和端口，如下所示。

```
proxies = {
  "http": "http://{proxy_server_private_ip}:3128",
  "https": "http://{proxy_server_private_ip}:3128"
}
```

代理服务器IP即**步骤二：使用Docker安装和配置正向代理**中创建的ECS私有IP，获取方式请见**查看弹性云服务器详细信息**。

图 15-4 ECS 私有 IP

2. 调用公网地址时，使用服务URL进行业务请求，如：

```
https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
```

15.2 端到端运维 ModelArts Standard 推理服务方案

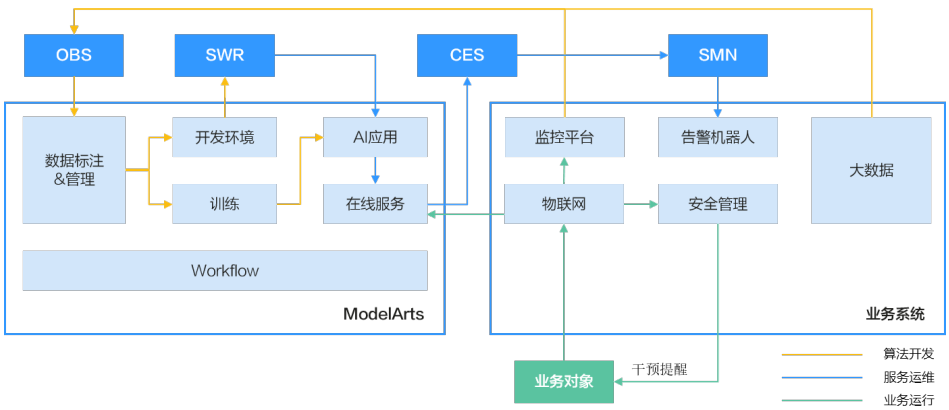
ModelArts推理服务的端到端运维覆盖了算法开发、服务运维和业务运行的整个AI流程。

方案概述

推理服务的端到端运维流程

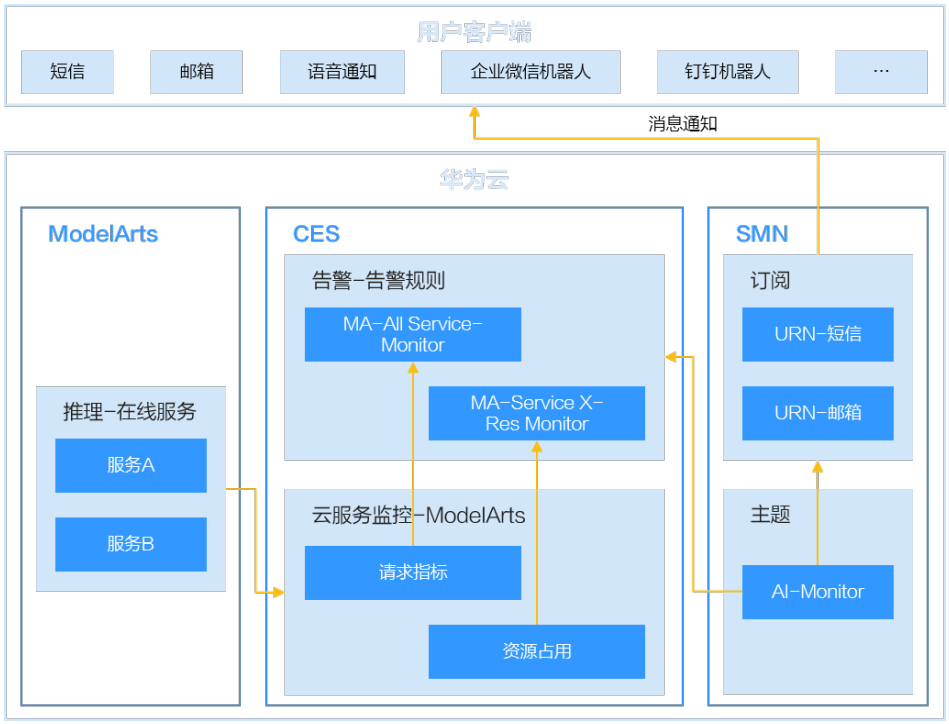
- 算法开发阶段，先将业务AI数据存放到对象存储服务（OBS）中，接着通过ModelArts数据管理进行标注和版本管理，然后通过训练获得AI模型结果，最后通过开发环境构建模型镜像。
- 服务运维阶段，先利用镜像构建模型，接着部署模型为在线服务，然后可在云监控服务（CES）中获得ModelArts推理在线服务的监控数据，最后可配置告警规则实现实时告警通知。
- 业务运行阶段，先将业务系统对接在线服务请求，然后进行业务逻辑处理和监控设置。

图 15-5 推理服务的端到端运维流程图



整个运维过程会对服务请求失败和资源占用过高的场景进行监控，当超过阈值时发送告警通知。

图 15-6 监控告警流程图



方案优势

通过端到端的服务运维配置，可方便地查看业务运行高低峰情况，并能够实时感知在线服务的健康状态。

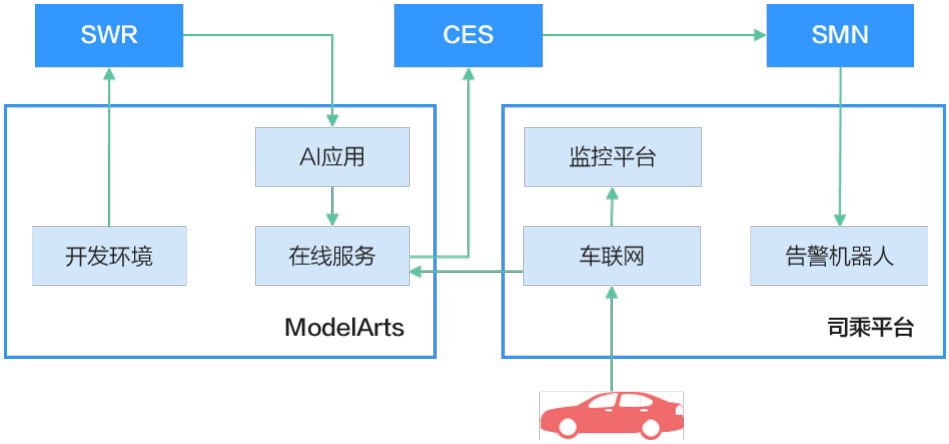
约束限制

端到端服务运维只支持在线服务，因为推理的批量服务和边缘服务无CES监控数据，不支持完整的端到端服务运维设置。

实施步骤

以出行场景的司乘安全算法为例，介绍使用ModelArts进行流程化服务部署和更新、自动化服务运维和监控的实现步骤。

图 15-7 司乘安全算法



步骤1 将用户本地开发完成的模型，使用自定义镜像构建成ModelArts Standard推理平台可以用的模型。具体操作请参考**从0-1制作自定义镜像并创建模型**。

步骤2 在ModelArts管理控制台，使用创建好的模型部署为在线服务。

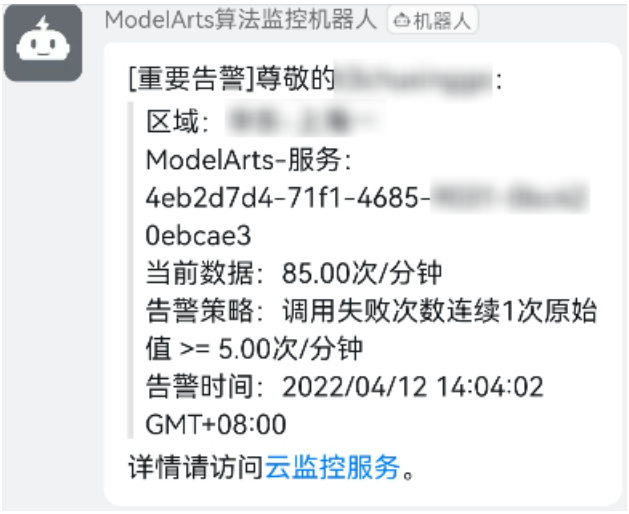
步骤3 登录云监控服务CES管理控制台，设置ModelArts服务的告警规则并配置主题订阅方式发送通知。具体操作请参考**设置告警规则**。

当配置完成后，在左侧导航栏选择“云服务监控 > ModelArts”即可查看在线服务的请求情况和资源占用情况，如下图所示。

图 15-8 查看服务的监控指标

当监控信息触发告警时，主题订阅对象将会收到消息通知。

图 15-9 告警消息通知



----结束

15.3 使用自定义引擎在 ModelArts Standard 创建模型

使用自定义引擎创建模型，用户可以通过选择自己存储在SWR服务中的镜像作为模型的引擎，指定预先存储于OBS服务中的文件目录路径作为模型包来创建模型，轻松地应对ModelArts平台预置引擎无法满足个性化诉求的场景。

自定义引擎创建模型的规范

使用自定义引擎创建模型，用户的SWR镜像、OBS模型包和文件大小需要满足以下规范：

- SWR镜像规范：
  - 镜像必须内置一个用户名为“ma-user”，组名为“ma-group”的普通用户，且必须确保该用户的uid=1000、gid=100。内置用户的dockerfile指令如下：

    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```

  - 明确设置镜像的启动命令。在dockerfile文件中指定cmd，dockerfile指令示例如下：

    ```
    CMD sh /home/mind/run.sh
    ```

    启动入口文件run.sh需要自定义。示例如下：

    ```
    #!/bin/bash

    # 自定义脚本内容
    ...

    # run.sh调用app.py启动服务器，app.py请参考https示例
    python app.py
    ```

    说明

    除了按上述要求设置启动命令，您也可以在镜像中自定义启动命令，在创建模型时填写与您镜像中相同的启动命令。

  - 提供的服务可使用HTTPS/HTTP协议和监听的容器端口，端口和协议可根据镜像实际使用情况自行填写，ModelArts提供的请求协议和端口号的缺省值是HTTPS和8080。请参考**https示例**。
  - （可选）健康检查的URL路径必须为"/health"。
- OBS模型包规范

  模型包的名字必须为model。模型包规范请参见**模型包规范介绍**。
- 文件大小规范

  当使用公共资源池时，SWR的镜像大小（指下载后的镜像大小，非SWR界面显示的压缩后的镜像大小）和OBS模型包大小总和不大于30G。

https 示例

使用Flask启动https，Webserver代码示例如下：

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

在本地机器调试

自定义引擎的规范可以在安装有docker的本地机器上通过以下步骤提前验证：

1. 将自定义引擎镜像下载至本地机器，假设镜像名为custom_engine:v1。
2. 将模型包文件夹复制到本地机器，假设模型包文件夹名字为model。
3. 在模型包文件夹的同级目录下验证如下命令拉起服务：

   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   说明

   该指令无法完全模拟线上，主要是由于-v挂载进去的目录是root权限。在线上，模型文件从OBS下载到/home/mind/model目录之后，文件owner将统一修改为ma-user。

4. 在本地机器上启动另一个终端，执行以下验证指令，得到符合预期的推理结果。
```
curl https://127.0.0.1:8080/${推理服务的请求路径}
```

推理部署示例

本节将详细说明以自定义引擎方式创建模型的步骤。

1. 创建模型并查看模型详情

   登录ModelArts管理控制台，进入“模型管理”页面中，单击“创建模型”，进入模型创建页面，设置相关参数如下：

   - 元模型来源：选择“从对象存储服务（OBS）中选择”。
   - 选择元模型：从OBS中选择一个模型包。
   - AI引擎：选择“Custom”。
   - 引擎包：从容器镜像中选择一个镜像。
   - 容器调用接口：端口和协议可根据镜像实际使用情况自行填写。

     其他参数保持默认值。

   单击“立即创建”，跳转到模型列表页，查看模型状态，当状态变为“正常”，模型创建成功。

   图 15-10 创建模型

   单击模型名称，进入模型详情页面，查看模型详情信息。

2. 部署服务并查看详情

   在模型详情页面，单击右上角“部署>在线服务”，进入服务部署页面，模型和版本默认选中，选择合适的“实例规格”（例如CPU：2核 8GB），其他参数可保持默认值，单击“下一步”，跳转至服务列表页，当服务状态变为“运行中”，服务部署成功。

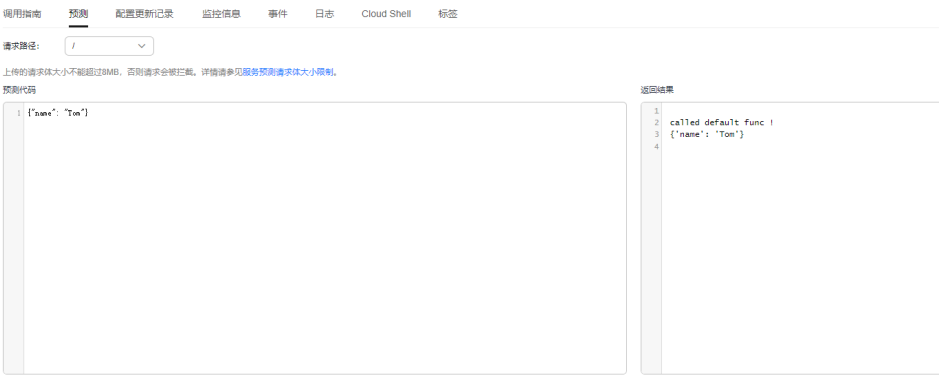
   单击服务名称，进入服务详情页面，查看服务详情信息，单击“日志”页签，查看服务日志信息。

图 15-11 查看服务日志信息

3. 服务预测

   在服务详情页面，单击“预测”页签，进行服务预测。

图 15-12 服务预测



15.4 使用大模型在 ModelArts Standard 创建模型部署在线服务

背景说明

目前大模型的参数量已经达到千亿甚至万亿，随之大模型的体积也越来越大。千亿参数大模型的体积超过200G，在版本管理、生产部署上对平台系统产生了新的要求。例如：导入模型时，需要支持动态调整租户存储配额；模型加载、启动慢，部署时需要灵活的超时配置；当负载异常重启，模型需要重新加载，服务恢复时间长的问题亟待解决。

为了应对如上诉求，ModelArts推理平台针对性给出解决方案，用于支持大模型场景下的模型管理和服务部署。

约束与限制

- 需要申请单个模型大小配额和添加使用节点本地存储缓存的白名单。
- 需要使用自定义引擎Custom，配置动态加载。
- 需要使用专属资源池部署服务。
- 专属资源池磁盘空间需大于1T。

操作事项

1. 申请扩大模型的大小配额和使用节点本地存储缓存白名单
2. 上传模型数据并校验上传对象的一致性
3. 创建专属资源池
4. 创建模型
5. 部署在线服务

申请扩大模型的大小配额和使用节点本地存储缓存白名单

服务部署时，默认情况下，动态加载的模型包位于临时磁盘空间，服务停止时已加载的文件会被删除，再次启动时需要重新加载。为了避免反复加载，平台允许使用资源池节点的本地存储空间来加载模型包，并在服务停止和重启时仍有效（通过哈希值保证数据一致性）

使用大模型要求用户采用自定义引擎，并开启动态加载的模式导入模型。基于此，需要执行以下操作：

- 如果模型超过默认配额值，需要提工单申请扩大单个模型的大小配额。单个模型大小配额默认值为20GB。
- 需要提工单申请添加使用节点本地存储缓存的白名单。

上传模型数据并校验上传对象的一致性

为了动态加载时保证数据完整性，需要在上传模型数据至OBS时，进行上传对象的一致性校验。obsutil、OBS Browser+以及OBS SDK都支持在上传对象时进行一致性校验，您可以根据自己的业务选择任意一种方式进行校验。详见**校验上传对象的一致性**。

以OBS Browser+为例，如**图15-13**。使用OBS Browser+上传数据，开启MD5校验，动态加载并使用节点本地的持久化存储时，检查数据一致性。

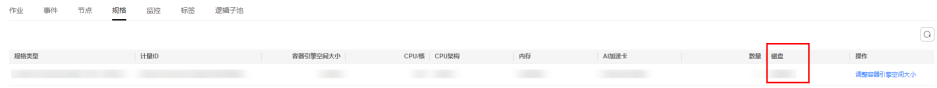
图 15-13 OBS Browser+配置 MD5 校验

创建专属资源池

使用本地的持久化存储功能，需使用专属资源池，且专属资源池磁盘空间大小必须超过1T。您可以通过专属资源池详情页面，规格页签，查看专属资源池磁盘信息。当服

务部署失败，提示磁盘空间不足时，请参考**服务部署、启动、升级和修改时，资源不足如何处理?**

图 15-14 查看专属资源池磁盘信息



创建模型

使用大模型创建模型，选择从对象存储服务（OBS）中导入，需满足以下参数配置：

1. 采用自定义引擎，开启动态加载

   使用大模型要求用户使用自定义引擎，并开启动态加载的模式导入模型。用户可以制作自定义引擎，满足大模型场景下对镜像依赖包、推理框架等的特殊需求。自定义引擎的制作请参考**使用自定义引擎在ModelArts Standard创建模型**。

   当用户使用自定义引擎时，默认开启动态加载，模型包与镜像分离，在服务部署时动态将模型加载到服务负载。

2. 配置健康检查

   大模型场景下导入的模型，要求配置健康检查，避免在部署时服务显示已启动但实际不可用。

   图 15-15 采用自定义引擎，开启动态加载并配置健康检查示例图

部署在线服务

部署服务时，需满足以下参数配置：

1. 自定义部署超时时间

   大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

2. 添加环境变量

   部署服务时，增加如下环境变量，会将负载均衡的请求亲和策略配置为集群亲和，避免未就绪的服务实例影响预测成功率。

   ```
   MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
   ```

图 15-16 自定义部署超时时间和添加环境变量示例图

建议部署多实例，增加服务可靠性。

15.5 第三方推理框架迁移到 ModelArts Standard 推理自定义引擎

背景说明

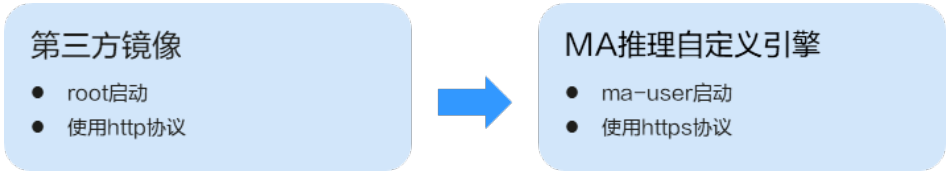
ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

- TensorFlow Serving是一个灵活、高性能的机器学习模型部署系统，提供模型版本管理、服务回滚等能力。通过配置模型路径、模型端口、模型名称等参数，原生TFServing镜像可以快速启动提供服务，并支持gRPC和HTTP Restful API的访问方式。
- Triton是一个高性能推理服务框架，提供HTTP/gRPC等多种服务协议，支持TensorFlow、TensorRT、PyTorch、ONNXRuntime等多种推理引擎后端，并且支持多模型并发、动态batch等功能，能够提高GPU的使用率，改善推理服务的性能。

当从第三方推理框架迁移到使用ModelArts推理的模型管理和服务管理时，需要对原生第三方推理框架镜像的构建方式做一定的改造，以使用ModelArts推理平台的模型版本管理能力和动态加载模型的部署能力。本案例将指导用户完成原生第三方推理框架镜像到ModelArts推理自定义引擎的改造。自定义引擎的镜像制作完成后，即可以通过模型导入对模型版本进行管理，并基于模型进行部署和管理服务。

适配和改造的主要工作项如下：

图 15-17 改造工作项



针对不同框架的镜像，可能还需要做额外的适配工作，具体差异请见对应框架的操作步骤。

- **TFServing框架迁移操作步骤**
- **Triton框架迁移操作步骤**

TFServing 框架迁移操作步骤

步骤1 增加用户ma-user。

基于原生"tensorflow/serving:2.8.0"镜像构建，镜像中100的用户组默认已存在，Dockerfile中执行如下命令增加用户ma-user。

```
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

协议转换为https之后，对外暴露的端口从tfserving的8501变为8080。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
```



```
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8501;
    }
}
```

5. 准备启动脚本。

说明

启动前先创建ssl证书，然后启动TFServing的启动脚本。

启动脚本run.sh示例代码如下：

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
```

```
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText
sh /usr/bin/tf_serving_entrypoint.sh
```

步骤3 修改模型默认路径，支持ModelArts推理模型动态加载。

Dockerfile中执行如下命令修改默认的模型路径。

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----结束

完整的Dockerfile参考：

```
FROM tensorflow/serving:2.8.0
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

Triton 框架迁移操作步骤

本教程基于nvidia官方提供的nvcr.io/nvidia/tritonserver:23.03-py3镜像进行适配，使用开源大模型llama7b进行推理任务。

步骤1 增加用户ma-user。

Triton镜像中默认已存在id为1000的triton-server用户，需先修改triton-server用户名id后再增加用户ma-user，Dockerfile中执行如下命令。

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
```


2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
```

```
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

5. 准备启动脚本run.sh。

   说明

   启动前先创建ssl证书，然后启动Triton的启动脚本。

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText

bash /home/mind/model/triton_serving.sh
```

步骤3 编译安装tensorrtllm_backend。

1. Dockerfile中执行如下命令获取tensorrtllm_backend源码，安装tensorrt、cmake和pytorch等相关依赖，并进行编译安装。

```
# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
```



```
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver
```

2. 准备triton serving的启动脚本triton_serving.sh，llama模型的参考样例如下：

```
MODEL_NAME=llama_7b
MODEL_DIR=/home/mind/model/${MODEL_NAME}
OUTPUT_DIR=/tmp/llama/7B/trt_engines/fp16/1-gpu/
MAX_BATCH_SIZE=1
export LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH}

# build tensorrt_llm engine
cd /opt/tritonserver/tensorrtllm_backend/tensorrt_llm/examples/llama
python build.py --model_dir ${MODEL_DIR} \
                --dtype float16 \
                --remove_input_padding \
                --use_gpt_attention_plugin float16 \
                --enable_context_fmha \
                --use_weight_only \
                --use_gemm_plugin float16 \
                --output_dir ${OUTPUT_DIR} \
                --paged_kv_cache \
                --max_batch_size ${MAX_BATCH_SIZE}

# set config parameters
cd /opt/tritonserver/tensorrtllm_backend
mkdir triton_model_repo
cp all_models/inflight_batcher_llm/* triton_model_repo/ -r

python3 tools/fill_template.py -i triton_model_repo/preprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},preprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/postprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},postprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/ensemble/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE}
python3 tools/fill_template.py -i triton_model_repo/tensorrt_llm/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE},decoupled_mode:False,max_beam_width:1,engine_dir:$
{OUTPUT_DIR},max_tokens_in_paged_kv_cache:2560,max_attention_window_size:2560,kv_cache_free_
gpu_mem_fraction:0.5,exclude_input_in_output:True,enable_kv_cache_reuse:False,batching_strategy:V1,
max_queue_delay_microseconds:600

# launch tritonserver
python3 scripts/launch_triton_server.py --world_size 1 --model_repo=triton_model_repo/
while true; do sleep 10000; done
```

部分参数说明：

- MODEL_NAME：HuggingFace格式模型权重文件所在OBS文件夹名称。
- OUTPUT_DIR：通过TensorRT-LLM转换后的模型文件在容器中的路径。

完整的Dockerfile如下：

```
FROM nvcr.io/nvidia/tritonserver:23.03-py3

# add ma-user and install nginx
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash
ma-user && \
    apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
```

```
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

完成镜像构建后，将镜像注册至华为云容器镜像服务SWR中，用于后续在ModelArts上部署推理服务。

步骤4 使用适配后的镜像在ModelArts部署在线推理服务。

1. 在obs中创建model目录，并将triton_serving.sh文件和llama_7b文件夹上传至model目录下，如下图所示。

图 15-18 上传至 model 目录

2. 创建模型，源模型来源选择“从对象存储服务（OBS）中选择”，元模型选择至model目录，AI引擎选择Custom，引擎包选择步骤3构建的镜像。

图 15-19 创建模型

3. 将创建的模型部署为在线服务，大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

图 15-20 部署为在线服务

4. 调用在线服务进行大模型推理，请求路径填写/v2/models/ensemble/infer，调用样例如下：

```
{
  "inputs": [
    {
      "name": "text_input",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": ["what is machine learning"]
    },
    {
      "name": "max_tokens",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [64]
    },
    {
      "name": "bad_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "stop_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "pad_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    },
    {
      "name": "end_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    }
  ],
  "outputs": [
    {
      "name": "text_output"
    }
  ]
}
```

说明

- "inputs"中"name"为"text_input"的元素代表输入，"data"为具体输入语句，本示例中为"what is machine learning"。
- "inputs"中"name"为"max_tokens"的元素代表输出最大tokens数，"data"为具体数值，本示例中为64。

图 15-21 调用在线服务

----结束

15.6 ModelArts Standard 推理服务支持 VPC 直连的高速访问通道配置

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

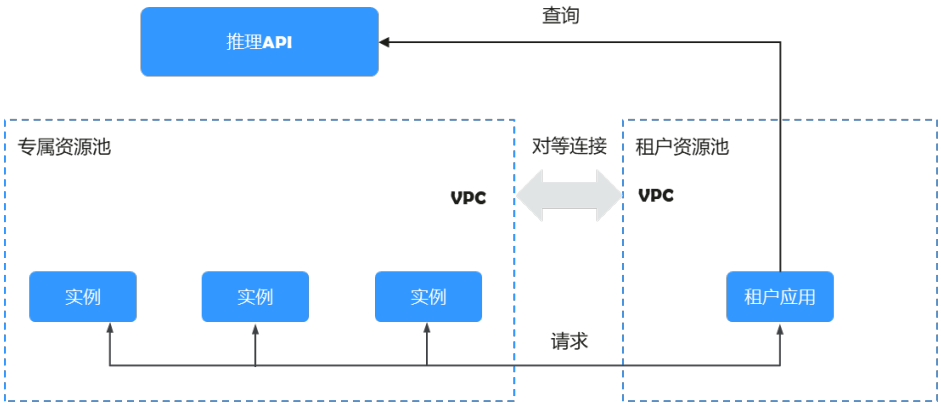
使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

说明

由于请求不经过推理平台，所以会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

图 15-22 VPC 直连的高速访问通道示意图



约束限制

调用API访问在线服务时，对预测请求体大小和预测时间有限制：

- 请求体的大小不超过12MB，超过后请求会被拦截。
- 因APIG（API网关）限制，平台每次请求预测的时间不超过40秒。

准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

须知

- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“AI专属资源池 > 弹性集群Cluster”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回专属资源池列表，选择“网络”页签，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

图 15-23 查看网络配置

图 15-24 打通 VPC

步骤2 VPC下创建弹性云服务器

登录弹性云服务器ECS控制台，单击右上角“购买弹性云服务器”，进入购买弹性云服务器页面，完成基本配置后单击“下一步：网络配置”，进入网络配置页面，选择**步骤1**中打通的VPC，完成其他参数配置，完成高级配置并确认配置，下发购买弹性云服务器的任务。等待服务器的状态变为“运行中”时，弹性云服务器创建成功。单击“名称/ID”，进入服务器详情页面，查看虚拟私有云配置信息。

图 15-25 购买弹性云服务器时选择 VPC

图 15-26 查看虚拟私有云配置信息

步骤3 获取在线服务的IP和端口号

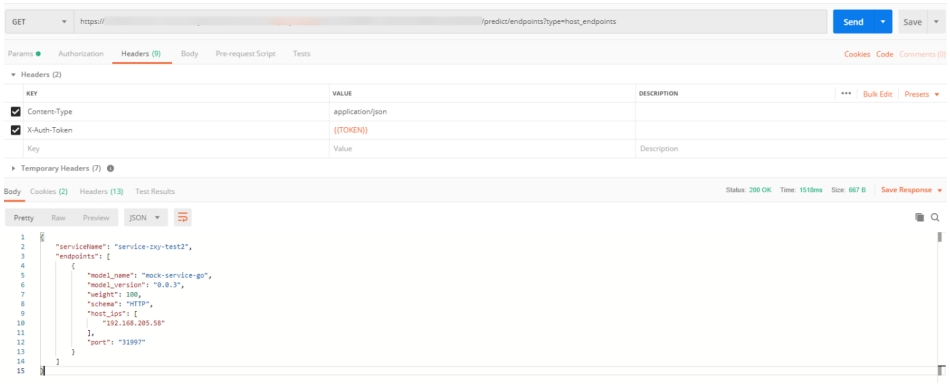
可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

API接口：

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- 方式一：图形界面的软件获取服务的IP和端口号

  图 15-27 接口返回示例

- 方式二：Python语言获取IP和端口号

  Python代码如下，下述代码中以下参数需要手动修改：

  - project_id：用户项目ID，获取方法请参见**获取项目ID和名称**。
  - service_id：服务ID，在服务详情页可查看。
  - REGION_ENDPOINT：服务的终端节点，查询请参见**终端节点**。

```
def get_app_info(project_id, service_id):
    list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
    headers = {'X-Auth-Token': X_Auth_Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

步骤4 通过IP和端口号直连应用

登录弹性云服务器（ECS），可以通过Linux命令行访问在线服务，也可以创建Python环境运行Python代码访问在线服务。schema、 ip、port参数值从**步骤3**获取。

- 执行命令示例如下，直接访问在线服务。

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

  图 15-28 访问在线服务

```
[root@ecs-zxy ~]# curl --location --request POST 'http://192.168.205.58:31997' \
> --header 'Content-Type: application/json' \
> --data-raw '{"a":"a"}'
call Post()[root@ecs-zxy ~]#
```

- 创建Python环境，运行Python代码访问在线服务。

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    url = infer_url.format(schema, ip, port)
    response = requests.post(url, data=body)
    print(response.content)
```

说明

由于高速通道特性会缺失负载均衡的能力，因此在多实例时需要自主制定负载均衡策略。

----结束

15.7 ModelArts Standard 的 WebSocket 在线服务全流程开发

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 用户需有一定的Java开发经验，熟悉jar打包流程。
- 用户需了解WebSocket协议的基本概念及调用方法。
- 用户需熟悉Docker制作镜像的方法。

约束与限制

- WebSocket协议只支持部署在线服务。
- 只支持自定义镜像导入模型部署的在线服务。

准备工作

ModelArts使用WebSocket完成推理需要用户自己准备自定义镜像，该自定义镜像需要在单机环境下能够提供完整的WebSocket服务，如完成WebSocket的握手，client向server发送数据，server向client发送数据等。模型的推理过程在自定义镜像中完成，如下载模型，加载模型，执行预处理，完成推理，拼装响应体等。

操作步骤

WebSocket在线服务开发操作步骤如下：

- **上传镜像至容器镜像服务**
- **使用镜像创建模型**
- **使用模型部署在线服务**
- **WebSocket在线服务调用**

上传镜像至容器镜像服务

将准备好的本地镜像上传到容器镜像服务（SWR）。

使用镜像创建模型

1. 登录ModelArts管理控制台，进入“ 模型管理”页面，单击“创建”，跳转至创建模型页面。

2. 完成模型配置，部分配置如下：
   - 元模型来源：选择“从容器镜像中选择”。
   - 容器镜像所在的路径：选择**上传镜像至容器镜像服务**上传的路径。
   - 容器调用接口：根据实际情况配置容器调用接口。
   - 健康检查：保持默认。如果镜像中配置了健康检查则按实际情况配置健康检查。

   图 15-29 模型配置参数

3. 单击“立即创建”，进入模型列表页，等模型状态变为“正常”，表示模型创建成功。

使用模型部署在线服务

1. 登录ModelArts管理控制台，进入“模型部署 >在线服务”页面，单击“部署”，跳转至在线服务部署页面。
2. 完成服务的配置，部分配置如下：
   - 选择模型及版本：选择**使用镜像创建模型**创建完成的模型及版本
   - 升级为WebSocket：打开开关

   图 15-30 升级为 WebSocket

3. 单击“下一步”，确认配置后“提交”，完成在线服务的部署。返回在线服务列表页，查看服务状态变为“运行中”，表示服务部署成功。

WebSocket 在线服务调用

WebSocket协议本身不提供额外的认证方式。不管自定义镜像里面是ws还是wss，经过ModelArts平台出去的WebSocket协议都是wss的。同时wss只支持客户端对服务端的单向认证，不支持服务端对客户端的双向认证。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

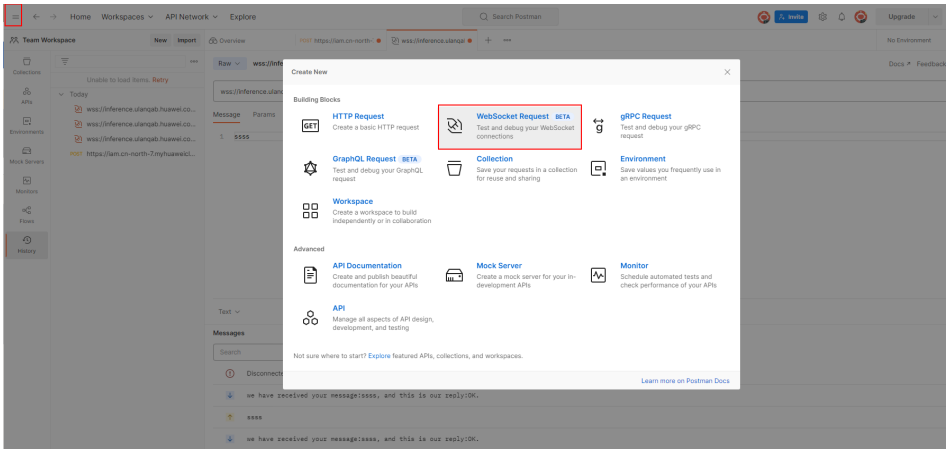
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

1. **WebSocket连接的建立**
2. **WebSocket客户端和服务端双向传输数据**

步骤1 WebSocket连接的建立

1. 打开Postman（需选择8.5 以上版本，以10.12.0为例）工具，单击左上角 ，选择“File>New”，弹出新建对话框，选择“WebSocket Request”（当前为beta版本）功能：

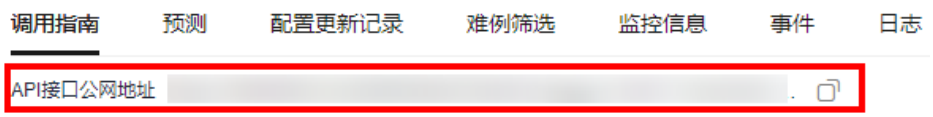
   图 15-31 选择 WebSocket Request 功能

2. 在新建的窗口中填入WebSocket连接信息：

   左上角选择Raw，不要选择Socket.IO（一种WebSocket实现，要求客户端跟服务端都要基于Socket.IO），地址栏中填入从服务详情页“调用指南”页签中获取“API接口调用公网地址”后面的地址。如果自定义镜像中有更细粒度的地址，则在地址后面追加该URL。如果有queryString，那么在params栏中添加参数。在header中添加认证信息（不同认证方式有不同header，跟https的推理服务相同）。选择单击右上的connect按钮，建立WebSocket连接。

   图 15-32 获取 API 接口调用公网地址

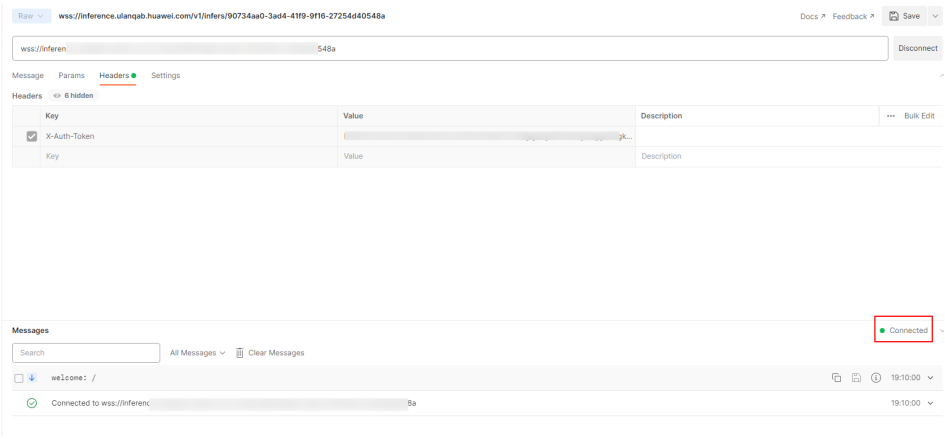
   

说明

- 如果信息正确，右下角连接状态处会显示：CONNECTED；
- 如果无法建立连接，如果是401状态码，检查认证信息；
- 如果显示WRONG_VERSION_NUMBER等关键字，检查自定义镜像的端口和ws跟wss的配置是否正确。

连接成功后结果如下：

图 15-33 连接成功



须知

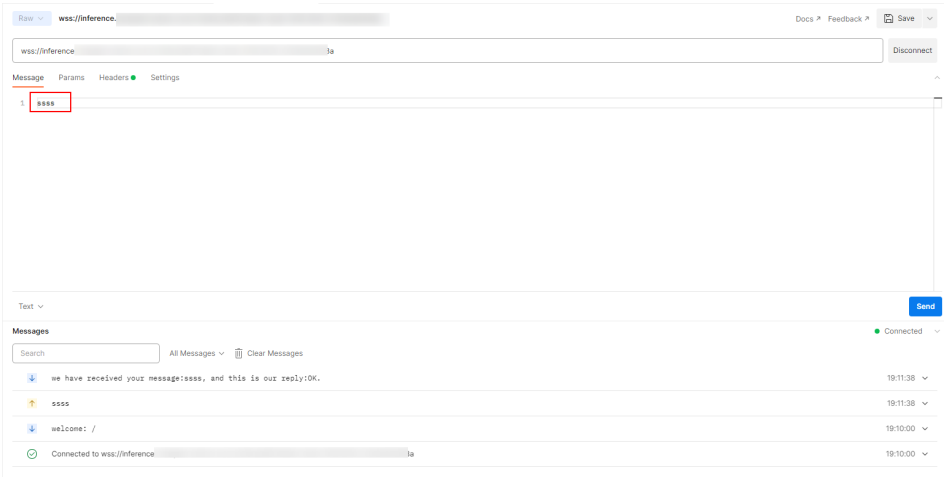
优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

步骤2 WebSocket客户端和服务端双向传输数据

连接建立后，WebSocket使用TCP完成全双工通信。WebSocket的客户端可以往服务端发送数据，客户端有不同的实现，同一种语言也存在不同的lib包的实现，这里不考虑实现的不同种类。

客户端发送的内容在协议的角度不限定格式，Postman支持Text/Json/XML/HTML/Binary，以text为例，在输入框中输入要发送的文本，单击右侧中部的Send按钮即可将请求发往服务端，当文本内容过长，可能会导致postman工具卡住。

图 15-34 发送数据



----结束

15.8 从 0-1 制作自定义镜像并创建模型

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为模型。本文详细介绍如何使用自定义镜像完成模型的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考**创建AI应用的自定义镜像规范**。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。
3. **将自定义镜像创建为模型**：将上传至SWR服务的镜像导入ModelArts的模型。
4. **将模型部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录弹性云服务器**。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 15-35 创建 ECS 服务器-选择 X86 架构的公共镜像

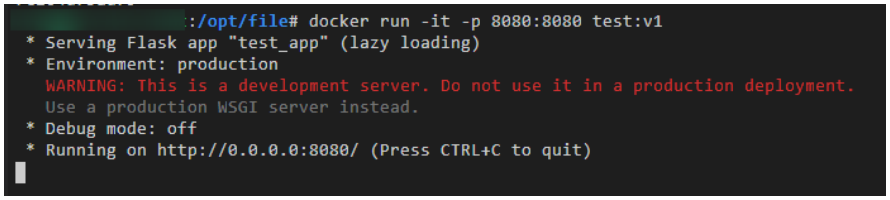
1. 登录主机后，安装Docker，可参考**Docker官方文档**。也可执行以下命令安装docker。
```
curl -fsSL get.docker.com -o get-docker.sh
sh get-docker.sh
```
2. 获取基础镜像。本示例以Ubuntu18.04为例。
```
docker pull ubuntu:18.04
```
3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。

   文件结构如下所示
```
self-define-images/
   --Dockerfile
   --test_app.py
```
   - “Dockerfile”
```
From ubuntu:18.04
# 配置华为云的源，安装 python、python3-pip 和 Flask
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
  sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
&& \
  apt-get update && \
  apt-get install -y python3 python3-pip && \
  pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
repository/pypi/simple  Flask

# 复制应用服务代码进镜像里面
COPY test_app.py /opt/test_app.py

# 指定镜像的启动命令
CMD python3  /opt/test_app.py
```
   - “test_app.py”
```
from flask import Flask, request
import json
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```
4. 进入“self-define-images”文件夹，执行以下命令构建自定义镜像“test:v1”。
```
docker build -t test:v1 .
```
5. 您可以使用“docker images”查看您构建的自定义镜像。

本地验证镜像并上传镜像至 SWR 服务

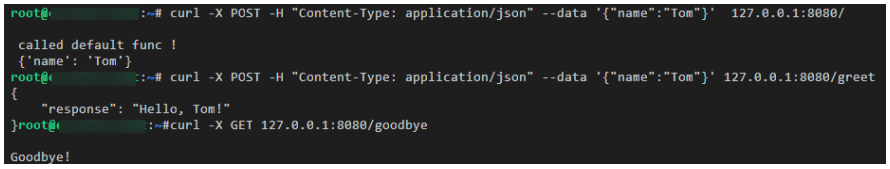
1. 在本地环境执行以下命令启动自定义镜像
```
docker run -it -p 8080:8080 test:v1
```

图 15-36 启动自定义镜像



2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。
```
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
curl -X GET 127.0.0.1:8080/goodbye
```
如果验证自定义镜像功能成功，结果如下图所示。

图 15-37 校验接口



3. 上传自定义镜像至SWR服务。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为模型

参考**从容器镜像中选择元模型**导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
  - 容器镜像所在的路径：选择已制作好的自有镜像

图 15-38 选择已制作好的自有镜像

  - 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
  - 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
  - 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致模型创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见**模型配置文件说明**。

本样例的配置文件如下所示：

```
[{
        "url": "/",
        "method": "post",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    },
{
        "url": "/greet",
        "method": "post",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    },
{
        "url": "/goodbye",
        "method": "get",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    }
]
```

将模型部署为在线服务

1. 参考**部署为在线服务**将模型部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。
3. 您可以通过“预测”页签访问在线服务。

图 15-39 访问在线服务

15.9 使用 AppCode 认证鉴权方式进行在线预测

场景描述

APPcode认证是一种简易的API调用认证方式，通过在HTTP请求头中添加参数X-Apig-AppCode来实现身份认证，无需复杂的签名过程，适合于客户端环境安全可控的场景，如内网系统之间的API调用。在ModelArts中，支持在部署在线服务时开启

AppCode认证（**部署模型为在线服务**中的“支持APP认证”参数）。对于已部署的在线服务，ModelArts支持修改其配置开启AppCode认证。

本文主要介绍如何修改一个已有的在线服务，使其支持AppCode认证并进行在线预测。

前提条件

提前部署在线服务，具体操作可以参考案例：**使用ModelArts Standard一键完成商超商品识别模型部署**。

操作步骤

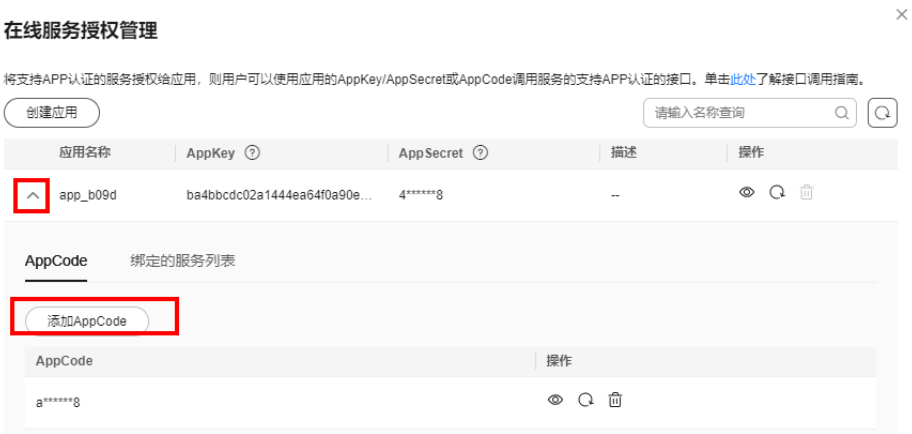
步骤1 在ModelArts控制台页面菜单栏中，单击“模型部署 > 在线服务”，进入在线服务页面。

步骤2 单击“授权管理”后，单击创建应用即可创建App应用。

图 15-40 授权管理

步骤3 创建完App应用后即可展开App，然后单击添加AppCode即可添加AppCode。

图 15-41 添加 AppCode



步骤4 单击**使用ModelArts Standard一键完成商超商品识别模型部署**案例中创建的在线服务名称，进入在线服务详情页，单击“修改”，进入修改在线服务页面。

图 15-42 修改服务

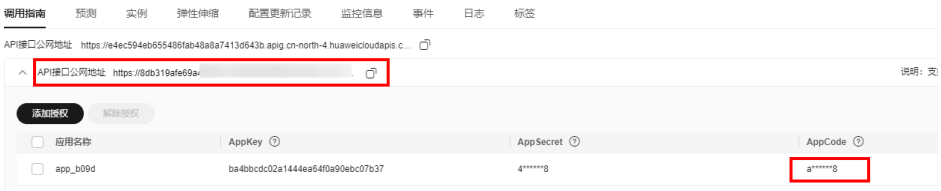
步骤5 开启APP认证开关，选择自己创建的APP应用名称，单击“下一步”，然后单击“提交”即可保存修改。

图 15-43 开启 APP 认证

步骤6 AppCode认证预测。

1. 在部署服务详情中单击“调用指南”，第二行的API接口公网地址即为APP认证调用地址，展开后即可看到AppCode值。

   图 15-44 调用指南

2. 在postman调试预测采用AppCode认证：
   - 请求POST URL填APP认证调用地址
   - 请求头Headers中KEY参数为X-Apig-AppCode，VALUE参数为AppCode值
   - 请求Body按照接口定义传参，本案例中KEY参数为images，选择为File格式，VALUE参数单击上传需要识别的图片。

   图 15-45 Headers

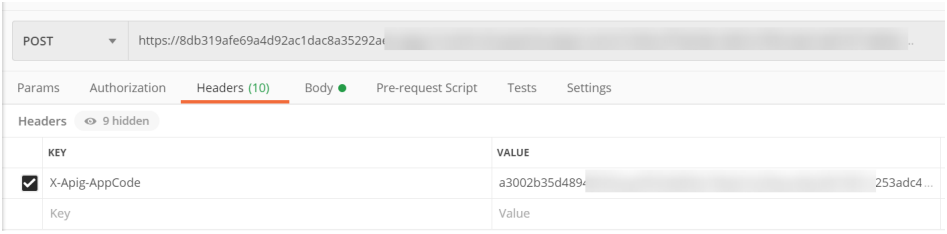
   

   图 15-46 Body

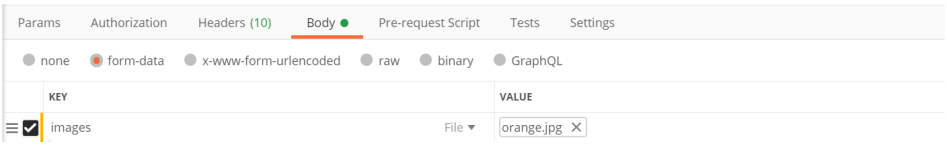
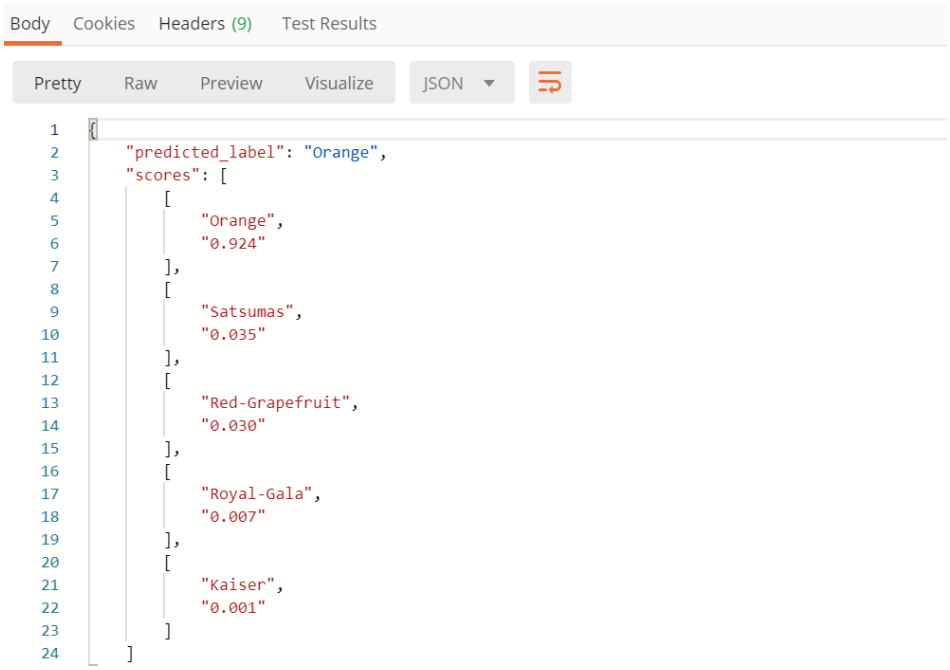
   

图 15-47 返回结果



----结束

常见 APP 认证报错分析

- 报错信息 "error_msg": "The API does not exist or has not been published in the environment", "error_code": "APIG.0101"

  该报错需要检查App认证API是否还存在或者URL是否正确。

- 报错信息 "error_msg": "Incorrect app authentication information: app not found with specified appCode", "error_code": "APIG.0303"

  该报错需要检查请求头Headers参数中X-Apig-AppCode参数的值是否填错。

- 报错信息 "error_msg": "Backend unavailable", "error_code": "APIG.0202"

  该报错信息需要检查dispatcher实例是否正常。

16 历史待下线案例

16.1 使用 AI Gallery 的订阅算法实现花卉识别

本案例以“ResNet_v1_50”算法、花卉识别数据集为例，指导如何从AI Gallery下载数据集和订阅算法，然后使用算法创建训练模型，将所得的模型部署为在线服务。其他算法操作步骤类似，可参考“ResNet_v1_50”算法操作。

步骤1：准备训练数据

步骤2：订阅算法

步骤3：使用订阅算法创建训练作业

步骤4：创建AI应用

步骤5：部署为在线服务（CPU）

步骤6：清除资源

说明

费用说明：本案例使用过程中，从AI Gallery下载数据集和订阅算法免费，在ModelArts上运行训练作业推荐使用免费资源，将模型部署为在线服务推荐使用免费资源。但是数据集存储在OBS桶中会收取少量费用，具体计费请参见**OBS价格详情页**，案例使用完成后请及时清除资源和数据。

准备工作

- 注册华为账号并开通华为云、实名认证
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在弹出的“访问授权”窗口中，

  授权对象类型：**所有用户**

委托选择：新增委托

权限配置：普通用户

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 1：准备训练数据

1. 从AI Gallery下载训练数据，单击链接**四类花卉图像分类小数据集**，进入数据集详情页。
2. 选择“数据集文件”页签后，单击“下载文件”跳转至下载详情页面。
3. 在下载详情页面，填写参数。
   - 下载方式：选择“对象存储服务（OBS）”
   - 目标区域：选择“华北-北京四”（即要部署服务的云服务区）
   - 目标位置：请选择一个空的OBS目录，本示例为“/test-modelartsz/dataset-flower/”

     说明

     此处从AI Gallery下载并使用数据集是限时免费的，但数据集存储在OBS，从OBS中读取数据需要根据OBS的计费原则收费。
4. 确认无误后，单击确定。页面自动跳转到“我的数据>我的下载”页面，请耐心等待，预计5分钟左右。
5. 下载完成后，您可以单击目标位置跳转至OBS桶中查看是否存在已下载的数据。

步骤 2：订阅算法

1. 在AI Gallery搜索“ResNet_v1_50”，进入算法详情页。
2. 单击右侧的“训练 > ModelArts”后，选择ModelArts的云服务区域（即要部署服务的云服务区），单击“确认”，跳转至ModelArts的“算法管理>我的订阅”中。

步骤 3：使用订阅算法创建训练作业

算法订阅成功后，算法将呈现在“算法管理>我的订阅”中，您可以使用订阅的“ResNet_v1_50”算法创建训练作业，获得模型。

1. 进入“算法管理 > 我的订阅”页面，选择订阅的“图像分类-ResNet_v1_50”算法，单击操作列的“创建训练作业”。
2. 在创建训练作业页面，参考如下说明填写关键参数。
   - “创建方式>我的订阅”：系统默认选择订阅的算法，请勿随意修改。
   - “训练输入”：选择数据存储位置，然后从弹出的窗口中选择**步骤1：准备训练数据**中下载好的数据dataset-flower。
   - “训练输出”：选择一个OBS空目录存储训练输出的模型，例如：“test-modelartsz/output-flower”。
   - “超参”：建议采用默认值。
   - “资源类型”：可以选择限时免费的GPU规格资源，如果希望训练效率更高，可以选择收费的GPU资源。

- “计算节点个数”：建议采用默认值1。

3. 参数填写完成后，单击“提交”，根据界面提示确认规格，单击“确定”，完成训练作业创建。
4. 进入“训练管理 > 训练作业”页面，等待训练作业完成。

   训练作业运行需要几分钟时间，请耐心等待。根据经验，选择样例数据集，使用GPU资源运行，预计3分钟左右可完成。

   当训练作业的状态变更为“已完成”时，表示已运行结束。

   您可以单击训练作业名称，进入详情页面，了解训练作业的“配置信息”、“日志”、“资源占用情况”和“评估结果”等信息。您也可以在配置的“训练输出位置”对应的OBS目录下获得训练生成的模型。

步骤 4：创建 AI 应用

1. 在训练作业详情页的右上角单击“创建AI应用”，进入创建AI应用页面。

   也可以在ModelArts管理控制台，选择“资产管理 > AI应用”，在“自定义AI应用”页面，单击“创建”，进入创建AI应用页面。
2. 在创建AI应用页面，系统会自动根据上一步训练作业填写参数，参考如下说明确认关键参数。

   “元模型来源”：系统自动选择“从训练中选择”。

   “选择训练作业”：系统自动选择上一步创建的训练作业。

   “AI引擎”：系统自动写入该模型的AI引擎，无需修改。

   “推理代码”：系统自动放置推理代码到OBS输出路径，无需修改。

   “部署类型”：默认选择“在线服务”。
3. 参数填写完成后，单击“立即创建”。页面自动跳转至AI应用列表页面，等待创建结果，预计2分钟左右。

   当AI应用的状态变为“正常”时，表示创建成功。

步骤 5：部署为在线服务（CPU）

AI应用创建成功后，可将其部署为在线服务，在部署时可使用CPU资源。

1. 单击AI应用名称左侧的单选按钮，在列表页底部展开“版本列表”，在版本的操作列中单击“部署 > 在线服务”。
2. 在部署页面，参考如下说明填写关键参数。
   - “资源池”：选择“公共资源池”。
   - “选择AI应用及版本”：AI应用来源及版本会自动选择前面创建的AI应用。
   - “计算节点规格”：在下拉框中选择限时免费的CPU资源，如果限时免费资源售罄，建议选择收费CPU资源进行部署。
   - “计算节点个数”，默认设置为“1”。
   - 其他参数可使用默认值。

   说明

   选择CPU资源部署模型会收取少量费用，具体费用以界面信息为准。

   如果需要使用GPU资源部署上线，需要进入模型所在位置，即**步骤3：使用订阅算法创建训练作业**步骤生成的“训练输出”路径，进入“model”目录，打开并编辑“config.json”文件，将“runtime”的配置修改为ModelArts支持的GPU规格，例如"runtime": "tf1.13-python3.6-gpu"。修改完成后，重新执行**导入模型**和**部署为在线服务**的操作。

3. 参数设置完成后，单击“下一步”，确认规格参数，单击“提交”，完成在线服务的部署。
4. 您可以进入“模型部署 > 在线服务”页面，等待服务部署完成，当服务状态变为“运行中”时，表示服务部署成功。预计时长2分钟左右。
5. 在线服务部署完成后，您可以单击操作列的预测，进入服务详情页的“预测”页面。
6. 在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”进行预测。此处提供一个预测样例图供使用。

步骤 6：清除资源

为避免产生不必要的费用，通过此示例学习订阅算法的使用后，建议您清除相关资源，避免造成资源浪费。

- 停止在线服务：在“在线服务”页面，单击对应服务操作列的“停止”。
- 删除训练作业：在“训练作业”页面，单击操作列的“删除”。
- 删除数据：前往OBS，删除数据，然后删除文件夹及OBS桶。

16.2 使用 ModelArts PyCharm 插件调试训练 ResNet50 图像分类模型

本案例介绍如何将本地开发好的MindSpore模型代码，通过PyCharm ToolKit连接到ModelArts进行云上调试和训练。

开始使用样例前，请仔细阅读**准备工作**罗列的要求，提前完成准备工作。本案例的步骤如下所示：

步骤1：安装和登录PyCharm ToolKit

步骤2：使用PyCharm进行本地开发调试

步骤3：使用ModelArts Notebook进行开发调试

步骤4：使用PyCharm提交训练作业至ModelArts

步骤5：清除相应资源

准备工作

- 本地已安装2019.2-2023.2之间（包含2019.2和2023.2）版本的PyCharm专业版工具，推荐Windows版本，社区版或专业版均可，请单击**PyCharm工具下载地址**获取工具并在本地完成安装。
  - 使用PyCharm ToolKit远程连接Notebook开发环境，仅限PyCharm专业版。
  - 使用PyCharm ToolKit提交训练作业，社区版和专业版都支持。
- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 已创建当前使用账号的访问密钥，并获得对应的AK和SK。如果未创建，请参见**创建访问密钥（AK和SK）**。
- 当前账号已完成访问授权的配置。如未完成，请参考**使用委托授权**。

环境说明

- Python 3.7.6
- PyCharm 2023.1.3 (Professional Edition)

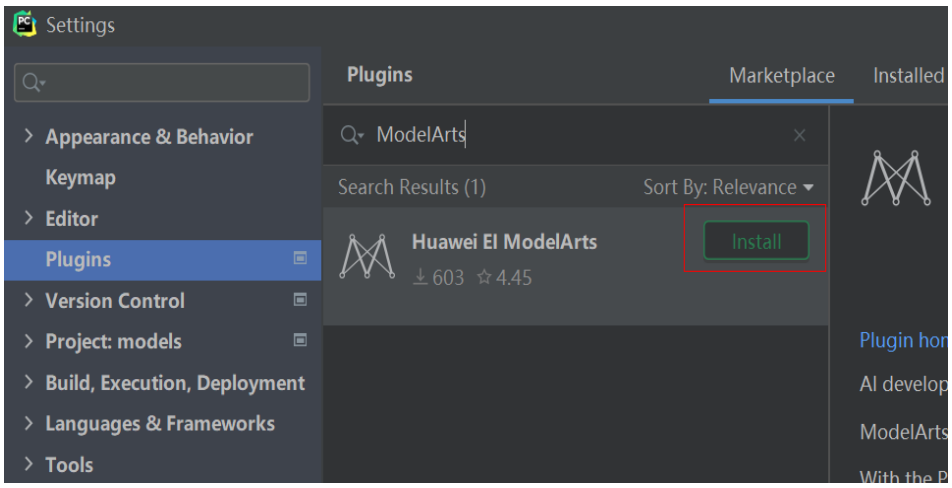
> **说明**
>
> 本案例使用PyCharm版本为PyCharm 2023.1.3 (Professional Edition)，不同版本PyCharm之间部分界面可能不同，仅供参考。

步骤 1：安装和登录 PyCharm ToolKit

1. 安装PyCharm ToolKit。

   在PyCharm中选择“File>Settings>Plugins”，在Marketplace里搜索“ModelArts”，单击“Install”即可完成安装。

   图 16-1 通过 Marketplace 安装

2. 登录PyCharm ToolKit。

   a. 打开“Edit Credential”界面。

   安装完插件后，会在IDE菜单栏出现“ModelArts”，单击后选择“Edit Credential”。

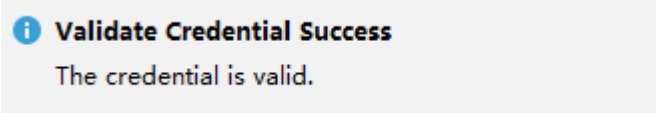
   > **说明**
   >
   > 如果菜单栏中找不到“ModelArts > Edit Credential”，可能是PyCharm版本过高，PyCharm toolkit未适配2023.2之后版本的PyCharm工具。请下载2019.2-2023.2之间（包含2019.2和2023.2）版本的PyCharm专业版工具。

   b. 验证登录信息。

   将创建访问密钥（AK和SK）输入到ToolKit对应位置，单击OK按钮进行登录，出现下图提示即为登录成功。

   如果未创建，请参见**创建访问密钥（AK和SK）**

   图 16-2 成功登录提示

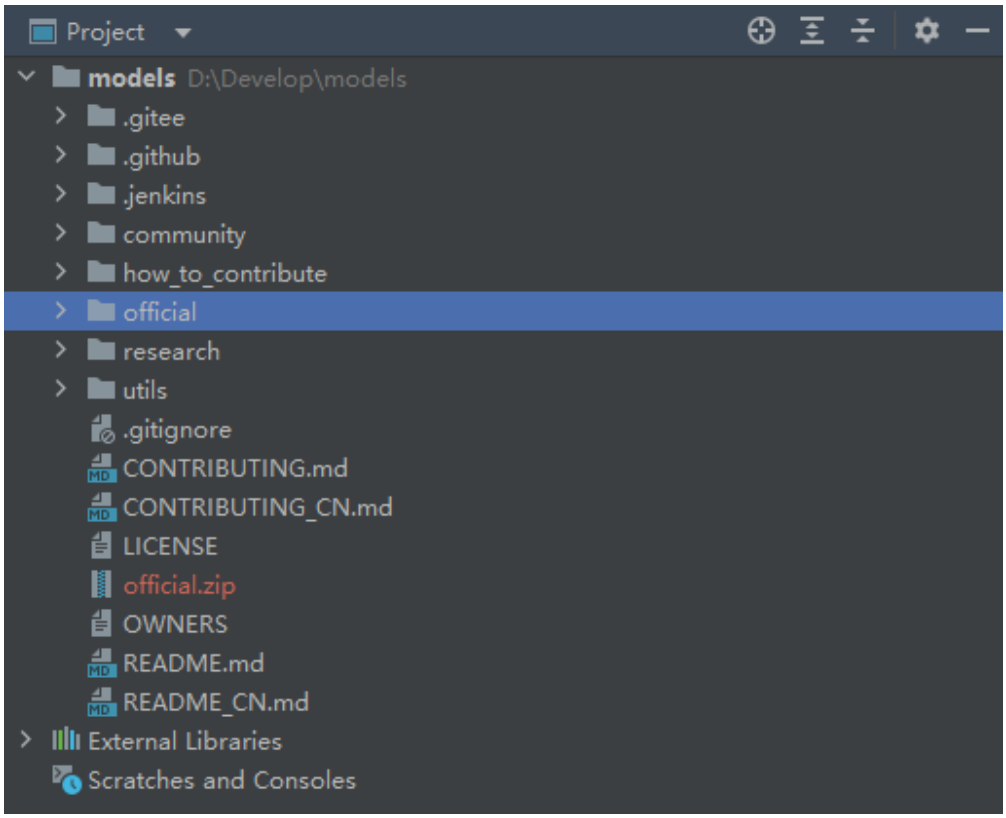
步骤 2：使用 PyCharm 进行本地开发调试

1. 下载代码至本地。

   本案例中，以图像分类模型resnet50模型为例，路径为“./models/official/cv/resnet/”

   ```
   # 在本地电脑Terminal下载代码至本地
   git clone https://gitee.com/mindspore/models.git -b v1.5.0
   ```

   图 16-3 下载代码至本地

2. 配置本地PC开发环境。

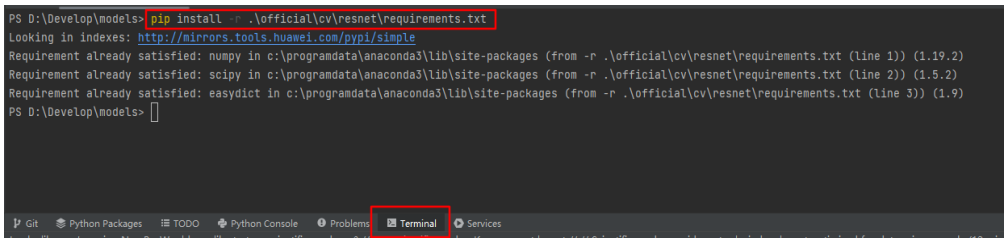
   修改“models/official/cv/resnet/requirements.txt”文件，改为：

   ```
   numpy==1.17.5
   scipy==1.5.4
   easydict==1.9
   ```

   执行pip命令安装：

   ```
   # 在PyCharm的Terminal安装mindspore
   pip install mindspore==1.7.0 --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/repository/pypi/simple
   # 在PyCharm的Terminal安装resnet依赖
   pip install -r .\official\cv\resnet\requirements.txt --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/repository/pypi/simple
   ```

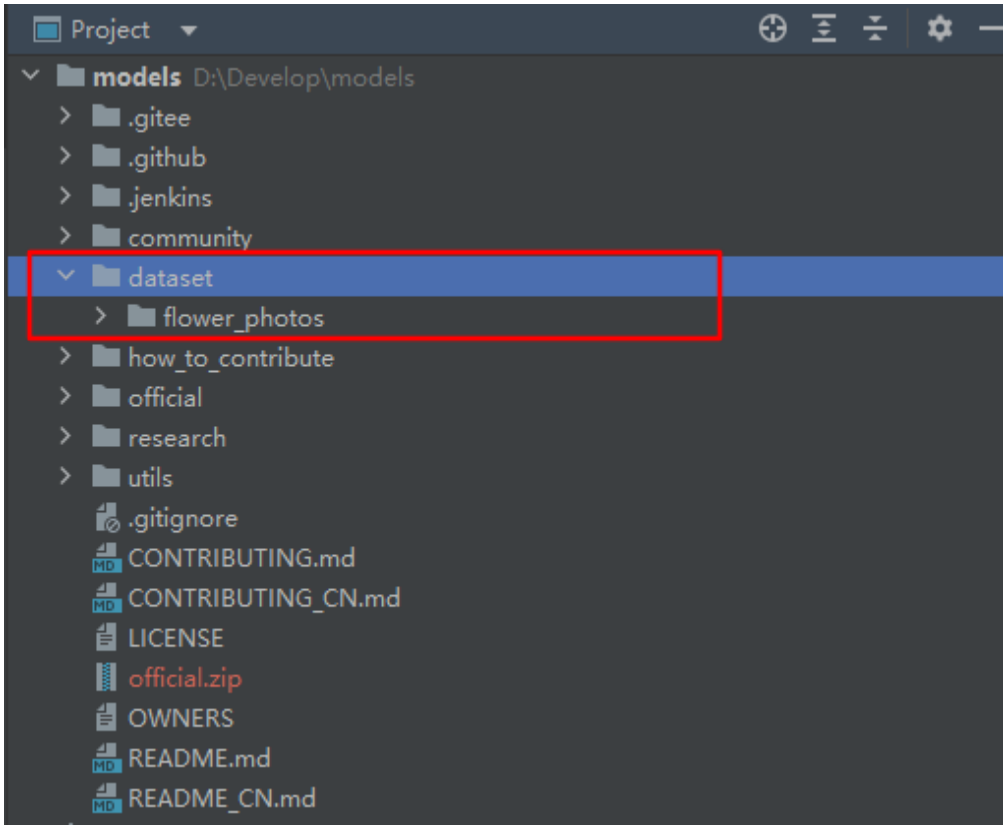
   图 16-4 安装 resnet 依赖

3. 准备数据集。

   本样例使用的数据集为类别数为五类的花卉识别数据集，**下载数据集**并解压数据到工程目录。新建dataset文件夹，将解压后数据集保存在dataset文件夹下。

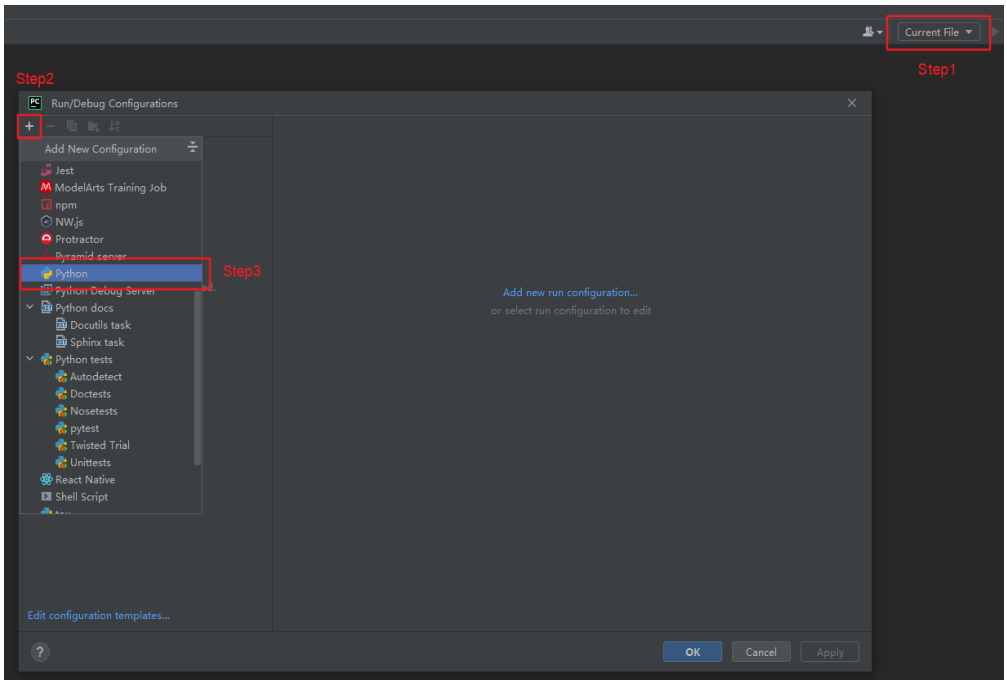
   图 16-5 准备数据集



4. 配置PyCharm解释器和入参。

   单击右上角“Current File”，选择“Edit Configuration”，打开“Run/Debug Configuration”对话框。在对话框中单击“+”，选择“Python”。

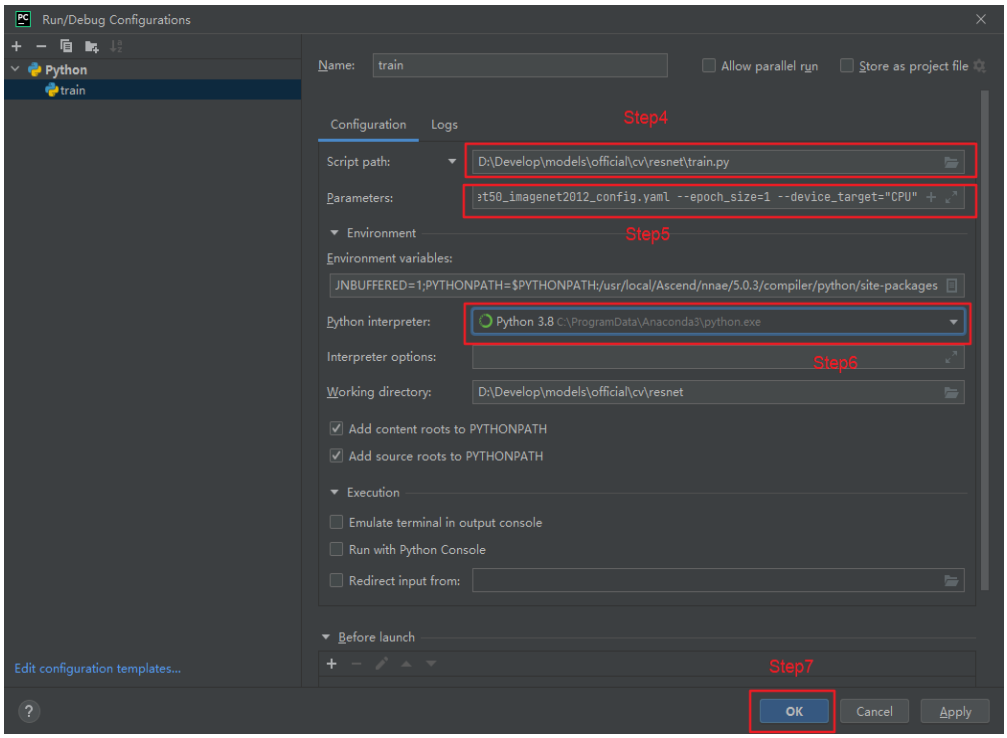
图 16-6 前往 PyCharm 解释器



“Script path”选择train.py文件，“Parameters”命令如下所示，并选择Python解释器，然后单击“OK”：

```
--net_name=resnet50 --dataset=imagenet2012 --data_path=../../../dataset/flower_photos/ --class_num=5 --config_path=./config/resnet50_imagenet2012_config.yaml --epoch_size=1 --device_target="CPU"
```

图 16-7 配置 PyCharm 解释器



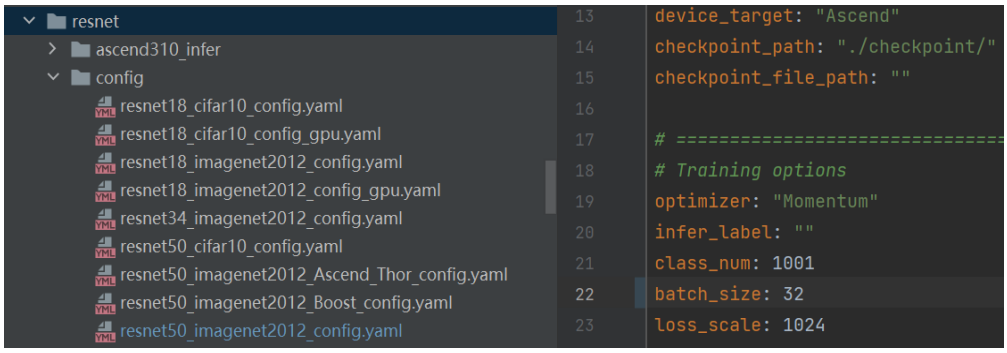
说明

根据README说明文档，配置Parameter参数device_target="CPU"表示CPU环境运行，device_target="Ascend"表示在Ascend环境运行。

5. 本地代码开发调测。

一般本地CPU算力较低并且内存较小，可能出现内存溢出的报错，因此可以把“models/official/cv/resnet/config/resnet50_imagenet2012_config.yaml”的“batch_size”由“256”改为“32”，使得训练作业可以快速运行。

图 16-8 修改 batch_size



AI开发过程中的数据集开发及模型开发是和硬件规格无关的，而且这一部分的开发耗时是最长的，因此可以先在本地PC的CPU环境进行数据集和模型开发调试。

说明

本例中，因为样例代码已经支持在CPU上进行训练，因此用户能够在CPU上完成整个训练流程。如果代码只支持在GPU或者Ascend上训练，那么可能会报错，需要使用Notebook进行云端调试。

设置断点后单击“调试”，可实现代码逐步调试，查看中间变量值。

图 16-9 “调试”按钮

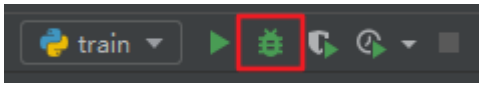
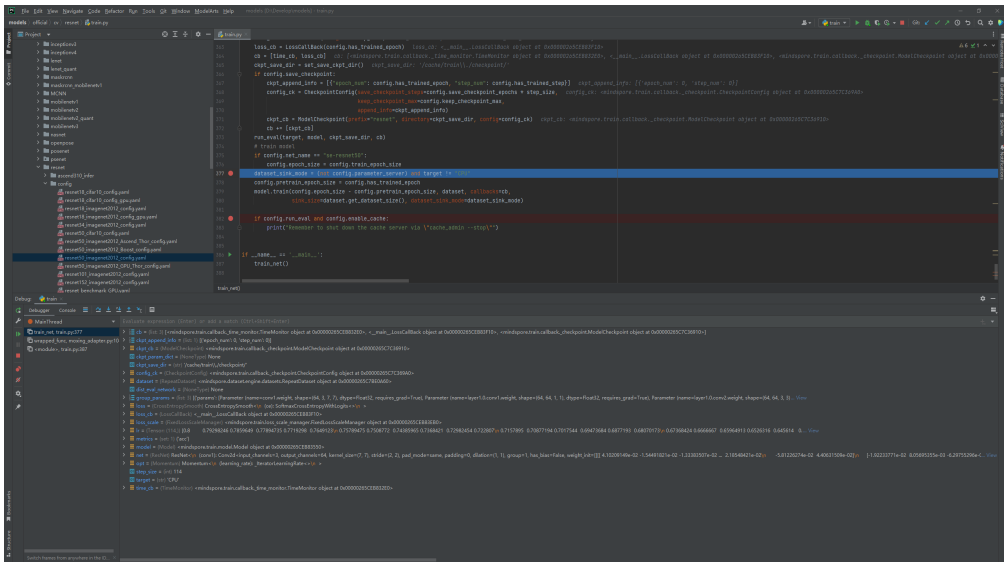


图 16-10 通过设置断点实现代码调试



可单击“运行”按钮，通过日志观察是否能正常训练。

图 16-11 “运行”按钮

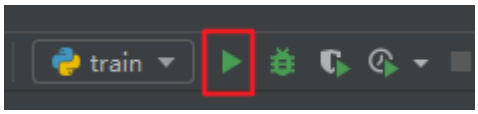
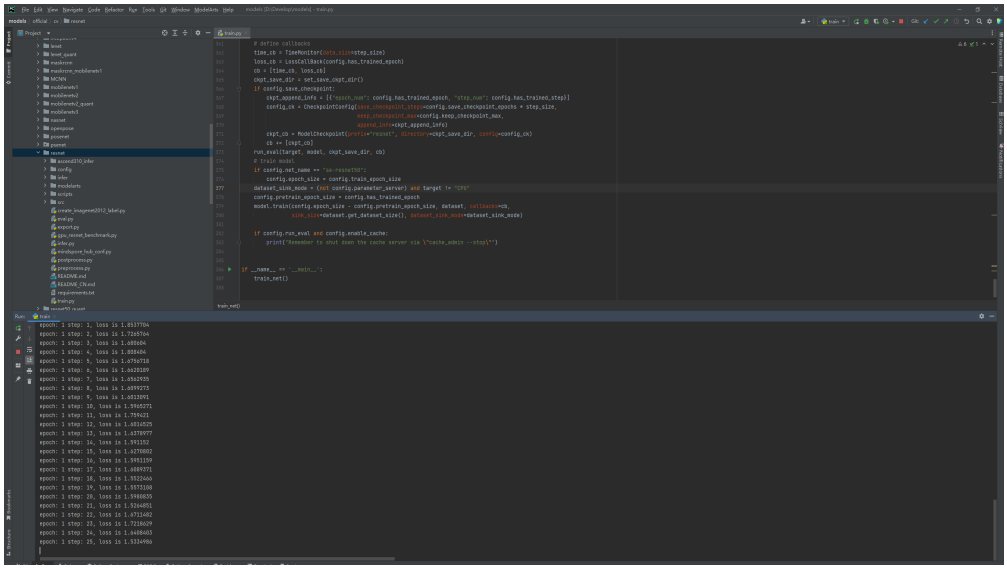


图 16-12 训练日志



步骤 3：使用 ModelArts Notebook 进行开发调试

使用ModelArts Notebook进行开发调试具有如下优势：

- 环境保持一致
- 配置一键完成

- 代码远程Debug
- 资源按需使用

说明

只有PyCharm专业版支持本章节，社区版可以直接跳转至**步骤4：使用PyCharm提交训练作业至ModelArts**完成创建训练作业。

1. 连接Notebook开发环境。
   a. 创建或打开云端Ascend规格的Notebook。创建Notebook详细操作请参见**创建Notebook实例**，Notebook规格相关信息如下所示：

   “镜像”：tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64。

   “资源选择”：公共资源池。

   “类型”：ASCEND。

   “规格”：选Ascend类型的，以界面实际可选值为准。

   “存储配置”：EVS存储。

   “SSH远程开发”：开启。

   “密钥对”：选择已有密钥对，或单击密钥对右侧的“立即创建”创建密钥对。
2. 通过ToolKit连接云端Notebook。
   a. 在IDE菜单栏中选择“ModelArts>Notebook>Remote Config”，在打开的界面中选择要连接的Notebook实例。

   说明

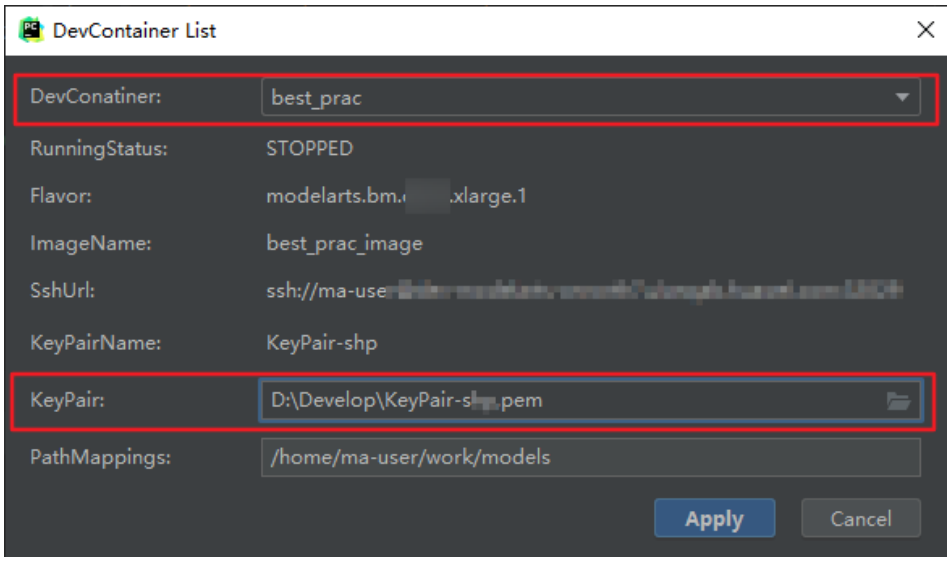
   此处如果看不到Connect to Remote选项，请先参考**创建Notebook实例**章节，创建Notebook实例，并开启该实例的SSH远程开发功能。

   也可能是PyCharm ToolKit的版本不正确，请按照文档要求下载新版本的PyCharm ToolKit。

   下载前请先清除浏览器缓存，如果之前下载过老版本的PyCharm ToolKit，浏览器会有缓存，可能会导致新版本下载失败。

   b. 在KeyPair中选择该Notebook实例对应的密钥，选择完成后，单击Apply进行远程Notebook一键配置，等待一段时间后，会出现重启IDE的确认框，单击确认重启，重启后即可生效。

图 16-13 ToolKit 连接 Notebook 配置界面

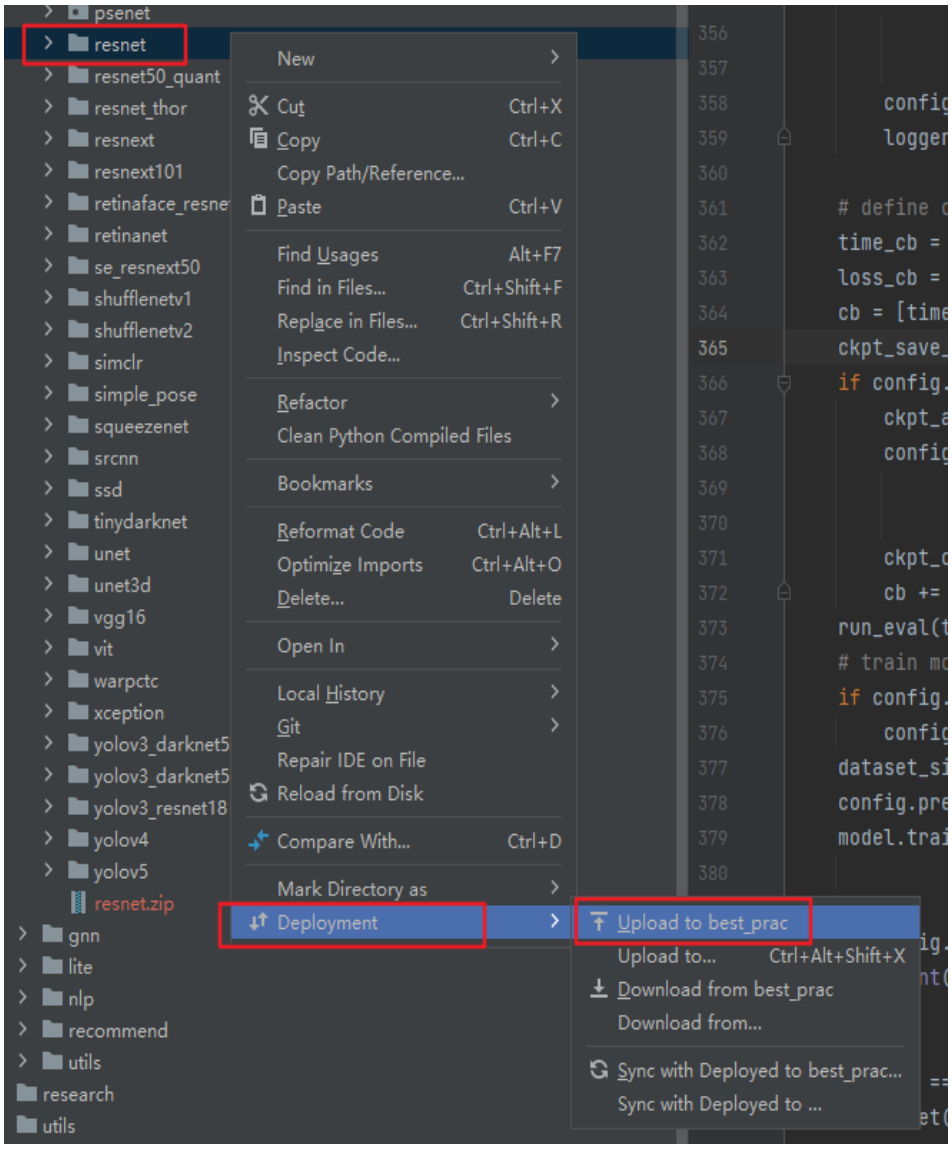


说明

- KeyPair: 需要选择保存在本地的Notebook对应的keypair认证。即创建Notebook时创建的密钥对文件，创建时会直接保存到浏览器默认的下载文件夹中。
- PathMappings: 该参数为本地IDE项目和Notebook对应的同步目录，默认为“/home/ma-user/work/project”，可根据自己实际情况更改。

3. 同步代码和数据至云端Notebook。
   a. 将代码同步至Notebook。
      选择resnet文件夹，右键选择“Deployment>Upload to”上传代码至Notebook。

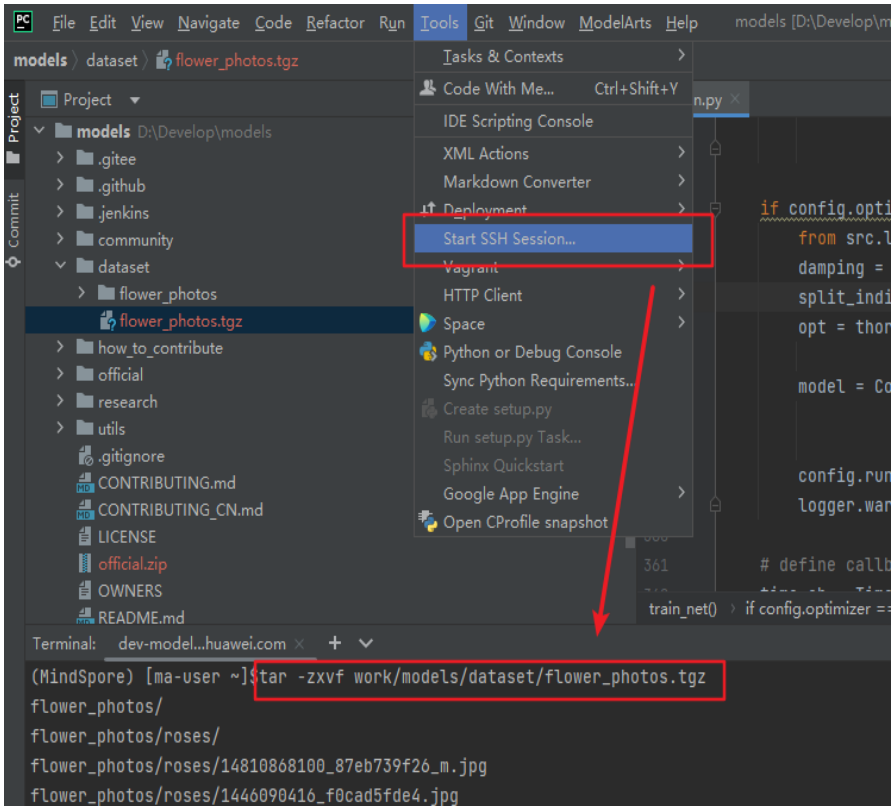
      图 16-14 同步代码至 Notebook



   b. 将数据同步至Notebook。
      - （推荐）方法一：数据集压缩包上传至Notebook后解压
        把数据集压缩包右键选择“Deployment>Upload to”的方式上传至Notebook后，在Notebook中对数据集进行解压操作，解压命令如下：

```
tar -zxvf work/models/dataset/flower_photos.tgz
```

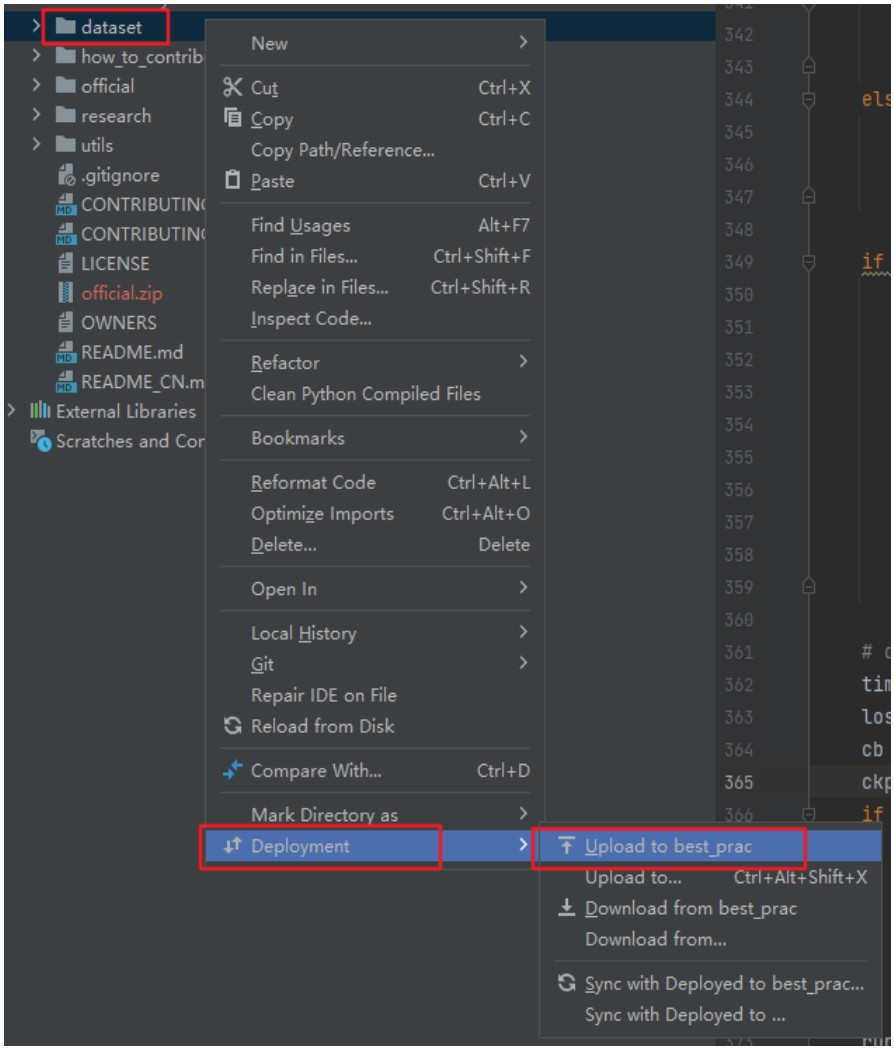

图 16-15 数据集压缩包上传至 Notebook 后解压



- 方法二：文件夹直接上传至Notebook。

  类似上传代码至Notebook，直接上传数据文件夹。（由于本案例数据集中图片数量较多，通过IDE进行上传比较耗时，推荐使用方法一进行上传）

图 16-16 文件夹直接上传至 Notebook



注意

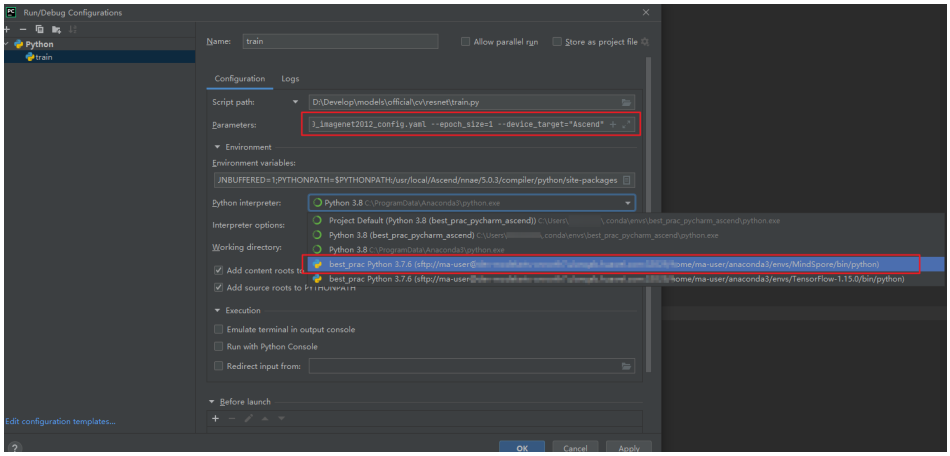
- 当数据集比较大达到数GB时，建议先将数据集先上传至OBS再通过OBS上传至Notebook，PyCharm只适合做小文件的同步上传。
- 调试时建议使用较小的数据集子集，方便数据同步与数据加载。

4. 配置云端Python解释器。

   修改Parameters参数，并选择云端Python解释器。

   ```
   --net_name=resnet50 --dataset=imagenet2012 --data_path=../../../dataset/flower_photos/ --class_num=5 --config_path=./config/resnet50_imagenet2012_config.yaml --epoch_size=1 --device_target="Ascend"
   ```

图 16-17 配置云端 python 解释器

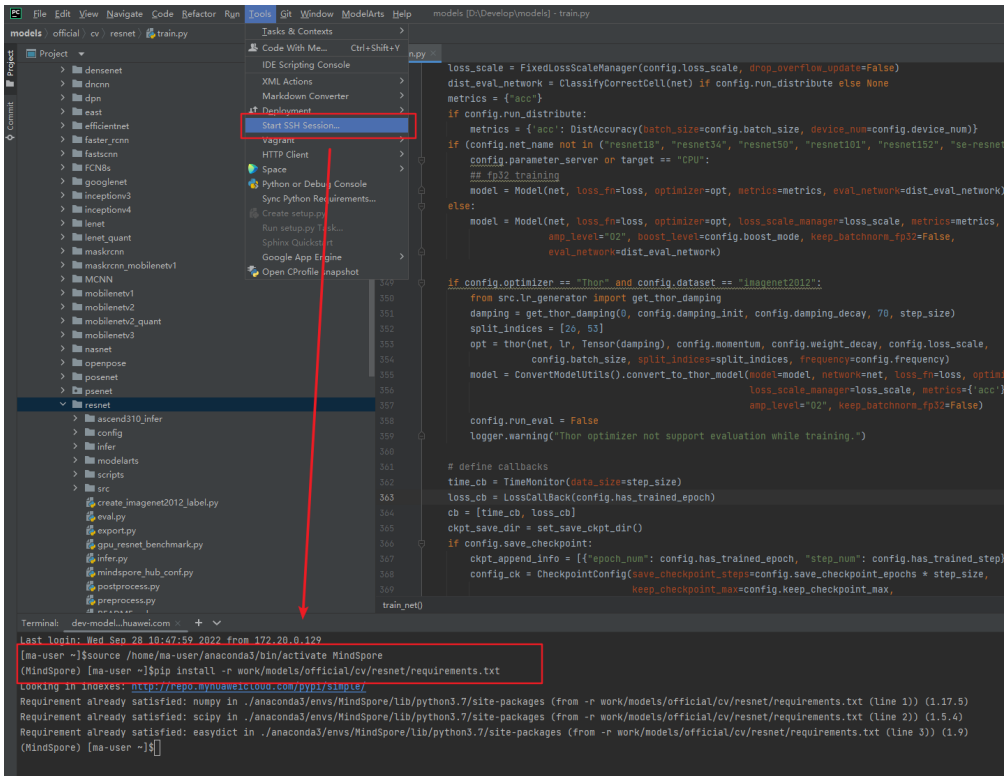


5. 云端Notebook安装依赖。

   打开“Tool>Start SSH Section”，安装依赖软件。

```
# 进入MindSpore环境
source /home/ma-user/anaconda3/bin/activate MindSpore
# 安装resnet依赖
pip install -r work/models/official/cv/resnet/requirements.txt
```

图 16-18 云端 Notebook 安装依赖



6. 云端调试与运行。

   配置完云端的解释器后，PyCharm可以直接使用远端Notebook中的python解释器和硬件规格，满足用户在本地体验到真实的硬件环境并进行全流程的调试和验证。

注意

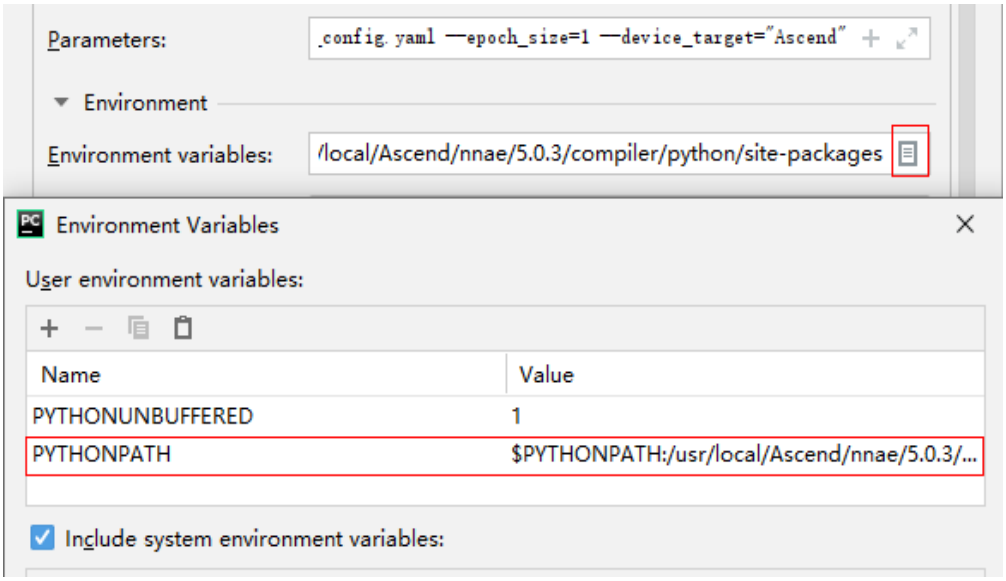
基于Ascend的样例中，可能会抛出异常。

```
ModuleNotFoundError: No module named 'te'
```

原因是：PyCharm的PYTHONPATH会将Notebook中的环境变量中指定的“PYTHONPATH”进行覆盖，因此，还需要将te包所在的路径添加到PyCharm的“PYTHONPATH”中。

te包的路径通过“pip show te”查看，例如te包返回对应的路径为：“/usr/local/Ascend/nnae/5.0.3/compiler/python/site-package”，则“PYTHONPATH”对应的“Value”为“$PYTHONPATH:/usr/local/Ascend/nnae/5.0.3/compiler/python/site-package”

图 16-19 将 te 包所在的路径添加到 PyCharm 的 PYTHONPATH 中



7. 保存开发环境镜像。

   成功完成Notebook调测后，此时的Notebook已经包含了模型训练所有的依赖环境，因此可以将已经调测完成的开发环境保存成一个镜像，选择“Notebook>更多>保存镜像”。此时Notebook会冻结，需要等待几分钟（只需要保存一次）。

   保存后的镜像可以在“ModelArts>镜像管理”中进行查看，对应完整的镜像名称为“详情->SWR地址”。

图 16-20 查看保存后的镜像

说明

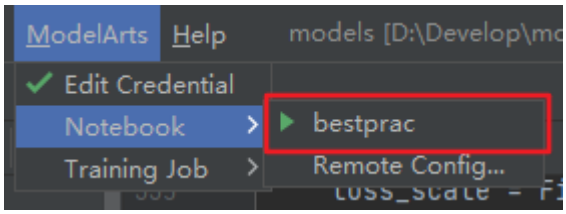
Notebook在代码调试完成及保存镜像后就可以关闭了，减少资源浪费。

8. 连接、停止、启动和断开Notebook实例。

- 连接Notebook实例。

  当Notebook实例为绿色三角形状态时，表示该实例运行中（但未与PyCharm连接）。此时单击该实例名称，实例会变为绿色勾状态，表示PyCharm已与实例连接成功。

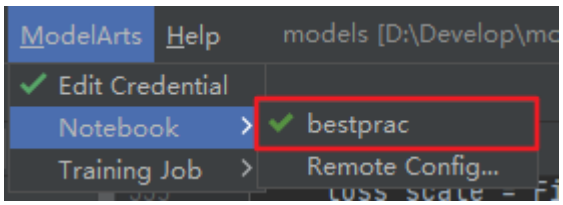
  图 16-21 实例运行中状态

- 停止Notebook实例。

  当Notebook实例为绿色勾状态时，表示该实例运行中且与PyCharm连接成功。此时单击该实例名称，实例会变为黄色感叹号状态，表示停止Notebook实例。

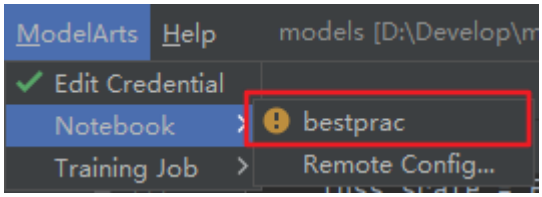
  图 16-22 实例运行中且与 PyCharm 连接成功状态

- 启动Notebook实例。

  当Notebook实例为黄色感叹号状态时，表示该实例已停止。此时单击该实例名称，实例会变为绿色勾状态，表示启动Notebook实例且与PyCharm连接成功（默认启动时间为4小时）。

  图 16-23 实例已停止状态

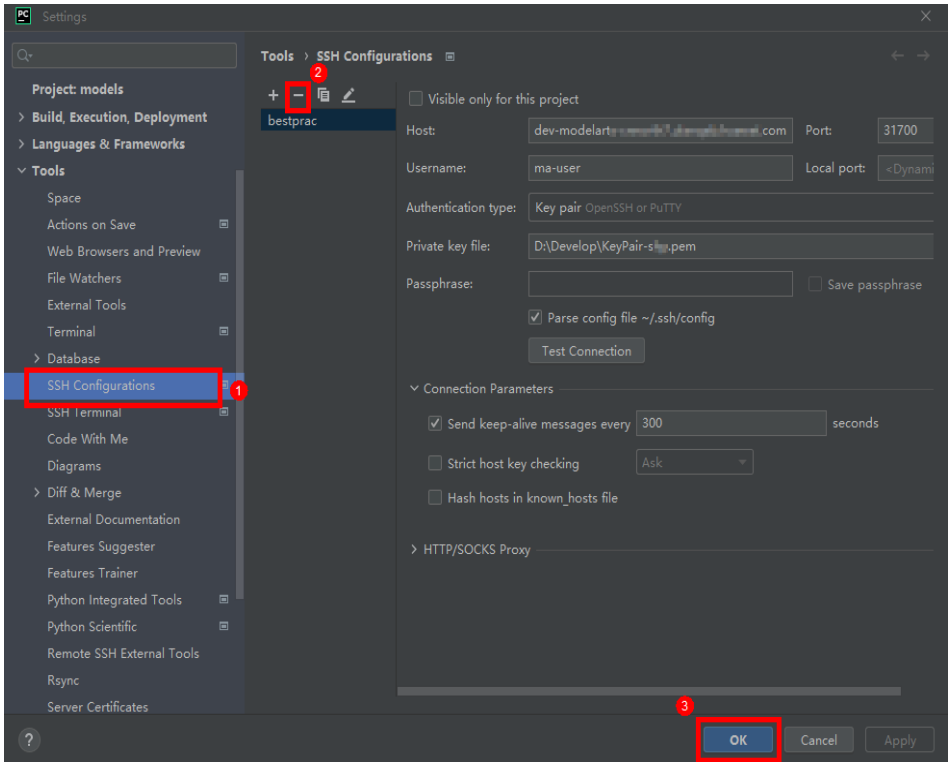
- 断开PyCharm ToolKit中的Notebook实例SSH连接。

  选择“File>Settings>Tool>SSH Configurations”，单击需要断开的实例，选择“-”，单击“OK”，则IDE菜单栏“ModelArts>Notebook”中的Notebook实例连接断开。

⚠ 注意

该步骤会使Notebook实例不在PyCharm ToolKit中呈现，但Notebook实例仍然存在于控制台。如果想删除Notebook实例以释放资源，请登录ModelArts管理控制台，在Notebook管理页面进行删除。

图 16-24 断开 PyCharm ToolKit 中的 Notebook 实例 SSH 连接



步骤 4：使用 PyCharm 提交训练作业至 ModelArts

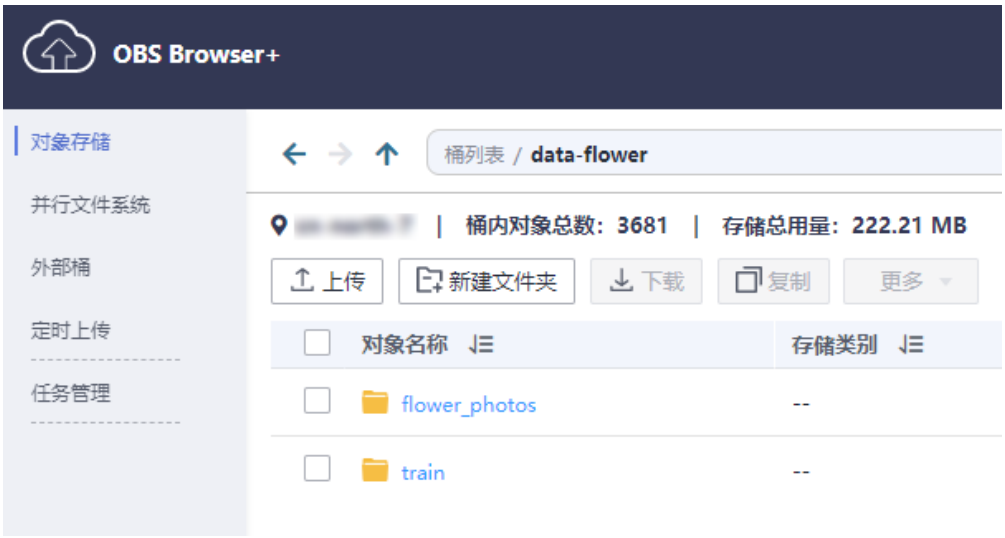
ModelArts训练平台提供了海量的算力规格和训练优化，支持将本地调试好的代码以及之前保存的开发环境镜像直接在PyCharm中提交训练作业。

1. 创建OBS桶并上传数据。

   由于训练作业是在ModelArts端运行，因此需要把训练数据和训练代码上传至云端Notebook。可借助OBS Browser+把下载好的训练数据上传至OBS，具体安装步骤请见**安装OBS Browser+**。

   新建data-flower桶，把训练数据flower_photos文件夹通过OBS Browser+上传至对应的OBS，并新建train文件夹用来存放训练作业相关数据。

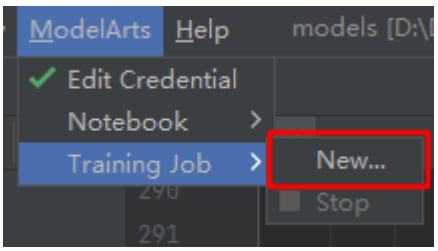
图 16-25 上传数据至 OBS



2. 创建训练作业。

   在IDE菜单栏选择“ModelArts>Training Job>New...”创建训练作业。

图 16-26 创建训练作业



创建训练作业界面各参数名称及含义如下表所示。

表 16-1 参数名称及含义

参数名称	含义
JobName	训练作业的名称，默认为当前的时间。
AI Engine	训练引擎，这里选择“mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64”
Boot File Path	本地训练启动代码。
Code Directory	本地代码目录
Image Path(optional)	可选项，输入自定义镜像swr路径地址（使用的自定义镜像和预置的训练镜像引擎一致）
Data OBS Path	OBS上的数据集路径（需要提前把数据上传到OBS中）
Training OBS Path	OBS路径（该路径必须是存在的），用于保存代码和训练模型及日志的输出

参数名称	含义
Running Parameters	训练脚本接收的参数。
Specifications	计算规格，这里选择Ascend类型的，以界面实际可选值为准。
Compute Node	节点数（单机训练默认为1）

PyCharm中支持两种方式创建训练作业：使用预置镜像训练作业、自定义镜像创建训练作业。

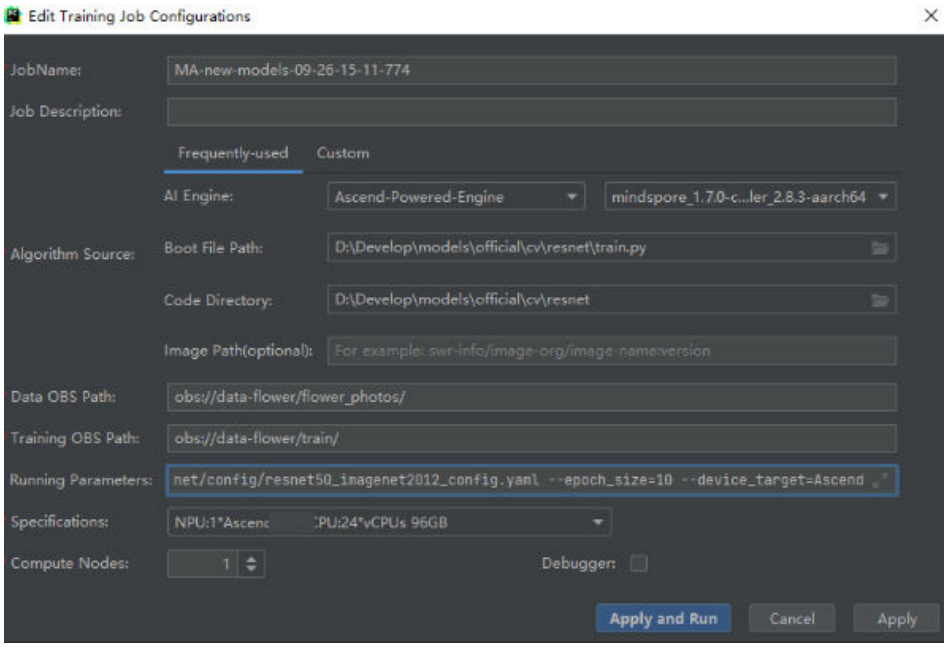
- 使用预置镜像创建训练作业。

  在RunningParameters中填入如下训练参数，其余参数按实际路径填写。

  ```
  --net_name=resnet50 --dataset=imagenet2012 --enable_modelarts=True --class_num=5 --config_path=/home/ma-user/modelarts/user-job-dir/resnet/config/resnet50_imagenet2012_config.yaml --epoch_size=10 --device_target=Ascend
  ```

  填写完训练作业参数后，单击“Apply and Run”即完成训练作业创建。

  图 16-27 使用预置镜像创建训练作业

- 使用自定义镜像创建训练作业。

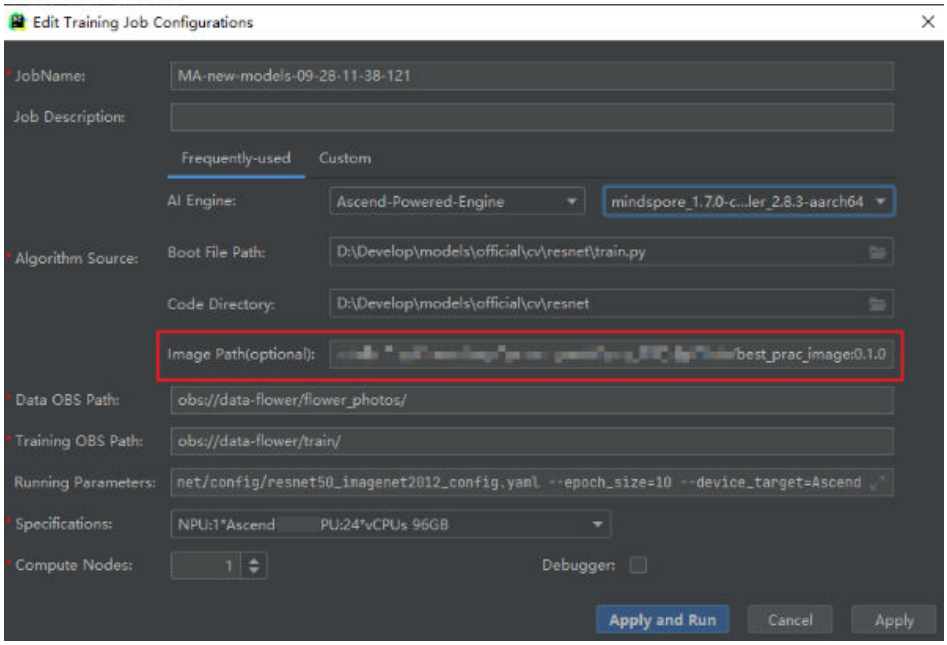
  使用自定义镜像创建训练作业和使用预置镜像创建训练作业的差别，在于Image Path处填入了自定义镜像的地址。填写完训练作业参数后，单击“Apply and Run”即完成训练作业创建。

  说明

  在选择AI Engine预置镜像时，需要和自定义镜像保持一致，该设置的作用为通过预置镜像的启动命令启动自定义镜像。

  例如自定义镜像中用到Mindspore，则预置镜像中可选择包含Mindspore的镜像。

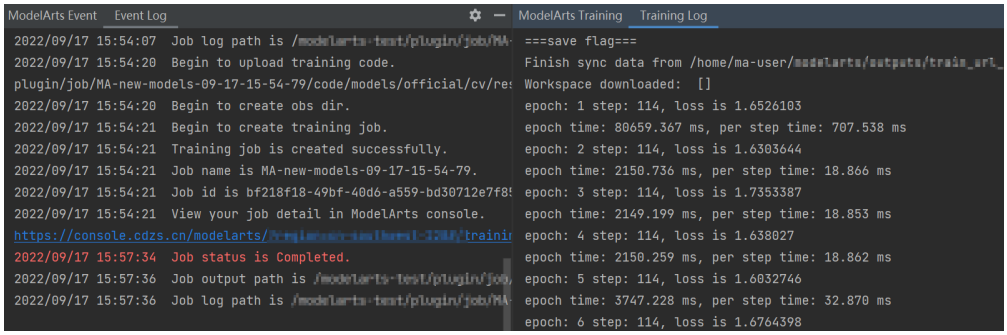
图 16-28 使用自定义镜像创建训练作业



3. 查看训练日志。

   在单击“Apply and Run”按钮后，训练的日志可以在PyCharm窗口中实时展示。也可以单击Event Log中的控制台链接，转调到网页端中查看训练日志。

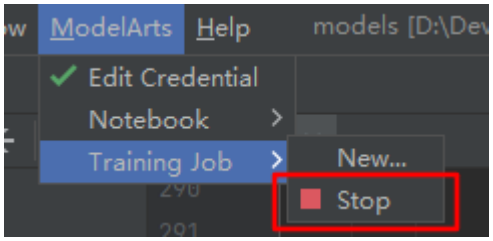
图 16-29 在 PyCharm 中查看训练日志



4. 终止训练作业。

   如果想要在中途终止训练，可以在PyCharm中单击“ModelArts>Training Job>Stop”，或者直接在网页端单击终止。

图 16-30 终止训练作业



步骤 5：清除相应资源

为避免产生不必要的费用，在完成试用后，建议您删除相关资源，如在线服务、训练作业及其OBS目录。

- 停止Notebook：在“Notebook”页面，单击对应实例操作列的“停止”。
- 在PyCharm菜单栏中，选择“ModelArts > Stop Training Job”停止此训练作业。
- 进入OBS管理控制台，删除创建的OBS桶。先逐个删除桶内文件夹和文件，再执行删除桶的操作。

16.3 示例：从 0 到 1 制作自定义镜像并用于训练（PyTorch +CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备训练脚本并上传至OBS**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**
6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表16-2**所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**。

请确保您使用的OBS与ModelArts在同一区域。

表 16-2 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/pytorch/demo-code/”	用于存储训练脚本文件。
“obs://test-modelarts/pytorch/log/”	用于存储训练日志文件。

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本“pytorch-verification.py”文件，并上传至OBS桶的“obs://test-modelarts/pytorch/demo-code/”文件夹下。

“pytorch-verification.py”文件内容如下：

```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用Ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04

- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤 。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以执行以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 执行如下命令确认Docker Engine版本。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   ...
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 准备可用的pip源文件pip.conf 。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi ，也可以查看“pip.conf”文件内容。

5. 下载“torch*.whl ”文件。

   在网站“https://download.pytorch.org/whl/torch_stable.html”搜索并下载如下whl文件。

   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   说明

   “+”符号的URL编码为“%2B”，在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

   例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

6. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。


```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

8. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim和curl 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
```



```
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

9. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像pytorch:1.8.1-cuda11.1。

```
docker build . -t pytorch:1.8.1-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:1.8.1-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
#此处以华为云cn-north-4为例
sudo docker tag pytorch:1.8.1-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```

   b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
#此处以华为云cn-north-4为例
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```

6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。

“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”

- 镜像地址：**Step5 上传镜像至SWR服务**中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”
- 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
- 启动命令：“/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
- 资源池：选择公共资源池
- 类型：选择GPU或者CPU规格。
- 永久保存日志：打开
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

16.4 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如表16-3所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见创建桶。

请确保您使用的OBS与ModelArts在同一区域。

表 16-3 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/mpi/demo-code/”	用于存储MPI启动脚本与训练脚本文件。
“obs://test-modelarts/mpi/log/”	用于存储训练日志文件。

Step2 准备脚本文件并上传至 OBS 中

准备本案例所需的MPI启动脚本run_mpi.sh文件和训练脚本mpi-verification.py文件，并上传至OBS桶的“obs://test-modelarts/mpi/demo-code/”文件夹下。

- MPI启动脚本run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
```



```
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-
p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
```



```
    -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
    "$@"

RET_CODE=$?

if [ $RET_CODE -ne 0 ]; then
    echo "[run_mpi] exec command failed, exited with $RET_CODE"
else
    echo "[run_mpi] exec command successfully, exited with $RET_CODE"
fi

# stop 1...N worker by killing the sleep proc
sed -i '1d' ${MY_HOME}/hostfile
if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
    echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

    sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

    mpirun \
    --hostfile ${MY_HOME}/hostfile \
    --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
    -x PATH -x LD_LIBRARY_PATH \
    pkill sleep \
    > /dev/null 2>&1
fi

echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
echo "[run_mpi] the training log is in worker-0"
sleep 365d
echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

说明

“run_mpi.sh”脚本需要以LF作为换行符。使用CRLF作为换行符会导致训练作业运行失败，日志中会打印“$'\r': command not found”的错误信息。

- 训练脚本mpi-verification.py文件内容如下：

```
import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

    # https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
    print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
    print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
    print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04

- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 下载Miniconda3安装文件。

   使用地址 https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应 python 3.7.13）。

5. 下载openmpi 3.0.0安装文件。

   使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载 horovod v0.22.1已经编译好的openmpi 3.0.0文件。

6. 将上述Miniconda3安装文件、openmpi 3.0.0文件放置在context文件夹内，context文件夹内容如下。

   ```
   context
   ├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
   └── openmpi-3.0.0-bin.tar.gz
   ```

7. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

   ```
   # 容器镜像构建主机需要连通公网

   # 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
   #
   # https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
   # require Docker Engine >= 17.05
   #
   # builder stage
   FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

   # 基础容器镜像的默认用户已经是 root
   # USER root

   # 复制 Miniconda3 (python 3.7.13) 安装文件到基础容器镜像中的 /tmp 目录
   COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

   # 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
   # https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
   RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3
   ```



```
# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim / curl / net-tools / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

关于Dockerfile文件编写的更多指导内容参见 **Docker 官方文档**。

8. 确认已创建完成Dockerfile 文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mpi:3.0.0-cuda11.1。
```
docker build . -t mpi:3.0.0-cuda11.1
```
构建过程结束时出现如下构建日志说明镜像构建成功。
```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
   “swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在ModelArts管理控制台，左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mpi/demo-code/”
   - 启动命令：bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py
   - 环境变量：添加“MY_SSHD_PORT = 38888”
   - 资源池：选择公共资源池
   - 类型：选择GPU规格
   - 计算节点个数：选择“1”或“2”

- 永久保存日志：打开
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mpi/log/”

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

   计算节点个数选择为2，训练作业也可以运行。

16.5 使用 ModelArts Standard 一键完成商超商品识别模型部署

ModelArts的AI Gallery中提供了大量免费的模型供用户一键部署，进行AI体验学习。

本文以“商超商品识别”模型为例，完成从AI Gallery订阅模型，到ModelArts一键部署为在线服务的免费体验过程。

“商超商品识别”模型可以识别81类常见超市商品（包括蔬菜、水果和饮品），并给出置信度最高的5类商品的置信度得分。

步骤 1：准备工作

- 已注册华为账号并开通华为云，进行了实名认证，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在“访问授权”页面，选择需要授权的“授权对象类型”，选择新增委托及其对应的权限“普通用户”，并勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

  c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 2：订阅模型

“商超商品识别”的模型共享在AI Gallery中。您可以前往AI Gallery，免费订阅此模型。

1. 单击案例链接**商超商品识别**，进入模型详情页。
2. 完成模型订阅。

   在模型详情页，单击“订阅”，阅读并勾选同意《数据安全与隐私风险承担条款》和《华为云AI Gallery服务协议》，单击“继续订阅”。订阅模型完成后，页面的“订阅”按钮显示为“已订阅”。

3. 从模型详情页进入ModelArts控制台的订阅列表。

   在模型详情页，单击“前往控制台”。在弹出的“选择云服务区域”页面选择ModelArts所在的云服务区域，单击“确定”跳转至ModelArts控制台的“AI应用 > 订阅应用”页面。

4. 在“订阅应用”列表，单击“版本数量”，在右侧展开版本列表，当订阅模型的版本列表的状态显示为“就绪”时表示模型可以使用。

步骤 3：使用订阅模型部署在线服务

模型订阅成功后，可将此模型部署为在线服务

1. 在展开的版本列表中，单击“部署 > 在线服务”跳转至部署页面。
2. 在部署页面，参考如下说明填写关键参数。

   “名称”：自定义一个在线服务的名称，也可以使用默认值，此处以“商超商品识别服务”为例。

   “资源池”：选择“公共资源池”。

   “AI应用来源”和“选择AI应用及版本”：会自动选择订阅模型。

   “计算节点规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

   其他参数可使用默认值。

   说明

   如果限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。

3. 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
4. 任务提交成功后，单击“查看任务详情”，等待服务状态变为“运行中”时，表示服务部署成功。预计时长4分钟左右。

步骤 4：预测结果

1. 在线服务部署完成后，单击“预测”页签。
2. 在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”查看预测结果。此处提供一个样例图片供预测使用。

   说明

   本案例中使用的订阅模型可以识别**81类**常见超市商品，模型对预测图片有一定范围和要求，不满足条件的图片会影响预测结果的准确性。

步骤 5：清理资源

体验结束后，建议暂停或删除服务，避免占用资源，造成资源浪费。

- 停止在线服务：在“在线服务”列表，单击对应服务操作列的“更多 > 停止”。
- 删除在线服务：在“在线服务”列表，单击对应服务操作列的“更多 > 删除”。

16.6 专属资源池训练

16.6.1 资源选择推荐

不同AI模型训练所需要的数据量和算力不同，在训练时选择合适存储及训练方案可提升模型训练效率与资源性价比。ModelArts支持单机单卡、单机多卡和多机多卡的训练场景，满足不同AI模型训练的要求。针对第一次使用ModelArts的用户，本文提供端到端案例指导，帮助您快速了解如何在ModelArts上选择合适的训练方案并进行模型训练。

针对不同的数据量和算法情况，推荐以下训练方案：

- 单机单卡：小数据量（1G训练数据）、低算力场景（1卡Vnt1），存储方案使用“OBS的并行文件系统（存放数据和代码）”。
- 单机多卡：中等数据量（50G左右训练数据）、中等算力场景（8卡Vnt1），存储方案使用“SFS（存放数据和代码）”。
- 多机多卡：大数据量（1T训练数据）、高算力场景（4台8卡Vnt1），存储方案使用“SFS（存放数据）+普通OBS桶（存放代码）”，采用分布式训练。

表 16-4 不同场景所需服务及购买推荐

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
单机单卡	按需购买。（并行文件系统）	×	免费。	免费。	包月购买。	免费。	×	按需购买。
单机多卡	×	包月购买。（HPC型 500G）	免费。	免费。	包月购买。	免费。	包月购买。（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
多机多卡	按需购买。（普通OBS桶）	包月购买。（HPC型500G）	免费。	免费。	包月购买。	免费。	包月购买。（建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

表 16-5 开源数据集训练效率参考

算法及数据	资源规格	Epoch数	运行时长（hh:mm:ss）
算法：PyTorch官方针对ImageNet的样例 数据：ImageNet分类数据子集	1机1卡Vnt1	10	0:05:03
算法：YOLOX 数据：COCO2017	1机1卡Vnt1	10	03:33:13
	1机8卡Vnt1	10	01:11:48
	4机8卡Vnt1	10	0:36:17
算法：Swin-Transformer 数据：ImageNet21K	1机1卡Vnt1	10	197:25:03
	1机8卡Vnt1	10	26:10:25
	4机8卡Vnt1	10	07:08:44

表 16-6 训练各步骤性能参考

步骤	说明	时长
镜像下载	首次下载镜像的时间（25G）。	8分钟
资源调度	点创建训练任务开始到变成运行中的时间（资源充足、镜像已缓存）。	20秒
训练列表页打开	已有50条训练作业，单击训练模块后的时间。	6秒

步骤	说明	时长
日志加载	作业运行中，已经输出1兆的日志文本，单击训练详情页面需要多久加载出日志。	2.5秒
训练详情页	作业运行中，没有用户日志情况下，在ModelArts控制台主页面单击训练详情页面后加载页面内容。	2.5秒
JupyterLab页面	进入JupyterLab页面后加载页面内容。	0.5秒
Notebook列表页	已有50个Notebook实例，在ModelArts控制台主页面单击开发环境后的时间。	4.5秒

说明

镜像下载时间受节点规格、节点硬盘类型（高IO/普通IO）、是否SSD等因素影响，以上数据仅供参考。

16.6.2 步骤总览

单机单卡

1. 资源购买：
   a. **购买对象存储服务OBS**
   b. **购买容器镜像服务SWR**
   c. **创建网络**
   d. **购买ModelArts专属资源池**
2. 基本配置：
   a. **权限配置**
   b. **obsutils安装和配置**
   c. （可选）**工作空间配置**
3. 训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据和算法至OBS（首次使用时需要）**
   d. **使用Notebook进行代码调试**
   e. **创建训练任务**

单机多卡

1. 资源购买：
   a. **购买虚拟私有云VPC**

b. **购买弹性文件服务SFS**
c. **购买容器镜像服务SWR**
d. **创建网络**
e. **购买ModelArts专属资源池**
f. **购买弹性云服务器ECS**

2. 基本配置：
   a. **权限配置**
   b. **专属资源池VPC打通**
   c. **ECS服务器挂载SFS Turbo存储**
   d. （可选）**工作空间配置**
3. 训练：
   a. **上传数据和算法至SFS（首次使用时需要）**
   b. **使用Notebook进行代码调试**
   c. **创建训练任务**

多机多卡

1. 资源购买：
   a. **购买虚拟私有云VPC**
   b. **购买弹性文件服务SFS**
   c. **购买对象存储服务OBS**
   d. **购买容器镜像服务SWR**
   e. **创建网络**
   f. **购买ModelArts专属资源池**
   g. **购买弹性云服务器ECS**
2. 基本配置：
   a. **权限配置**
   b. **专属资源池VPC打通**
   c. **ECS服务器挂载SFS Turbo存储**
   d. **在ECS中创建ma-user和ma-group**
   e. **obsutils安装和配置**
   f. （可选）**工作空间配置**
3. 训练：
   a. **上传数据至OBS（首次使用时需要）**
   b. **上传算法至SFS**
   c. **创建训练任务**

16.6.3 资源购买

购买弹性文件服务 SFS

弹性文件服务默认为按需计费，即按购买的存储容量和时长收费。您也可以购买包年包月套餐，提前规划资源的使用额度和时长。在欠费时，您需要及时（15天之内）续费以避免您的文件系统资源被清空。SFS购买指导请参考**如何购买弹性文件服务？**。

购买容器镜像服务 SWR

容器镜像服务分为企业版和共享版。

共享版计费项包括存储空间和流量费用，目前均免费提供给您。

企业版当前仅支持按需计费模式，公测期间，可免费使用。

说明

上传镜像前需要创建组织，创建步骤请参考**创建组织**。

购买对象存储服务 OBS

对象存储服务提供按需计费和包年包月两种计费模式，用户可以根据实际需求购买OBS服务。OBS服务支持以下两种存储方式，单机单卡场景使用文件系统，多机多卡场景使用普通OBS桶。

- **创建普通OBS桶**
- **创建并行文件系统**

购买数据加密服务 DEW

在使用Notebook进行代码调试时，如果要开启“SSH远程开发”功能，需要选择已有密钥对。密钥对可免费创建，您可通过管理控制台创建密钥对，操作指导请参考**如何创建密钥对？**

购买虚拟私有云 VPC

虚拟私有云可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

购买弹性云服务器 ECS

如果您需要在服务器上部署相关业务，较之物理服务器，弹性云服务器的创建成本较低，并且可以在几分钟之内快速获得基于云服务平台的弹性云服务器设施，并且这些基础设施是弹性的，可以根据需求伸缩。操作指导请参考**自定义购买ECS**。

说明

购买时需注意，ECS需要和SFS买到同一个VPC才能挂载SFS存储。

购买 ModelArts 专属资源池

提供独享的计算资源，可用于Notebook、训练作业、部署模型。专属资源池不与其他用户共享，更加高效。在使用专属资源池之前，您需要先创建一个专属资源池，操作指导请参考**创建专属资源池**。

说明

创建一个专属资源池前需要先创建网络，创建网络指导可参考**创建网络**。

购买 Notebook 存储

使用Notebook代码调试时，需要创建Notebook实例，如果创建时选择“云硬盘EVS”作为存储位置，会创建云硬盘EVS。

磁盘规格默认5GB，从Notebook实例创建成功开始，直至实例删除成功，磁盘每GB按照规定费用收费。

说明

云硬盘EVS会在创建Notebook实例时自动购买，无需用户单独创建。

16.6.4 基本配置

16.6.4.1 权限配置

权限列表

为了便于理解权限相关内容，建议先阅读**ModelArts权限管理基本概念**。

表 16-7 服务授权列表

待授权的服务	适用场景
ModelArts	授予子用户使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。
	如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
SFS弹性文件服务	弹性文件服务SFS Turbo的所有权限。使用SFS服务时需要配置。
ECS弹性云服务器	弹性云服务器所有权限。使用ECS服务时需要配置。
SWR容器镜像仓库	容器镜像仓库所有权限。使用SWR服务时需要配置。同时，还需开通SWR组织权限。
VPC虚拟私有云	子用户在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。

待授权的服务	适用场景
DEW密钥管理服务	当子用户使用ModelArts Notebook的SSH远程功能时，需要配置子用户密钥管理服务的使用权限。
OBS对象存储服务	具有对象存储服务（OBS）查看桶列表、获取桶元数据、列举桶内对象、查询桶位置、上传对象、获取对象、删除对象、获取对象ACL等对象基本操作权限。

16.6.4.1.1 配置 IAM 权限

1. 使用华为云主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。
2. 创建自定义策略。
   a. 使用华为云主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。
   b. 在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy1”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。
   c. 自定义策略“Policy1”的具体内容如下，可以直接复制粘贴。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Action": [
        "modelarts:*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "modelarts:pool:create",
        "modelarts:pool:update",
        "modelarts:pool:delete"
      ],
      "Effect": "Deny"
    },
    {
      "Action": [
        "sfsturbo:*:*",
        "vpc:*:*",
        "dss:*:get",
        "dss:*:list"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecs:*:*",
        "evs:*:get",
        "evs:*:list",
        "evs:volumes:create",
        "evs:volumes:delete",
        "evs:volumes:attach",
        "evs:volumes:detach",
        "evs:volumes:manage",
        "evs:volumes:update",
        "evs:volumes:use",
        "evs:volumes:uploadImage",
        "evs:snapshots:create",
```

```
                "vpc:*:get",
                "vpc:*:list",
                "vpc:networks:create",
                "vpc:networks:update",
                "vpc:subnets:update",
                "vpc:subnets:create",
                "vpc:ports:*",
                "vpc:routers:get",
                "vpc:routers:update",
                "vpc:securityGroups:*",
                "vpc:securityGroupRules:*",
                "vpc:floatingIps:*",
                "vpc:publicIps:*",
                "ims:images:create",
                "ims:images:delete",
                "ims:images:get",
                "ims:images:list",
                "ims:images:update",
                "ims:images:upload"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "vpc:*:*",
                "ecs:*:get*",
                "ecs:*:list*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:cmk:*",
                "kms:dek:*",
                "kms:grant:*",
                "kms:cmkTag:*",
                "kms:partition:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. 自定义策略“Policy2”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
                "obs:object:ListMultipartUploadParts",
                "obs:object:AbortMultipartUpload",
                "obs:object:GetObjectAcl",
                "obs:object:GetObjectVersionAcl"
            ],
            "Effect": "Allow"
        }
    ]
}
```

> **说明**
>
> 创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“Policy1”为项目级云服务、“Policy2”为全局级云服务。**了解更多**。

4. 将自定义策略授权给开发者用户组user_group。
   a. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1”、“Policy2”、“SWR Admin”。单击“下一步”。

      > **说明**
      >
      > SWR的权限有SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess。但SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess仅限容器镜像服务企业版使用，目前企业版已暂停公测。非企业版用户暂不支持使用此权限。因此需要在此勾选“SWR Admin” 策略。

   b. 选择授权范围方案为“所有资源”，单击“确定”。

精细化授权管理

如果您需要进行精细的权限管理，可参考《ModelArts API参考》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**
- **工作空间管理权限**

精细化授权案例可参考**管理员和开发者权限分离**。

16.6.4.1.2 配置 ModelArts 委托权限

给用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

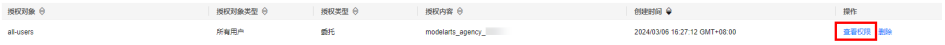
1. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。
2. 在弹出的“添加授权”窗口中，选择：
   - **授权对象类型**：**所有用户**
   - **委托选择**：**新增委托**
   - **权限配置**：**普通用户**

   选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 16-31 配置委托访问授权

3. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

图 16-32 查看委托配置信息



16.6.4.1.3 配置 SWR 组织权限

IAM用户创建后，需要管理员在组织中为用户添加授权，使IAM用户对组织内所有镜像享有读取/编辑/管理的权限。

只有具备“管理”权限的账号和IAM用户才能添加授权。

1. 登录容器镜像服务控制台。
2. 在左侧菜单栏选择“组织管理”，单击组织名称。
3. 在“用户”页签下单击“添加授权”，在弹出的窗口中为IAM用户选择权限，然后单击“确定”。

SWR授权管理详情可参考**授权管理**。

说明

如果给子用户的SWR授权不是SWR Admin权限，则需要继续配置SWR组织权限。

16.6.4.1.4 测试用户权限

由于权限配置需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 验证ModelArts权限。
   a. 在左上角的服务列表中，选择ModelArts服务，进入ModelArts管理控制台。
   b. 在ModelArts管理控制台，可正常创建Notebook、训练作业、注册镜像。
3. 验证SFS权限。
   a. 在左上角的服务列表中，选择SFS服务，进入SFS管理控制台。
   b. 在SFS管理控制台，在SFS Turbo中单击右上角的“创建文件系统”，如果能正常打开页面，表示当前用户具备SFS的操作权限。

4. 验证ECS权限。
   a. 在左上角的服务列表中，选择ECS服务，进入ECS管理控制台。
   b. 在ECS管理控制台，单击右上角的“购买弹性云服务器”，如果能正常打开页面，表示当前用户具备ECS的操作权限。
5. 验证VPC权限。
   a. 在左上角的服务列表中，选择VPC服务，进入VPC管理控制台。
   b. 在VPC管理控制台，单击右上角的“创建虚拟私有云”，如果能正常打开页面，表示当前用户具备VPC的操作权限。
6. 验证DEW权限。
   a. 在左上角的服务列表中，选择DEW服务，进入DEW管理控制台。
   b. 在DEW管理控制台，在“密钥对管理”-“私有密钥对”中单击“创建密钥对”，如果能正常打开页面，表示当前用户具备DEW的操作权限。
7. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
8. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。
   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。
   c. 单击右上角的“上传镜像”，如果能看到授权的组织，表示当前用户具备SWR组织权限。

16.6.4.2 创建网络

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“弹性集群 Cluster”页面。
2. 切换到“网络”页签，单击“创建”，弹出“创建网络”页面。

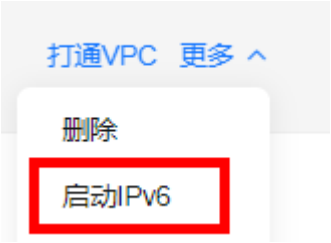
   图 16-33 网络列表

3. 在“创建网络”弹窗中填写网络信息。
   - 网络名称：创建网络时默认生成网络名称，也可自行修改。
   - 网段类型：可选“预置”和“自定义”。自定义网络目前支持网段范围：10.0.0.0/8~26、172.16.0.0/12~26、192.168.0.0/16~26。

- IPV6：开启IPv6功能后，将自动为子网分配IPv6网段，暂不支持自定义设置IPv6网段，该功能一旦开启，将不能关闭。
  - 若创建网络时未勾选开启IPv6，也可在创建网络后在操作列单击“启动IPv6”，如图16-35
  - 打通VPC前，需要保证ModelArts网络和您的VPC网络都已开启IPv6，IPv6才会生效。若是打通VPC后，才开启ModelArts网络的IPv6或VPC网络的IPv6，此时需要重新打通VPC及子网，IPv6才会生效。

图 16-34 创建网络

图 16-35 启动 IPv6



说明

- 单用户最多可创建15个网络。
- 网段设置以后不能修改，避免与将要打通的VPC网段冲突。可能冲突的网段包括：
  - 用户的vpc网段
  - 容器网段（固定是172.16.0.0/16）
  - 服务网段（固定是10.247.0.0/16）

4. 确认无误后，单击“确定”。

16.6.4.3 专属资源池 VPC 打通

通过打通VPC，可以方便用户跨VPC使用资源，提升资源利用率。

步骤一：打通 VPC

通过打通VPC，可以方便用户跨VPC使用资源，提升资源利用率。

1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，在“网络”页签，单击网络列表中某个网络操作列的“打通VPC”。

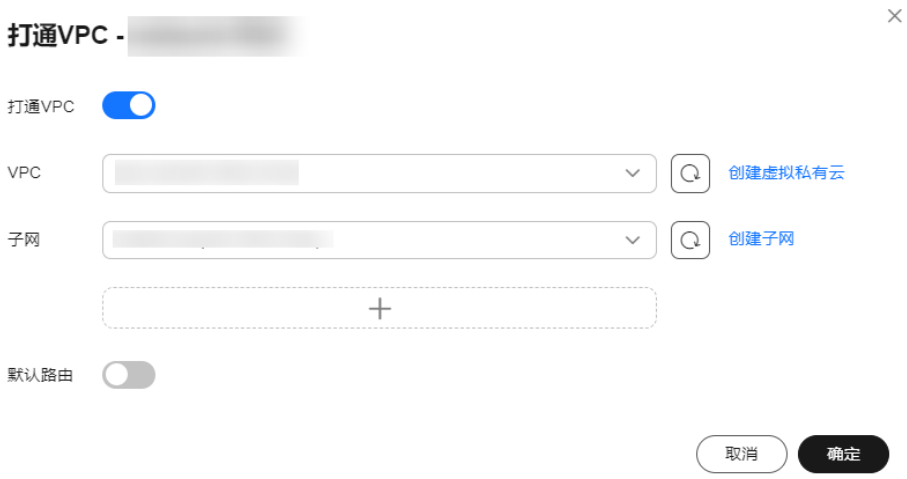
图 16-36 打通 VPC

2. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择可用的VPC和子网。

说明

需要打通的对端网络不能和当前网段重叠。

图 16-37 打通 VPC 参数选择



- 如果没有VPC可选，可以单击右侧的“创建虚拟私有云”，跳转到网络控制台，申请创建虚拟私有云。
- 如果没有子网可选，可以单击右侧的“创建子网”，跳转到网络控制台，创建可用的子网。
- 支持1个VPC下多个子网的打通，如果VPC下有多个子网，会显示“+”，您可单击“+”即可添加子网（上限10个）。
- 如果需要使用打通VPC的方式实现专属资源池访问公网，由于要访问的公网地址不确定，一般是建议用户在VPC中创建SNAT。此场景下，在打通VPC后，专属资源池中作业访问公网地址，默认不能转发到用户VPC的SNAT，需要提交工单联系技术支持在专属资源池VPC的路由中添加指向对等连接的默认路由。当您开启默认路由后，在打通VPC时，会给ModelArts网络0.0.0.0/0路由作为默认路由，此时无需提交工单添加默认路由即可完成网络配置。

16.6.4.4 ECS 服务器挂载 SFS Turbo 存储

本小节介绍如何在ECS服务器挂载SFS Turbo存储，挂载完成后可在后续步骤中，将训练所需的数据通过ECS上传至SFS Turbo。

前提条件

- 已创建SFS Turbo，如果未创建，请参考**创建文件系统**。
- 数据及算法已经上传至OBS，如果未上传，请参考**上传数据和算法至OBS（首次使用时需要）**。
- ECS服务器和SFS的共享硬盘在相同的VPC或者对应VPC能够互联。
- ECS服务器基础镜像需要用Ubuntu 18.04的。
- ECS服务器和SFS Turbo需要在同一子网中。

操作步骤

1. 在ECS服务器中设置华为云镜像源。
```
sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
```
2. 安装NFS客户端，挂载对应盘。
```
sudo apt-get update
sudo apt-get install nfs-common
```
3. 获取SFS Turbo的挂载命令。
   a. 进入弹性文件服务SFS管理控制台。
   b. 选择“SFS Turbo”进入文件系统列表，单击文件系统名称，进入详情页面。
   c. 在“基本信息”页签获取并记录“Linux挂载命令”。
4. 在ECS服务器中挂载NFS存储。

   首先保证对应目录存在，然后输入对应指令即可。命令参考：
```
mkdir -p /mnt/sfs_turbo
mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
```

16.6.4.5 在 ECS 中创建 ma-user 和 ma-group

在ModelArts训练平台使用的自定义镜像时，默认用户为ma-user、默认用户组为ma-group。如果在训练时调用ECS中的文件，需要修改文件权限改为ma-user可读，否则会出现Permission denied错误，因此需要在ECS中提前创建好ma-user和ma-group。

在terminal中执行以下命令：

```
default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
    groupdel -f ${default_group}; \
    groupadd -g 100 ma-group; \
fi && \
if [ -z ${default_group} ]; then \
    groupadd -g 100 ma-group; \
fi && \
if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
    userdel -r ${default_user}; \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
if [ -z ${default_user} ]; then \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
```

```
fi && \
# set bash as default
rm /bin/sh && ln -s /bin/bash /bin/sh
```

查看创建的用户，执行以下命令：

```
id ma-user
```

如果出现以下信息则表示创建成功。

```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

16.6.4.6 obsutil 安装和配置

obsutil是用于访问、管理对象存储服务OBS的命令行工具，使用该工具可以对OBS进行常用的配置管理操作，如创建桶、上传文件/文件夹、下载文件/文件夹、删除文件/文件夹等。

obsutil安装和配置的具体操作指导请参见**obsutils快速入门**。

须知

操作命令中的AK/SK要替换为用户实际获取的AK/SK，Endpoint可以参考**终端节点（Endpoint）和访问域名**获取。

16.6.4.7 （可选）工作空间配置

ModelArts支持设置子用户的细粒度权限、不同工作空间之间资源隔离。ModelArts工作空间帮您实现项目资源隔离、多项目分开结算等功能。

如果您开通了企业项目管理服务的权限，可以在创建工作空间的时候绑定企业项目ID，并在企业项目下添加用户组，为不同的用户组设置细粒度权限供组里的用户使用。

如果您未开通企业项目管理服务的权限，也可以在ModelArts创建自己独立的工作空间，但是无法使用跟企业项目相关的功能。

说明

工作空间为白名单功能，使用该功能需要提工单申请开通。

16.6.5 调试与训练

16.6.5.1 单机单卡

16.6.5.1.1 线下容器镜像构建及调试

镜像构建

1. 导出conda环境

   首先拉起线下的容器镜像：

   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```

在容器中输入如下命令，得到pytorch.tar.gz：

```
# run on container

# 基于想要迁移的base环境创建一个名为pytorch的conda环境
conda create --name pytorch --clone base

pip install conda-pack

#将pytorch env打包生成pytorch.tar.gz
conda pack -n pytorch -o pytorch.tar.gz
```

将打包好的压缩包传到本地：

```
# run on terminal
docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
```

将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时wget获取、解压、清理。

2. 新镜像构建

   基础镜像一般选用ubuntu 18.04的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

   构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

   说明

   - 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
   - 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。

3. 构建参考样例

   Dockerfile样例：

```
FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

USER root

# section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there
alreay exists 1000:100 but not ma-user:ma-group, below code will remove it
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
```



```
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为**1**中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

说明

- 容器镜像的大小建议小于15G，不能大于25G。否则镜像的迁移、拉起都会存在性能问题。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 不建议把数据、代码放到容器镜像里。因为对应内容应该是经常变动的，会导致频繁的容器镜像构建操作。
- 不建议在容器内再创建多个conda env。因为容器已经能满足隔离需求，没有必要再通过conda env做隔离。
- 本教程通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境的依赖。
- 更多ModelArts自定义镜像介绍请见**自定义镜像简介**。

调试要点

1. 确认对应的脚本、代码、流程在linux服务器上运行正常。
   如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

2. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

   训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

3. 测试训练启动脚本。

   a. 优先使用手工进行数据复制的工作并验证

      一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满（请见**ModelArts环境挂载目录说明**）。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

      docker和linux的文件交互命令如下：

      ```
      docker cp data/ 39c9ceedb1f6:/cache/
      ```

      数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

      ```
      cd /cache/code/
      python start_train.py
      ```

      如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

   b. 预置脚本测试整体流程

      一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**run.sh脚本测试ModelArts训练整体流程**。

      如果预置脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

   c. 针对专属池场景

      由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

      可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

      调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

      ```
      docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
      ```

      上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

4. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

   可以通过命令查看容器输出到stdout的所有日志：

   ```
   docker logs -f 39c9ceedb1f6
   ```

   一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

5. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

   ```
   docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
   ```

6. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

16.6.5.1.2 上传镜像

操作场景

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

约束与限制

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。
- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

操作步骤

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。

      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。

   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

3. 上传镜像至镜像仓库。

   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

常见问题

为什么使用客户端上传镜像失败?

16.6.5.1.3 上传数据和算法至 OBS（首次使用时需要）

前提条件

- 已经在OBS上创建好并行文件系统，请参见**创建并行文件系统**。
- 已经在obsutil安装和配置，请参见**obsutils安装和配置**。

准备数据

1. 单击下载**动物数据集**至本地，并解压。
2. 通过obsutil将数据集上传至OBS桶中。

   ```
   ./obsutil cp ./dog_cat_1w obs://${your_obs_buck}/demo/ -f -r
   ```

   说明

   OBS支持多种文件上传方式，当文件少于100个时，可以在OBS Console中上传，当文件大于100个时，推荐使用工具，推荐**OBS Browser+**（win）、**obsutil**（linux）。上述例子为obsutil使用方法。

准备算法

main.py文件内容如下，并将其上传至OBS桶的demo文件夹中：

```
import argparse
import os
import random
import shutil
import time
import warnings
from enum import Enum
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.distributed as dist
import torch.optim
from torch.optim.lr_scheduler import StepLR
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))
parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
```



```
parser.add_argument('data', metavar='DIR', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                        'batch size of all GPUs on the current node when '
                        'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int,
                    help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--dist-backend', default='nccl', type=str,
                    help='distributed backend')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing-distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                        'N processes per node, which has N GPUs. This is the '
                        'fastest way to use PyTorch for either single node or '
                        'multi node data parallel training')
best_acc1 = 0


def main():
    args = parser.parse_args()
    if args.seed is not None:
        random.seed(args.seed)
        torch.manual_seed(args.seed)
        cudnn.deterministic = True
        warnings.warn('You have chosen to seed training. '
                      'This will turn on the CUDNN deterministic setting, '
                      'which can slow down your training considerably! '
                      'You may see unexpected behavior when restarting '
                      'from checkpoints.')
    if args.gpu is not None:
        warnings.warn('You have chosen a specific GPU. This will completely '
                      'disable data parallelism.')
    if args.dist_url == "env://" and args.world_size == -1:
        args.world_size = int(os.environ["WORLD_SIZE"])
```



```
    args.distributed = args.world_size > 1 or args.multiprocessing_distributed
    ngpus_per_node = torch.cuda.device_count()
    if args.multiprocessing_distributed:
        # Since we have ngpus_per_node processes per node, the total world_size
        # needs to be adjusted accordingly
        args.world_size = ngpus_per_node * args.world_size
        # Use torch.multiprocessing.spawn to launch distributed processes: the
        # main_worker process function
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
    else:
        # Simply call main_worker function
        main_worker(args.gpu, ngpus_per_node, args)
def main_worker(gpu, ngpus_per_node, args):
    global best_acc1
    args.gpu = gpu
    if args.gpu is not None:
        print("Use GPU: {} for training".format(args.gpu))
    if args.distributed:
        if args.dist_url == "env://" and args.rank == -1:
            args.rank = int(os.environ["RANK"])
        if args.multiprocessing_distributed:
            # For multiprocessing distributed training, rank needs to be the
            # global rank among all the processes
            args.rank = args.rank * ngpus_per_node + gpu
        dist.init_process_group(backend=args.dist_backend, init_method=args.dist_url,
                                world_size=args.world_size, rank=args.rank)
    # create model
    if args.pretrained:
        print("=> using pre-trained model '{}'".format(args.arch))
        model = models.__dict__[args.arch](pretrained=True)
    else:
        print("=> creating model '{}'".format(args.arch))
        model = models.__dict__[args.arch]()
    if not torch.cuda.is_available():
        print('using CPU, this will be slow')
    elif args.distributed:
        # For multiprocessing distributed, DistributedDataParallel constructor
        # should always set the single device scope, otherwise,
        # DistributedDataParallel will use all available devices.
        if args.gpu is not None:
            torch.cuda.set_device(args.gpu)
            model.cuda(args.gpu)
            # When using a single GPU per process and per
            # DistributedDataParallel, we need to divide the batch size
            # ourselves based on the total number of GPUs of the current node.
            args.batch_size = int(args.batch_size / ngpus_per_node)
            args.workers = int((args.workers + ngpus_per_node - 1) / ngpus_per_node)
            model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
        else:
            model.cuda()
            # DistributedDataParallel will divide and allocate batch_size to all
            # available GPUs if device_ids are not set
            model = torch.nn.parallel.DistributedDataParallel(model)
    elif args.gpu is not None:
        torch.cuda.set_device(args.gpu)
        model = model.cuda(args.gpu)
    else:
        # DataParallel will divide and allocate batch_size to all available GPUs
        if args.arch.startswith('alexnet') or args.arch.startswith('vgg'):
            model.features = torch.nn.DataParallel(model.features)
            model.cuda()
        else:
            model = torch.nn.DataParallel(model).cuda()
    # define loss function (criterion), optimizer, and learning rate scheduler
    criterion = nn.CrossEntropyLoss().cuda(args.gpu)
    optimizer = torch.optim.SGD(model.parameters(), args.lr,
                                momentum=args.momentum,
                                weight_decay=args.weight_decay)
    """Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
```



```
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
# optionally resume from a checkpoint
if args.resume:
    if os.path.isfile(args.resume):
        print("=> loading checkpoint '{}'".format(args.resume))
        if args.gpu is None:
            checkpoint = torch.load(args.resume)
        else:
            # Map model to be loaded to specified single gpu.
            loc = 'cuda:{}'.format(args.gpu)
            checkpoint = torch.load(args.resume, map_location=loc)
        args.start_epoch = checkpoint['epoch']
        best_acc1 = checkpoint['best_acc1']
        if args.gpu is not None:
            # best_acc1 may be from a checkpoint from a different GPU
            best_acc1 = best_acc1.to(args.gpu)
        model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer'])
        scheduler.load_state_dict(checkpoint['scheduler'])
        print("=> loaded checkpoint '{}' (epoch {})"
              .format(args.resume, checkpoint['epoch']))
    else:
        print("=> no checkpoint found at '{}'".format(args.resume))
cudnn.benchmark = True

# Data loading code
traindir = os.path.join(args.data, 'train')
valdir = os.path.join(args.data, 'val')
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize,
    ]))
if args.distributed:
    train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
else:
    train_sampler = None

train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, shuffle=(train_sampler is None),
    num_workers=args.workers, pin_memory=True, sampler=train_sampler)
val_loader = torch.utils.data.DataLoader(
    datasets.ImageFolder(valdir, transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize,
    ])),
    batch_size=args.batch_size, shuffle=False,
    num_workers=args.workers, pin_memory=True)
if args.evaluate:
    validate(val_loader, model, criterion, args)
    return

for epoch in range(args.start_epoch, args.epochs):
    if args.distributed:
        train_sampler.set_epoch(epoch)
    # train for one epoch
    train(train_loader, model, criterion, optimizer, epoch, args)
    # evaluate on validation set
    acc1 = validate(val_loader, model, criterion, args)
    scheduler.step()
    # remember best acc@1 and save checkpoint
    is_best = acc1 > best_acc1
```



```
        best_acc1 = max(acc1, best_acc1)
        if not args.multiprocessing_distributed or (args.multiprocessing_distributed
                                                    and args.rank % ngpus_per_node == 0):
            save_checkpoint({
                'epoch': epoch + 1,
                'arch': args.arch,
                'state_dict': model.state_dict(),
                'best_acc1': best_acc1,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict()
            }, is_best)
def train(train_loader, model, criterion, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses, top1, top5],
        prefix="Epoch: [{}]".format(epoch))
    # switch to train mode
    model.train()
    end = time.time()
    for i, (images, target) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        if torch.cuda.is_available():
            target = target.cuda(args.gpu, non_blocking=True)
        # compute output
        output = model(images)
        loss = criterion(output, target)
        # measure accuracy and record loss
        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        top5.update(acc5[0], images.size(0))
        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if i % args.print_freq == 0:
            progress.display(i)
def validate(val_loader, model, criterion, args):
    batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
    losses = AverageMeter('Loss', ':.4e', Summary.NONE)
    top1 = AverageMeter('Acc@1', ':6.2f', Summary.AVERAGE)
    top5 = AverageMeter('Acc@5', ':6.2f', Summary.AVERAGE)
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, losses, top1, top5],
        prefix='Test: ')
    # switch to evaluate mode
    model.eval()
    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            if args.gpu is not None:
                images = images.cuda(args.gpu, non_blocking=True)
            if torch.cuda.is_available():
                target = target.cuda(args.gpu, non_blocking=True)
            # compute output
            output = model(images)
            loss = criterion(output, target)
```



```
            # measure accuracy and record loss
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            losses.update(loss.item(), images.size(0))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))
            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()
            if i % args.print_freq == 0:
                progress.display(i)
        progress.display_summary()
    return top1.avg
def save_checkpoint(state, is_best, filename='checkpoint.pth.tar'):
    torch.save(state, filename)
    if is_best:
        shutil.copyfile(filename, 'model_best.pth.tar')
class Summary(Enum):
    NONE = 0
    AVERAGE = 1
    SUM = 2
    COUNT = 3


class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self, name, fmt=':f', summary_type=Summary.AVERAGE):
        self.name = name
        self.fmt = fmt
        self.summary_type = summary_type
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)
    def summary(self):
        fmtstr = ''
        if self.summary_type is Summary.NONE:
            fmtstr = ''
        elif self.summary_type is Summary.AVERAGE:
            fmtstr = '{name} {avg:.3f}'
        elif self.summary_type is Summary.SUM:
            fmtstr = '{name} {sum:.3f}'
        elif self.summary_type is Summary.COUNT:
            fmtstr = '{name} {count:.3f}'
        else:
            raise ValueError('invalid summary type %r' % self.summary_type)
        return fmtstr.format(**self.__dict__)
class ProgressMeter(object):
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix
    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))
    def display_summary(self):
```

```
        entries = [" *"]
        entries += [meter.summary() for meter in self.meters]
        print(' '.join(entries))
    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """Computes the accuracy over the k top predictions for the specified values of k"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))
        res = []
        for k in topk:
            correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res
if __name__ == '__main__':
    main()
```

16.6.5.1.4 使用 Notebook 进行代码调试

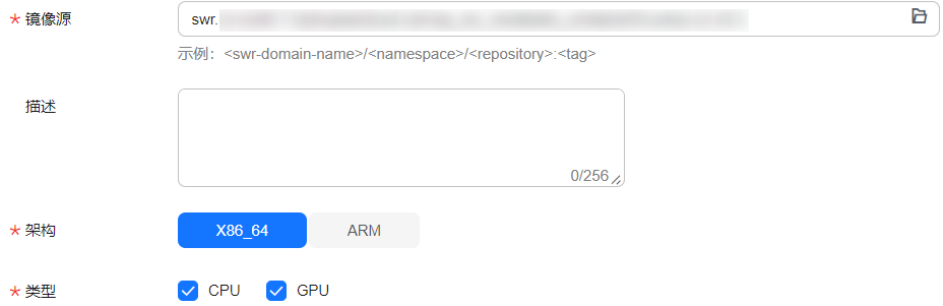
背景信息

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产品不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

创建 Notebook 实例

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如**图16-38**所示。

   图 16-38 注册镜像



2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见表16-8。

      表 16-8 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

   b. 填写Notebook详细参数，如镜像、资源规格等。
      - 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
      - 资源类型：按实际情况选择已创建的专属资源池。
      - 规格：选择1 GPU规格。
      - 存储配置：选择“云硬盘EVS”作为存储位置。

      说明

      如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考VS Code连接Notebook方式介绍。
4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。

   如果创建Notebook启动失败，建议参考调试要点进行检查。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 挂载OBS并行文件系统：在Notebook实例详情页面，选择“存储配置”页签，单击“添加数据存储”，设置挂载参数。
   a. 设置本地挂载目录，在“/data/”目录下输入一个文件夹名称，例如：demo。挂载时，后台自动会在Notebook容器“的/data/”目录下创建该文件夹，用来挂载OBS文件系统。
   b. 选择存放OBS并行文件系统下的文件夹，单击“确定”。
8. 挂载成功后，可以在Notebook实例详情页查看到挂载结果。

9. 代码调试。

   打开Notebook，打开Terminal，进入步骤7中挂载的目录。

   ```
   cd /data/demo
   ```

   执行训练命令：

   ```
   /home/ma-user/anaconda3/envs/pytorch/bin/python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
   ```

   告警"RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.12) doesn't match a supported version!"不影响训练，可忽略。

   说明

   Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

16.6.5.1.5 创建训练任务

说明

针对专属池场景，应注意挂载的目录设置和调试时一致。

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd ${MA_JOB_DIR}/demo && python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
     ```
     此处的“demo”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择单GPU规格。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

16.6.5.1.6 监控资源

用户可以通过资源占用情况窗口查看计算节点的资源使用情况，最多可显示最近三天的数据。在资源占用情况窗口打开时，会定期向后台获取最新的资源使用率数据并刷新。

操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

操作二：单击图例“cpuUsage”、“gpuMemUsage”、“gpuUtil”、“memUsage”“npuMemUsage”、“npuUtil”、可以添加或取消对应参数的使用情况图。

操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

表 16-9 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况。
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况。

16.6.5.2 单机多卡

16.6.5.2.1 准备镜像

构建容器镜像并调试

镜像构建及调试与单机单卡相同。

具体操作，请参考**线下容器镜像构建及调试**。

上传镜像

请参考单机单卡训练的**上传镜像**章节操作。

16.6.5.2.2 上传数据和算法至 SFS（首次使用时需要）

前提条件

- ECS服务器已挂载SFS，请参考**ECS服务器挂载SFS Turbo存储**。
- 在ECS中已经创建ma-user和ma-group用户，请参考**在ECS中创建ma-user和ma-group**。
- 已经安装obsutil，请参考**下载和安装obsutil**。
- 参考**线下容器镜像构建及调试**章节，构建容器镜像并调试，镜像构建及调试与单机单卡相同。
- 上传镜像，参考单机单卡训练的**上传镜像**章节操作。

准备数据

1. 登录coco数据集下载官网地址：**https://cocodataset.org/#download**
2. 下载coco2017数据集的Train（18GB）、Val images（1GB）、Train/Val annotations（241MB），分别解压后并放入coco文件夹中。
3. 下载完成后，将数据上传至SFS相应目录中。由于数据集过大，推荐先通过obsutil工具将数据集传到OBS桶后，再将数据集迁移至SFS。

   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

   ```
   # 将本地数据传至OBS中
   # ./obsutil cp ${数据集所在的本地文件夹路径} ${存放数据集的obs文件夹路径} -f -r
   # 例如
   ./obsutil cp ./coco obs://your_bucket/ -f -r
   ```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

   ```
   # 将OBS数据传至SFS中
   # ./obsutil cp ${数据集所在的obs文件夹路径} ${SFS文件夹路径} -f -r
   # 例如
   ./obsutil cp obs://your_bucket/coco/ /mnt/sfs_turbo/ -f -r
   ```

   /mnt/sfs_turbo/coco文件夹内目录结构如下：

   ```
   coco
   |---annotations
   |---train2017
   |---val2017
   ```

   更多obsutil的操作，可参考**obsutil简介**。

   c. 将文件设置归属为ma-user：

   ```
   chown -R ma-user:ma-group coco
   ```

代码云上适配

1. 下载YOLOX代码。代码仓地址：**https://github.com/Megvii-BaseDetection/YOLOX.git**。

   ```
   git clone https://github.com/Megvii-BaseDetection/YOLOX.git
   cd YOLOX
   git checkout 4f8f1d79c8b8e530495b5f183280bab99869e845
   ```

2. 修改“requirements.txt”中的onnx版本，改为“onnx>=1.12.0”。
3. 将“yolox/data/datasets/coco.py”第59行的“data_dir = os.path.join(get_yolox_datadir(), "COCO")”改为“data_dir = '/home/ma-user/coco'”。

   ```
   # data_dir = os.path.join(get_yolox_datadir(), "COCO")
   data_dir = '/home/ma-user/coco'
   ```

4. 在“tools/train.py”的第13行前加两句代码。

   ```
   # 加上这两句代码，防止运行时找不到yolox module
   import sys
   sys.path.append(os.getcwd())

   # line13
   from yolox.core import launch
   from yolox.exp import Exp, get_exp
   ```

5. 将“yolox/layers/jit_ops.py”第122行的“fast_cocoeval”改为“fast_coco_eval_api”。

   ```
   # def __init__(self, name="fast_cocoeval"):
   def __init__(self, name="fast_coco_eval_api"):
   ```

6. 将“yolox\evaluators\coco_evaluator.py”第294行的“from yolox.layers import COCOeval_opt as COCOeval”改为“from pycocotools.cocoeval import COCOeval”。

   ```
   try:
     # from yolox.layers import COCOeval_opt as COCOeval
   ```

```
  from pycocotools.cocoeval import COCOeval
except ImportError:
  from pycocotools.cocoeval import COCOeval

  logger.warning("Use standard COCOeval.")
```

7. 在tools目录下新建一个“run.sh”作为启动脚本，“run.sh”内容可参考：

```
#!/usr/bin/env sh
set -x
set -o pipefail

export NCCL_DEBUG=INFO

DEFAULT_ONE_GPU_BATCH_SIZE=32
BATCH_SIZE=$((${MA_NUM_GPUS:-8} * ${VC_WORKER_NUM:-1} * $
{DEFAULT_ONE_GPU_BATCH_SIZE}))
if [ ${VC_WORKER_HOSTS} ];then
    YOLOX_DIST_URL=tcp://$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1):6666
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0} \
                            --dist-url ${YOLOX_DIST_URL}
else
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
                            --fp16 \
                            --occupy \
                            --num_machines ${VC_WORKER_NUM:-1} \
                            --machine_rank ${VC_TASK_INDEX:-0}
fi
```

说明

部分环境变量在Notebook环境中不存在，因此需要提供默认值。

8. 将代码放到OBS上，然后通过OBS将代码传至SFS相应目录中。

   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

   ```
   # 将本地代码传至OBS中
   ./obsutil cp ./YOLOX obs://your_bucket/ -f -r
   ```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

   ```
   # 将OBS的代码传到SFS中
   ./obsutil cp obs://your_bucket/YOLOX/ /mnt/sfs_turbo/code/ -f -r
   ```

说明

本案例中以obsutils方式上传文件，除此之外也可通过SCP方式上传文件，具体操作步骤可参考**本地Linux主机使用SCP上传文件到Linux云服务器**。

9. 在SFS中将文件设置归属为ma-user。

```
chown -R ma-user:ma-group YOLOX
```

10. 执行以下命令，去除Shell脚本的\r字符。

```
cd YOLOX
sed -i 's/\r//' run.sh
```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

16.6.5.2.3 使用 Notebook 进行代码调试

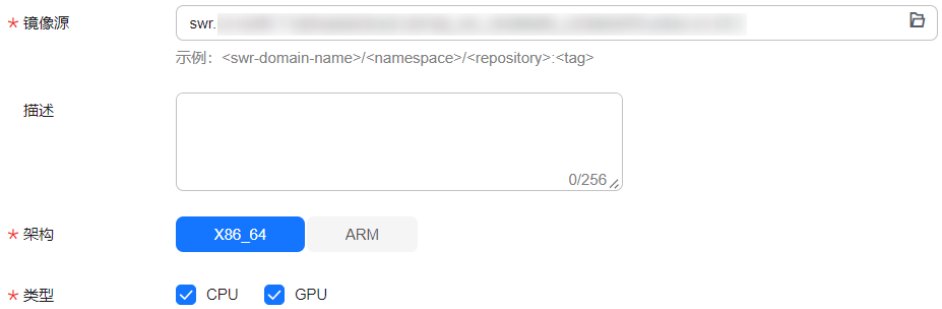
背景信息

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产品不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个账户最多创建10个Notebook。

创建 Notebook 实例

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如**图16-39**所示。

   图 16-39 注册镜像

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表16-10**。

      表 16-10 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。

参数名称	说明
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择8卡GPU规格，“run.sh”文件中默认MA_NUM_GPUS为8卡，因此选择notebook规格时需要与MA_NUM_GPUS默认值相同。
- 存储配置：选择“弹性文件服务SFS”作为存储位置。子目录挂载可不填写，如果需挂载SFS指定目录，则在子目录挂载处填写具体路径。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接N**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 在Notebook中打开Terminal，输入启动命令调试代码。

```
# 建立数据集软链接
# ln -s /home/ma-user/work/${coco数据集在SFS上的路径} /home/ma-user/coco
# 进入到对应目录
# cd /home/ma-user/work/${YOLOX在SFS上的路径}
# 安装环境并执行脚本
# /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh

# 例如
ln -s /home/ma-user/work/coco /home/ma-user/coco
cd /home/ma-user/work/code/YOLOX/
/home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
```

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

16.6.5.2.4 创建训练任务

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。

3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     ln -s /home/ma-user/work/coco /home/ma-user/coco && cd /home/ma-user/work/code/YOLOX/ && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择8卡GPU规格。
   - 计算节点：1。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，因此云上挂载路径需要填写为“/home/ma-user/work”。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

16.6.5.3 多机多卡

16.6.5.3.1 线下容器镜像构建及调试

构建容器镜像并调试

镜像构建及调试与单机单卡相同。

具体操作，请参考**线下容器镜像构建及调试**。

上传镜像

请参考单机单卡训练的**上传镜像**章节操作。

16.6.5.3.2 上传数据至 OBS（首次使用时需要）

前提条件

- 已经在OBS上创建好普通OBS桶，请参见**创建普通OBS桶**。
- 已经安装obsutil，请参考**下载和安装obsutil**。
- 参考**线下容器镜像构建及调试**章节，构建容器镜像并调试，镜像构建及调试与单机单卡相同。
- 上传镜像，参考单机单卡训练的**上传镜像**章节操作。

操作步骤

1. 登录Imagenet数据集下载官网地址，下载Imagenet21k数据集：**http://image-net.org/**
2. 下载格式转换后的annotation文件：**ILSVRC2021winner21k_whole_map_train.txt**和**ILSVRC2021winner21k_whole_map_val.txt**。
3. 下载完成后将上述3个文件数据上传至OBS桶中的imagenet21k_whole文件夹中。上传方法请参考**上传数据和算法至OBS（首次使用时需要）**。

16.6.5.3.3 上传算法至 SFS

1. 下载Swin-Transformer代码。
   ```
   git clone --recursive https://github.com/microsoft/Swin-Transformer.git
   ```
2. 修改lr_scheduler.py文件，把第27行：t_mul=1. 注释掉。
3. 修改data文件夹下imagenet22k_dataset.py，把第28行：print("ERROR IMG LOADED: ", path) 注释掉。
4. 修改data文件夹下的build.py文件，把第112行：prefix = 'ILSVRC2011fall_whole'，改为prefix = 'ILSVRC2021winner21k_whole'。
5. 在Swin-Transformer目录下创建requirements.txt指定python依赖库：
   ```
   # requirements.txt内容如下

   timm==0.4.12
   termcolor==1.1.0
   yacs==0.1.8
   ```
6. 准备run.sh文件中所需要的obs文件路径。
   a. 准备imagenet数据集的分享链接

      勾选要分享的imagenet21k_whole数据集文件夹，单击分享按钮，选择分享链接有效期，自定义提取码，例如123456，单击“复制链接”，记录该链接。

   b. 准备obsutil_linux_amd64.tar.gz的分享链接

      单击**此处**下载obsutil_linux_amd64.tar.gz，将其上传至OBS桶中，设置为公共读。单击属性，单击复制链接。

      链接样例如下：

      ```
      https://${bucketname_name}.obs.cn-north-4.myhuaweicloud.com/${folders_name}/pytorch.tar.gz
      ```
7. 在Swin-Transformer目录下，创建运行脚本run.sh。

   说明

   - 脚本中的"SRC_DATA_PATH=${imagenet数据集在obs中分享链接}"，需要替换为上一步中的imagenet21k_whole文件夹分享链接。
   - 脚本中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz"，需要替换为上一步中obsutil_linux_amd64.tar.gz在OBS上的路径（需将文件设置为公共读）。

   单机单卡运行脚本：

   ```
   # 在代码主目录下创建一个run.sh，内容如下

   #!/bin/bash

   # 从obs中下载数据到本地SSD盘
   DIS_DATA_PATH=/cache
   SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
   OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/
   ```

```
obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf
obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp
$SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -

IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
MASTER_PORT="6061"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nproc_per_node=1
--master_addr localhost --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --
cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml --local_rank 0
```

多机多卡运行脚本：

```
# 创建run.sh

#!/bin/bash

# 从obs中下载数据到本地SSD盘
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/
obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf
obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp
$SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -
IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
MASTER_ADDR=$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1)

MASTER_PORT="6060"
NNODES="$VC_WORKER_NUM"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nnodes=$NNODES
--node_rank=$NODE_RANK --nproc_per_node=$NGPUS_PER_NODE --master_addr $MASTER_ADDR --
master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --cfg ./configs/swin/
swin_base_patch4_window7_224_22k.yaml
```

说明

- 推荐先使用单机单卡运行脚本，待正常运行后再改用多机多卡运行脚本。
- 多机多卡run.sh中的“VC_WORKER_HOSTS”、“VC_WORKER_NUM”、“VC_TASK_INDEX”、“MA_NUM_GPUS”为ModelArts训练容器中预置的环境变量。训练容器环境变量详细介绍可参考**查看训练容器环境变量**。

8. 通过obsutils，将代码文件夹放到OBS上，然后通过OBS将代码传至SFS相应目录中。
9. 在SFS中将代码文件Swin-Transformer-main设置归属为ma-user。
   ```
   chown -R ma-user:ma-group Swin-Transformer
   ```
10. 执行以下命令，去除Shell脚本的\r字符。
    ```
    cd Swin-Transformer
    sed -i 's/\r//' run.sh
    ```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

16.6.5.3.4 创建训练任务

调试代码

创建训练任务之前，建议先调试代码。

由于Notebook的/cache目录只能支持500G的存储，超过后会导致实例重启，ImageNet数据集大小超过该限制，因此建议用线下资源调试、或用小批量数据集在Notebook调试（Notebook调试方法与**使用Notebook进行代码调试**、**使用Notebook进行代码调试**相同）。

创建训练任务

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd /home/ma-user/work/code/Swin-Transformer && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择所需GPU规格。
   - 计算节点个数：选择需要的节点个数。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，云上挂载路径需要填写为“/home/ma-user/work”。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

16.6.6 FAQ

16.6.6.1 CUDA 和 CUDNN

Vnt1 机型软件版本建议：gpu driver version : 440.95.01

- gpu driver version : 440.95.01（GPU驱动在宿主机中安装，镜像中无需安装）
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6.x（PyTorch自带，无需关心）
- pytorch version : 1.x.x+cu102

Vnt1 机型软件版本建议：gpu driver version : 470.57.02

- gpu driver version : 470.57.02（GPU驱动在宿主机中安装，镜像中无需安装）
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6（PyTorch自带，无需关心）
- pytorch version : 1.X.X-cu102

Vnt1 机型软件版本建议：gpu driver version : 510.65.01

- gpu driver version :510.65.01
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6（PyTorch自带，无需关心）
- pytorch version : 1.X.X-cu102

CUDA Compatibility 如何使用?

当CUDA 10.2与低版本GPU驱动（440.33以下）配合使用时，可能会出现兼容问题，此时需要使用CUDA Compatibility。在创建训练页面添加以下环境变量：

```
export LD_LIBRARY_PATH=/usr/local/cuda/compat
```

说明

训练时默认不需要加此环境变量，仅当发现驱动版本不够时才使用此方法。

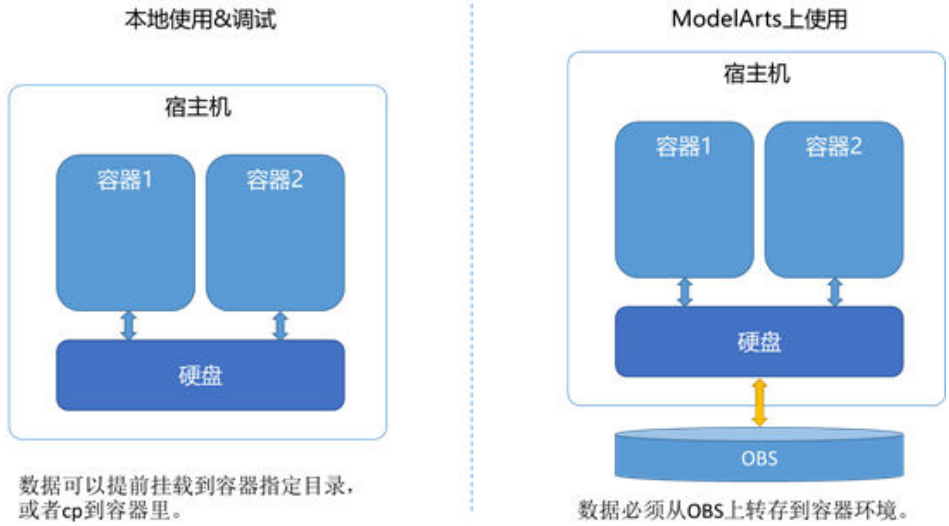
专属池驱动版本如何升级?

当专属资源池中的节点含有GPU/Ascend资源时，用户基于自己的业务，可能会有自定义GPU/Ascend驱动的需求，ModelArts面向此类客户提供了自助升级专属资源池GPU/Ascend驱动的能力，具体操作请参见资源池驱动升级。

16.6.6.2 run.sh 脚本测试 ModelArts 训练整体流程

自定义容器在ModelArts上训练和本地训练的区别如下图：

图 16-40 本地与 ModelArts 上训练对比



ModelArts上进行训练比本地训练多了一步OBS和容器环境的数据迁移工作。

增加了和OBS交互工作的整个训练流程如下：

说明

建议使用OBSutil作为和OBS交互的工具，如何在本机安装obsutil可以参考**obsutil安装和配置**。

1. 训练数据、代码、模型下载。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）
2. 启动脚本，用法无切换，一般就是到达执行目录，然后python xxx.py。
3. 训练结果、日志、checkpoints上传。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）

可以用一个run脚本把整个流程包起来。run.sh脚本的内容可以参考如下示例：

```
#!/bin/bash

##认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
##本示例以AK和SK保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

##安装obsutil，完成AKSK配置。建议在基础镜像里做好。
#mkdir -p /opt && cd /opt
#wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/obsutil_linux_amd64.tar.gz
#tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils
#alias obsutil='/opt/utils/obsutil'
#obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-north-4.myhuaweicloud.com

##训练输入复制到容器镜像本地。
#/cache目录的容量较大。

DATA_URL=`echo ${DLS_DATA_URL} | sed /s/s3/obs/`
mkdir –p /cache/data
/opt/utils/obsutil cp –r –f ${DATA_URL} /cache/data

##执行训练任务。
#涉及conda env切换时。
source /xxxxx/etc/profile.d/conda.sh
conda activate xxxenv
conda info --envs
#启动训练脚本。
cd xxxx
python xxx.py

##复制输出结果到OBS目录。
TRAIN_URL=`echo ${DLS_TRAIN_URL} | sed /s/s3/obs/`
/opt/utils/obsutil cp –r –f /cache/out ${TRAIN_URL}
```

把run.sh放到/opt目录，在实际启动任务的时候，使用以下命令启动任务即可：

```
bash –x /opt/run.sh
```

把run.sh放到/root目录，可以在原镜像里增加一层，这一层就只是COPY这个run脚本。在基础镜像里可以一起把obsutil安装、配置好。参考如下dockerfile：

```
FROM $your_docker_image_tag

RUN mkdir -p /opt && cd /opt && \
        wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/obsutil_linux_amd64.tar.gz && \
        tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils && \
        /opt/utils/obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-north-4.myhuaweicloud.com
```

```
COPY run.sh /opt/run.sh
```

须知

ModelArts的容器会有一个/cache目录，这个目录挂载的硬盘容量最大。建议下载数据和中间数据都存到这个目录中，防止因硬盘占满导致任务失败。

16.6.6.3 ModelArts 环境挂载目录说明

本小节介绍Notebook开发环境、训练任务实例的目录挂载情况（以下挂载点在保存镜像的时候不会保存）。详情如下：

Notebook

表 16-11 Notebook 挂载点介绍

挂载点	是否只读	备注
/home/ma-user/work/	否	客户数据的持久化目录。
/data	否	客户PFS的挂载目录。
/cache	否	裸机规格时支持，用于挂载宿主机NVMe的硬盘。
/train-worker1-log	否	兼容训练任务调试过程。
/dev/shm	否	用于PyTorch引擎加速。
/modelarts	是	/
/etc/secret-volume	是	/
/etc/sudoers	是	/
/etc/localtime	是	/
var/run/secrets/kubernetes.io/serviceaccount	是	/

训练任务

表 16-12 训练任务挂载点介绍

挂载点	是否只读	备注
/xxx	否	专属池使用SFS盘挂载的目录，路径由客户自己指定。

挂载点	是否只读	备注
/home/ma-user/modelarts	否	空文件夹，建议用户主要用这个目录。
/cache	否	裸机规格支持，挂载宿主机NVMe的硬盘。
/dev/shm	否	用于PyTorch引擎加速。
/usr/local/nvidia	是	宿主机的nvidia库。

16.6.6.4 infiniband 驱动的安装

infiniband 驱动的安装

如果安装了libibverbs-dev库后仍然无法使能infiniband网卡，您可以直接安装infiniband官方驱动，以使用infiniband网卡进行分布式通信，提升训练性能。infiniband驱动需要在制作镜像时安装。

操作步骤

1. 下载MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz。

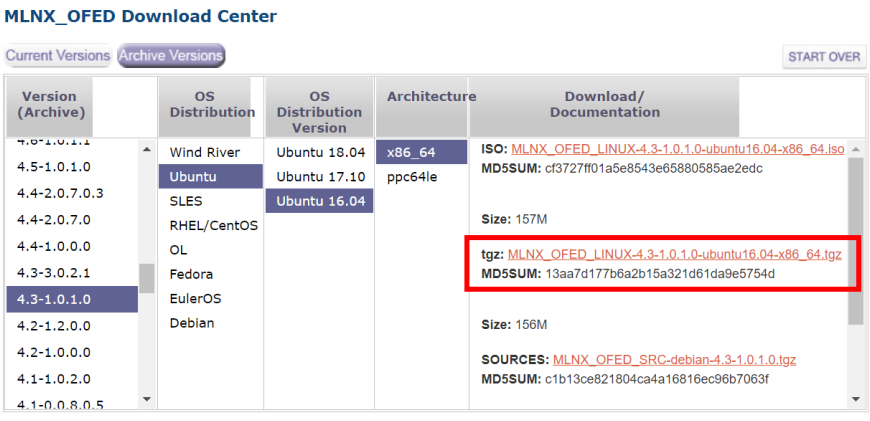
   进入地址，单击“Download”，选择“Archive Versions”，“Version”选择“4.3-1.0.1.0”，“OS Distribution”选择“Ubuntu”，“OS Distribution Version”选择“Ubuntu 16.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz。

   说明

   宿主机安装的infiniband驱动版本为4.3-1.0.1.0，容器镜像中安装的infiniband驱动版本需要与宿主机版本匹配，即同为4.3-1.0.1.0。

   可能部分区域的网卡较新，会出现更高版本的infiniband驱动版本，如果您遇到了infiniband驱动安装后，仍然无法使能infiniband网卡的问题，可以咨询相关运维人员以确认宿主机的实际infiniband驱动版本。

图 16-41 下载驱动



2. 参考如下Dockerfile中，以在容器镜像中安装infiniband驱动。

```
USER root

# copy MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz to docker image

RUN tar xzvf MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz && \
    cd MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64 && \
    chmod +x mlnxofedinstall && \
    ./mlnxofedinstall --user-space-only --without-fw-update --force && \
    cd - && \
    rm MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz && \
    rm -rf MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64

USER ma-user
```

3. 验证infiniband驱动是否安装成功。

   在训练代码中执行以下命令，如果无报错则infiniband驱动安装成功：

```
os.system("ofed_info")
```

16.6.6.5 如何保证训练和调试时文件路径保持一致

云上挂载路径

Notebook中挂载SFS后，SFS默认在“/home/ma-user/work”路径下。在创建训练作业时，设置SFS Turbo的“云上挂载路径”为“/home/ma-user/work”，使得训练环境下SFS也在“/home/ma-user/work”路径下。

ln -s 建立软连接

如果代码中涉及文件绝对路径，由于Notebook调试与训练作业环境不同，可能会导致文件绝对路径不一致，需要修改代码内容。推荐使用软链接的方式解决该问题，用户只需提前建立好软链接，代码中的地址可保持不变。

新建软链接：

```
# ln -s 源目录/文件 目标目录/文件
# 例如
ln -s /mnt/sfs_turbo/data/coco /coco
```

删除软链接：

```
# rm 目标目录/文件
rm /coco
```