ModelArts

最佳实践

文档版本 01

发布日期 2024-08-15

华为云计算技术有限公司

商标声明


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

注意

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

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

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 ModelArts 最佳实践案例列表 1
2 昇腾能力应用地图 7
3 LLM 大语言模型训练推理 12
3.1 主流开源大模型基于 DevServer 适配 ModelLink PyTorch NPU 训练指导（6.3.907） 12
3.1.1 场景介绍 12
3.1.2 准备工作 15
3.1.2.1 准备环境 15
3.1.2.2 准备代码 15
3.1.2.3 准备数据 18
3.1.2.4 准备镜像 19
3.1.3 预训练任务 21
3.1.4 SFT 全参微调训练任务 24
3.1.5 LoRA 微调训练 26
3.1.6 查看日志和性能 28
3.1.7 训练脚本说明 29
3.1.7.1 训练启动脚本说明和参数配置 29
3.1.7.2 训练的数据集预处理说明 35
3.1.7.3 训练中的权重转换说明 39
3.1.7.4 训练 tokenizer 文件说明 41
3.1.8 常见错误原因和解决方法 43
3.1.8.1 显存溢出错误 43
3.1.8.2 网卡名称错误 44
3.1.8.3 保存 ckpt 时超时报错 44
3.2 主流开源大模型基于 DevServer 适配 LlamaFactory PyTorch NPU 训练指导（6.3.907） 45
3.2.1 场景介绍 45
3.2.2 准备工作 46
3.2.2.1 准备环境 46
3.2.2.2 准备代码 47
3.2.2.3 准备镜像环境 48
3.2.2.4 准备数据（可选） 50
3.2.3 指令监督微调训练任务 51
3.2.4 查看日志和性能 54

3.2.5 训练脚本说明 56
3.2.5.1 yaml 配置文件参数配置说明 56
3.2.5.2 各个模型深度学习训练加速框架的选择 60
3.2.5.3 模型 NPU 卡数取值表 60
3.2.5.4 各个模型训练前文件替换 62
3.2.6 附录：指令微调训练常见问题 62
3.3 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907） 63
3.3.1 推理场景介绍 63
3.3.2 部署推理服务 69
3.3.3 推理性能测试 78
3.3.4 推理精度测试 81
3.3.5 推理模型量化 84
3.3.5.1 使用 AWQ 量化 84
3.3.5.2 使用 SmoothQuant 量化 85
3.3.5.3 使用 kv-cache-int8 量化 86
3.3.6 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明 87
3.3.7 附录：大模型推理常见问题 90
3.4 主流开源大模型基于 Standard+OBS 适配 PyTorch NPU 训练指导（6.3.907） 90
3.4.1 场景介绍 90
3.4.2 准备工作 94
3.4.2.1 准备资源 94
3.4.2.2 准备数据 94
3.4.2.3 准备权重 95
3.4.2.4 准备代码 96
3.4.2.5 准备镜像 97
3.4.2.6 准备 Notebook（可选） 101
3.4.3 预训练 102
3.4.4 SFT 全参微调训练 106
3.4.5 LoRA 微调训练 110
3.4.6 查看日志和性能 114
3.4.7 训练脚本说明 115
3.4.7.1 训练启动脚本说明和参数配置 115
3.4.7.2 训练的数据集预处理说明 120
3.4.7.3 训练的权重转换说明 122
3.4.7.4 训练 tokenizer 文件说明 124
3.4.8 常见错误原因和解决方法 125
3.4.8.1 显存溢出错误 125
3.4.8.2 网卡名称错误 126
3.4.8.3 保存 ckpt 时超时报错 126
3.5 主流开源大模型基于 Standard+OBS+SFS 适配 PyTorch NPU 训练指导（6.3.907） 127
3.5.1 场景介绍 127
3.5.2 准备工作 130

3.5.2.1 准备资源 130
3.5.2.2 准备数据 132
3.5.2.3 准备权重 133
3.5.2.4 准备代码 134
3.5.2.5 准备镜像 135
3.5.2.5.1 镜像方案说明 135
3.5.2.5.2 ECS 获取和上传基础镜像 136
3.5.2.5.3 使用基础镜像 138
3.5.2.5.4 ECS 中构建新镜像 138
3.5.2.5.5 Notebook 中构建新镜像 140
3.5.3 预训练 143
3.5.4 SFT 全参微调训练 146
3.5.5 LoRA 微调训练 150
3.5.6 查看日志和性能 153
3.5.7 训练脚本说明 154
3.5.7.1 训练启动脚本说明和参数配置 154
3.5.7.2 训练的数据集预处理说明 160
3.5.7.3 训练的权重转换说明 162
3.5.7.4 训练 tokenizer 文件说明 164
3.5.8 常见错误原因和解决方法 166
3.5.8.1 显存溢出错误 166
3.5.8.2 网卡名称错误 166
3.5.8.3 保存 ckpt 时超时报错 167
3.6 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.907） 167
3.6.1 场景介绍 167
3.6.2 准备工作 172
3.6.2.1 准备资源 173
3.6.2.2 准备权重 174
3.6.2.3 准备代码 174
3.6.2.4 准备镜像 175
3.6.2.5 准备 Notebook 178
3.6.3 在 Notebook 调试环境中部署推理服务 179
3.6.4 在推理生产环境中部署推理服务 188
3.6.5 推理精度测试 194
3.6.6 推理性能测试 197
3.6.7 推理模型量化 201
3.6.7.1 使用 AWQ 量化工具转换权重 201
3.6.7.2 使用 SmoothQuant 量化工具转换权重 202
3.6.7.3 使用 kv-cache-int8 量化 203
3.6.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明 204
3.6.9 附录：大模型推理 standard 常见问题 207
3.7 主流开源大模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.906） 208

3.7.1 场景介绍 208
3.7.2 准备工作 211
3.7.2.1 准备环境 211
3.7.2.2 准备代码 212
3.7.2.3 准备数据 214
3.7.2.4 准备镜像 215
3.7.3 预训练任务 217
3.7.4 SFT 全参微调训练任务 219
3.7.5 LoRA 微调训练 221
3.7.6 查看日志和性能 222
3.7.7 训练脚本说明 223
3.7.7.1 训练启动脚本说明和参数配置 223
3.7.7.2 训练的数据集预处理说明 229
3.7.7.3 训练中的权重转换说明 233
3.7.7.4 训练 tokenizer 文件说明 235
3.8 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906） 237
3.8.1 推理场景介绍 237
3.8.2 部署推理服务 242
3.8.3 推理性能测试 249
3.8.4 推理精度测试 251
3.8.5 推理模型量化 253
3.8.5.1 使用 AWQ 量化 253
3.8.5.2 使用 SmoothQuant 量化 254
3.8.5.3 使用 kv-cache-int8 量化 255
3.8.6 附录：大模型推理常见问题 257
3.9 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.906） 258
3.9.1 场景介绍 258
3.9.2 准备工作 261
3.9.2.1 准备资源 261
3.9.2.2 准备数据 262
3.9.2.3 准备权重 263
3.9.2.4 准备代码 264
3.9.2.5 准备镜像 265
3.9.2.6 准备 Notebook 268
3.9.3 预训练 271
3.9.4 SFT 全参微调训练 273
3.9.5 LoRA 微调训练 275
3.9.6 开启训练故障自动重启功能 277
3.9.7 查看日志和性能 277
3.9.8 训练脚本说明 278
3.9.8.1 训练启动脚本说明和参数配置 278
3.9.8.2 训练的数据集预处理说明 284

3.9.8.3 训练的权重转换说明 286
3.9.8.4 训练 tokenizer 文件说明 288
3.10 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.906） 290
3.10.1 场景介绍 290
3.10.2 准备工作 294
3.10.2.1 准备资源 295
3.10.2.2 准备权重 295
3.10.2.3 准备代码 295
3.10.2.4 准备镜像 296
3.10.2.5 准备 Notebook 300
3.10.3 在 Notebook 调试环境中部署推理服务 301
3.10.4 在推理生产环境中部署推理服务 306
3.10.5 推理精度测试 312
3.10.6 推理性能测试 314
3.10.7 推理模型量化 318
3.10.7.1 使用 AWQ 量化工具转换权重 318
3.10.7.2 使用 SmoothQuant 量化工具转换权重 319
3.10.7.3 使用 kv-cache-int8 量化 320
3.11 主流开源大模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.905） 321
3.11.1 场景介绍 321
3.11.2 准备工作 323
3.11.2.1 准备环境 324
3.11.2.2 准备代码 324
3.11.2.3 准备数据 327
3.11.2.4 准备镜像 327
3.11.3 预训练任务 329
3.11.4 SFT 全参微调训练任务 331
3.11.5 LoRA 微调训练 333
3.11.6 查看日志和性能 334
3.11.7 训练脚本说明 335
3.11.7.1 训练启动脚本说明和参数配置 335
3.11.7.2 训练的数据集预处理说明 340
3.11.7.3 训练中的权重转换说明 342
3.11.7.4 训练 tokenizer 文件说明 344
3.12 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.905） 345
3.12.1 推理场景介绍 345
3.12.2 部署推理服务 349
3.12.3 推理性能测试 356
3.12.4 推理精度测试 359
3.12.5 附录：大模型推理常见问题 361
3.13 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.905） 361
3.13.1 场景介绍 361

3.13.2 准备工作 364
3.13.2.1 准备资源 364
3.13.2.2 准备数据 365
3.13.2.3 准备权重 366
3.13.2.4 准备代码 367
3.13.2.5 准备镜像 368
3.13.2.6 准备 Notebook 370
3.13.3 预训练 373
3.13.4 SFT 全参微调训练 375
3.13.5 LoRA 微调训练 377
3.13.6 查看日志和性能 379
3.13.7 训练脚本说明 380
3.13.7.1 训练启动脚本说明和参数配置 380
3.13.7.2 训练的数据集预处理说明 385
3.13.7.3 训练的权重转换说明 387
3.13.7.4 训练 tokenizer 文件说明 389
3.14 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.905） 390
3.14.1 场景介绍 390
3.14.2 准备工作 393
3.14.2.1 准备资源 393
3.14.2.2 准备权重 393
3.14.2.3 准备代码 394
3.14.2.4 准备镜像 395
3.14.2.5 准备 Notebook 399
3.14.3 在 Notebook 调试环境中部署推理服务 399
3.14.4 在推理生产环境中部署推理服务 405
3.14.5 推理精度测试 410
3.14.6 推理性能测试 412
3.15 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.904） 416
3.15.1 推理场景介绍 416
3.15.2 部署推理服务 420
3.15.3 推理性能测试 427
3.15.4 推理精度测试 429
3.16 LLama2 系列模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904） 431
3.16.1 场景介绍 431
3.16.2 准备工作 433
3.16.2.1 准备环境 433
3.16.2.2 准备代码 433
3.16.2.3 准备数据 436
3.16.2.4 准备镜像 436
3.16.3 预训练 439
3.16.3.1 预训练数据处理 439

3.16.3.2 预训练任务 440
3.16.3.3 断点续训练 444
3.16.3.4 查看日志和性能 445
3.16.4 SFT 全参微调训练 446
3.16.4.1 SFT 全参微调数据处理 447
3.16.4.2 SFT 全参微调权重转换 448
3.16.4.3 SFT 全参微调任务 449
3.16.5 LoRA 微调训练 452
3.16.6 推理前的权重合并转换 455
3.17 Qwen 系列模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904） 457
3.17.1 场景介绍 457
3.17.2 准备工作 458
3.17.2.1 准备环境 458
3.17.2.2 准备代码 459
3.17.2.3 准备数据 461
3.17.2.4 准备镜像 462
3.17.3 预训练 464
3.17.3.1 预训练数据处理 464
3.17.3.2 预训练任务 466
3.17.3.3 断点续训练 469
3.17.3.4 查看日志和性能 471
3.17.4 SFT 微调训练 472
3.17.4.1 SFT 微调数据处理 472
3.17.4.2 SFT 微调权重转换 473
3.17.4.3 SFT 微调训练任务 474
3.17.5 LoRA 微调训练 477
3.17.6 推理前的权重合并转换 480
3.17.7 常见问题 482
3.17.7.1 访问容器目录时提示 Permission denied 482
3.17.7.2 如何在容器中安装依赖包 482
3.17.7.3 训练时报“EI0006: Getting socket times out” 483
3.18 GLM3-6B 模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904） 483
3.18.1 场景介绍 483
3.18.2 准备工作 484
3.18.2.1 准备环境 484
3.18.2.2 准备代码 485
3.18.2.3 准备数据 487
3.18.2.4 准备镜像 488
3.18.3 预训练 490
3.18.3.1 预训练数据处理 490
3.18.3.2 预训练任务 492
3.18.3.3 断点续训练 494

3.18.3.4 查看日志和性能 496
3.18.4 SFT 全参微调训练 497
3.18.4.1 SFT 全参微调数据处理 497
3.18.4.2 SFT 全参微调权重转换 499
3.18.4.3 SFT 全参微调任务 500
3.18.5 LoRA 微调训练 502
3.18.6 推理前的权重合并转换 505
3.19 Baichuan2-13B 模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904） 506
3.19.1 场景介绍 506
3.19.2 准备工作 508
3.19.2.1 准备环境 508
3.19.2.2 准备代码 508
3.19.2.3 准备数据 511
3.19.2.4 准备镜像 512
3.19.3 预训练 514
3.19.3.1 预训练数据处理 514
3.19.3.2 预训练超参配置 515
3.19.3.3 预训练任务 517
3.19.3.4 断点续训练 518
3.19.3.5 查看日志和性能 519
3.19.4 SFT 全参微调 520
3.19.4.1 SFT 全参微调数据处理 520
3.19.4.2 SFT 全参微调权重转换 522
3.19.4.3 SFT 全参微调超参配置 523
3.19.4.4 SFT 全参微调任务 524
3.19.4.5 查看性能 525
3.19.5 LoRA 微调训练 525
3.19.6 推理前的权重合并转换 528
4 AIGC 模型训练推理 530
4.1 SDXL 基于 Standard 适配 PyTorch NPU 的 LoRA 训练指导（6.3.907） 530
4.2 SD1.5&SDXL Diffusers 框架基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907） 537
4.2.1 训练场景和方案介绍 537
4.2.2 准备镜像环境 538
4.2.3 Finetune 训练 540
4.2.4 LoRA 训练 541
4.2.5 Controlnet 训练 542
4.3 SD1.5&SDXL Diffusers 框架基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907） 543
4.4 SD3 Diffusers 框架基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907） 548
4.5 SD WEBUI 套件适配 PyTorch NPU 的推理指导（6.3.907） 551
4.5.1 SD WebUI 推理方案概览 551
4.5.2 在 DevServer 上部署 SD WebUI 推理服务 552
4.5.3 在 Standard 上部署 SD WebUI 推理服务 556

4.5.4 SD WebUI 推理性能测试 565
4.6 SD1.5&SDXL Koyha 框架基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907） 567
4.6.1 训练场景和方案介绍 567
4.6.2 准备镜像环境 568
4.6.3 Finetune 训练 570
4.6.4 LoRA 训练 571
4.7 Open-Sora-Plan1.0 基于 DevServer 适配 PyTorch NPU 训练推理指导（6.3.907） 571
4.8 Open-Sora1.2 基于 DevServer 适配 PyTorch NPU 训练推理指导（6.3.907） 579
4.9 SDXL&SD1.5 ComfyUI 插件基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906） 584
4.10 SDXL&SD1.5 ComfyUI 基于 Lite Cluster 适配 NPU 推理指导（6.3.906） 591
4.11 SDXL&SD1.5 WebUI 基于 Lite Cluster 适配 NPU 推理指导（6.3.906） 593
4.12 LLaVA 模型基于 DevServer 适配 PyTorch NPU 预训练指导（6.3.906） 598
4.13 LLaVA 模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906） 603
4.14 Qwen-VL 基于 DevServer 适配 Pytorch NPU 的 Finetune 训练指导(6.3.906) 608
4.15 Qwen-VL 基于 DevServer 适配 Pytorch NPU 的推理指导(6.3.906) 613
4.16 Open-Sora 1.0 基于 DevServer 适配 PyTorch NPU 训练指导（6.3.905） 619
4.17 SDXL 基于 Standard 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905） 628
4.18 SDXL 基于 DevServer 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905） 633
4.19 SDXL 基于 DevServer 适配 PyTorch NPU 的 LoRA 训练指导（6.3.905） 637
4.20 SDXL ComfyUI 插件基于 DevServer 适配 PyTorch NPU 推理指导（6.3.904） 640
4.21 SD1.5 基于 DevServer 适配 PyTorch NPU Finetune 训练指导（6.3.904） 646
4.22 SDXL Diffusers 框架基于 Devserver 适配 PyTorch NPU 推理指导（6.3.902） 652
4.23 SDXL WebUI 基于 Devserver 适配 PyTorch NPU 推理指导（6.3.902） 657
4.24 Open-Clip 基于 DevServer 适配 PyTorch NPU 训练指导 664
4.25 moondream2 基于 DevServer 适配 PyTorch NPU 推理指导 670
4.26 AIGC 工具 tailor 使用指导 674
5 数字人模型训练推理 680
5.1 Wav2Lip 推理基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907） 680
5.2 Wav2Lip 训练基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907） 684
5.3 Wav2Lip 基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906） 689
5.4 Wav2Lip 基于 DevServer 适配 PyTorch NPU 训练指导（6.3.902） 693
6 GPU 业务迁移至昇腾训练推理 699
6.1 ModelArts 昇腾迁移调优工具总览 699
6.2 基于 LLM 模型的 GPU 训练业务迁移至昇腾指导 702
6.2.1 场景介绍 702
6.2.2 环境准备 702
6.2.3 迁移适配 703
6.2.4 精度对齐 706
6.2.5 性能调优 713
6.2.6 常见问题 715
6.2.6.1 报错提示 RuntimeError: Default process group has not been initialized, please make sure to call init_process_group 715

6.2.6.2 训练运行报错 AttributeError: 'torch_npu._C._NPUDeviceProperties' object has no attribute 'multi_processor_count'..... 716
6.2.6.3 deepspeed 多卡训练报错 TypeError: deepspeed_init() got an unexpected keyword argument 'resume_from_checkpoint'..... 716
6.2.6.4 Huggingface 缓存目录空间不足，出现 OSError: [Errno 122] Disk quota exceeded..... 717
6.2.6.5 调用 transformers 出现 ImportError: Using the `Trainer` with `PyTorch` requires `accelerate`: Run `pip install --upgrade accelerate`..... 717
6.2.6.6 调用 transformers 出现 ImportError: libcblas.so.3: cannot open shared object file: No such file or directory..... 717
6.2.6.7 transformers 调用 cuda 上的操作，或者执行卡死..... 718
6.3 GPU 训练业务迁移至昇腾的通用指导..... 718
6.3.1 训练业务迁移到昇腾设备场景介绍..... 718
6.3.2 训练迁移快速入门案例..... 719
6.3.3 迁移环境准备..... 719
6.3.4 训练代码迁移..... 720
6.3.5 PyTorch 迁移精度调优..... 722
6.3.6 PyTorch 迁移性能调优..... 730
6.3.6.1 性能调优总体原则和思路..... 730
6.3.6.2 自动诊断工具 MA-Advisor 使用指导..... 732
6.3.6.2.1 自动诊断工具 MA-Advisor 简介..... 732
6.3.6.2.2 MA-Advisor 使用指导..... 732
6.3.6.2.3 昇腾迁移融合算子 API 替换样例..... 743
6.3.6.2.4 AI CPU 算子替换样例..... 749
6.3.6.3 性能可视化工具 Ascend-Insight 使用指导..... 754
6.3.6.4 其他性能分析工具..... 754
6.3.7 训练网络迁移总结..... 754
6.4 基于 AIGC 模型的 GPU 推理业务迁移至昇腾指导..... 755
6.4.1 场景介绍..... 755
6.4.2 迁移环境准备..... 755
6.4.3 pipeline 应用准备..... 756
6.4.4 应用迁移..... 759
6.4.4.1 模型适配..... 759
6.4.4.2 pipeline 代码适配..... 764
6.4.5 迁移效果校验..... 769
6.4.6 模型精度调优..... 770
6.4.6.1 场景介绍..... 770
6.4.6.2 精度问题诊断..... 770
6.4.6.3 精度问题处理..... 771
6.4.7 性能调优..... 772
6.4.7.1 单模型性能测试工具 Mindspore lite benchmark..... 772
6.4.7.2 单模型性能调优 AOE..... 772
6.4.8 常见问题..... 774
6.4.8.1 模型转换失败怎么办？..... 774

6.4.8.2 图片大 Shape 性能劣化严重怎么办? 774
6.4.8.3 同样功能的 PyTorch Pipeline，因为指导要求适配 onnx pipeline，两个 pipeline 本身功能就有差别，如何适配? 775
6.4.8.4 AOE 的自动性能调优使用上完全没有效果怎么办? 775
6.4.8.5 迁移后应用出图效果相比 GPU 无法对齐怎么办 775
6.4.8.6 模型精度有问题怎么办? 775
6.4.8.7 模型转换失败时如何查看日志和定位原因? 775
6.4.8.8 Stable Diffusion WebUI 如何适配? 776
6.4.8.9 LoRA 适配流是怎么样的? 776
6.4.8.10 数据类型不匹配问题如何处理? 776
6.5 GPU 推理业务迁移至昇腾的通用指导 777
6.5.1 简介 777
6.5.2 昇腾迁移快速入门案例 779
6.5.3 迁移评估 781
6.5.4 环境准备 782
6.5.5 模型适配 784
6.5.5.1 基于 MindSpore Lite 的模型转换 784
6.5.5.2 动态 shape 786
6.5.6 精度校验 787
6.5.7 性能调优 788
6.5.8 迁移过程使用工具概览 791
6.5.9 常见问题 792
6.5.9.1 MindSpore Lite 问题定位指南 792
6.5.9.2 模型转换报错如何查看日志和定位? 792
6.5.9.3 日志提示 Compile graph failed 793
6.5.9.4 日志提示 Custom op has no reg_op_name attr 793
6.5.10 推理业务迁移评估表 793

7 Standard 权限管理 798

7.1 ModelArts 权限管理基本概念 798
7.2 权限控制方式 803
7.2.1 IAM 803
7.2.2 依赖和委托 811
7.2.3 工作空间 840
7.3 典型场景配置实践 840
7.3.1 个人用户快速配置 ModelArts 访问权限 840
7.3.2 配置 ModelArts 基本使用权限 844
7.3.2.1 场景描述 844
7.3.2.2 Step1 创建用户组并加入用户 845
7.3.2.3 Step2 为用户配置云服务使用权限 846
7.3.2.4 Step3 为用户配置 ModelArts 的委托访问授权 847
7.3.2.5 Step4 测试用户权限 848
7.3.3 给子用户配置开发环境基本使用权限 848

7.3.4 给子用户配置训练作业基本使用权限 856
7.3.5 给子用户配置部署上线基本使用权限 861
7.3.6 管理员和开发者权限分离 865
7.3.7 查找 Notebook 实例 869
7.3.8 使用 Cloud Shell 登录训练容器 871
7.3.9 限制用户使用公共资源池 872
7.3.10 给子用户配置文件夹级的 SFS Turbo 访问权限 874
7.4 FAQ 878
7.4.1 使用 ModelArts 时提示“权限不足”，如何解决？ 878
8 Standard 自动学习 881
8.1 使用 ModelArts Standard 自动学习实现口罩检测 881
8.2 使用 ModelArts Standard 自动学习实现垃圾分类 886
9 Standard 开发环境 894
9.1 将 Notebook 的 Conda 环境迁移到 SFS 磁盘 894
9.2 使用 ModelArts PyCharm 插件调试训练 ResNet50 图像分类模型 897
9.3 使用 ModelArts VSCode 插件调试训练 ResNet50 图像分类模型 915
10 Standard 模型训练 925
10.1 使用 ModelArts Standard 自定义算法实现手写数字识别 925
10.2 Standard 专属资源池训练 937
10.2.1 资源选择推荐 937
10.2.2 步骤总览 940
10.2.3 资源购买 941
10.2.4 基本配置 943
10.2.4.1 权限配置 943
10.2.4.1.1 配置 IAM 权限 943
10.2.4.1.2 配置 ModelArts 委托权限 946
10.2.4.1.3 配置 SWR 组织权限 946
10.2.4.1.4 测试用户权限 947
10.2.4.2 创建网络 948
10.2.4.3 专属资源池 VPC 打通 949
10.2.4.4 ECS 服务器挂载 SFS Turbo 存储 950
10.2.4.5 在 ECS 中创建 ma-user 和 ma-group 951
10.2.4.6 obsutil 安装和配置 951
10.2.4.7 （可选）工作空间配置 952
10.2.5 调试与训练 952
10.2.5.1 单机单卡 952
10.2.5.1.1 线下容器镜像构建及调试 952
10.2.5.1.2 上传镜像 955
10.2.5.1.3 上传数据和算法至 OBS（首次使用时需要） 957
10.2.5.1.4 使用 Notebook 进行代码调试 963
10.2.5.1.5 创建训练任务 965

10.2.5.1.6 监控资源 965
10.2.5.2 单机多卡 966
10.2.5.2.1 线下容器镜像构建及调试 966
10.2.5.2.2 上传镜像 966
10.2.5.2.3 上传数据和算法至 SFS（首次使用时需要） 966
10.2.5.2.4 使用 Notebook 进行代码调试 968
10.2.5.2.5 创建训练任务 970
10.2.5.3 多机多卡 971
10.2.5.3.1 线下容器镜像构建及调试 971
10.2.5.3.2 上传镜像 971
10.2.5.3.3 上传数据至 OBS（首次使用时需要） 971
10.2.5.3.4 上传算法至 SFS 971
10.2.5.3.5 使用 Notebook 进行代码调试 973
10.2.5.3.6 创建训练任务 973
10.2.6 FAQ 974
10.2.6.1 CUDA 和 CUDNN 974
10.2.6.1.1 Vnt1 机型软件版本建议 974
10.2.6.1.2 CUDA Compatibility 如何使用？ 974
10.2.6.1.3 专属池驱动版本如何升级？ 975
10.2.6.2 CloudShell 调试方法 975
10.2.6.3 run.sh 脚本测试 ModelArts 训练整体流程 975
10.2.6.4 ModelArts 环境挂载目录说明 976
10.2.6.5 如何查看训练环境变量 977
10.2.6.6 infiniband 驱动的安装 978
10.2.6.7 Tensorboard 的使用 979
10.2.6.8 如何保证训练和调试时文件路径保持一致 982
11 Standard 推理部署 984
11.1 ModelArts Standard 推理服务访问公网方案 984
11.2 端到端运维 ModelArts Standard 推理服务方案 986
11.3 使用自定义引擎在 ModelArts Standard 创建 AI 应用 989
11.4 使用大模型在 ModelArts Standard 创建 AI 应用部署在线服务 992
11.5 第三方推理框架迁移到 ModelArts Standard 推理自定义引擎 995
11.6 ModelArts Standard 推理服务支持 VPC 直连的高速访问通道配置 1005
11.7 ModelArts Standard 的 WebSocket 在线服务全流程开发 1009
12 历史待下线案例 1014
12.1 使用 AI Gallery 的订阅算法实现花卉识别 1014
12.2 示例：从 0 到 1 制作自定义镜像并用于训练（Pytorch+CPU/GPU） 1017
12.3 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU） 1022
12.4 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+Ascend） 1029
12.5 使用 ModelArts Standard 一键完成商超商品识别模型部署 1030
12.6 从 0-1 制作自定义镜像并创建 AI 应用 1031

1 ModelArts 最佳实践案例列表

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

LLM 大语言模型训练推理场景

样例	场景	说明
主流开源大模型基于DevServer适配PyTorch NPU训练指导（6.3.906）	预训练、SFT全参微调训练、LoRA微调训练	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。训练后的模型可用于推理部署，搭建大模型问答助手。
主流开源大模型基于Standard适配PyTorch NPU训练指导（6.3.906）	预训练、SFT全参微调训练、LoRA微调训练	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Standard的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，搭建大模型问答助手。
主流开源大模型基于DevServer适配PyTorch NPU推理指导（6.3.906）	推理部署、推理性能测试、推理精度测试、推理模型量化	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts DevServer的推理部署过程，推理使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可用于搭建大模型问答助手。
主流开源大模型基于Standard适配PyTorch NPU推理指导（6.3.906）	推理部署、推理性能测试、推理精度测试、推理模型量化	介绍主流的开源大模型Llama系列、Qwen系列、Yi系列、Baichuan系列、ChatGLM系列等基于ModelArts Standard的推理部署过程，推理使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可用于搭建大模型问答助手。

AIGC 模型训练推理场景

样例	场景	说明
SDXL基于Standard适配PyTorch NPU的Finetune训练指导（6.3.905）	SDXL、SD1.5模型训练	介绍AIGC模型SDXL、SD1.5基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。训练后的模型可用于推理部署，应用于文生图场景。
SDXL&SD1.5 ComfyUI插件基于DevServer适配PyTorch NPU推理指导（6.3.906）	SDXL、SD1.5模型推理	介绍AIGC模型SDXL、SD1.5基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 启动推理服务后，可应用于文生图场景。
Open-Sora 1.0基于DevServer适配PyTorch NPU训练指导（6.3.905）	Open-Sora 1.0模型训练	介绍Open-Sora 1.0模型基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，应用于文生视频场景。
Qwen-VL基于DevServer适配Pytorch NPU的Finetune训练指导(6.3.906) Qwen-VL基于DevServer适配Pytorch NPU的推理指导(6.3.906)	Qwen-VL模型训练推理	介绍Qwen-VL模型基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，应用于大模型对话场景。
LLaVA模型基于DevServer适配PyTorch NPU预训练指导（6.3.906） LLaVA模型基于DevServer适配PyTorch NPU推理指导（6.3.906）	LLaVA模型训练推理	介绍LLaVA模型基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 训练后的模型可用于推理部署，应用于大模型对话场景。

样例	场景	说明
Open-Clip基于DevServer适配PyTorch NPU训练指导	Open-Clip模型训练	介绍Open-Clip模型基于ModelArts DevServer的训练过程，训练使用PyTorch框架和昇腾NPU计算资源。 应用于AIGC和多模态视频编码器。

数字人场景

样例	场景	说明
Wav2Lip基于DevServer适配PyTorch NPU训练指导（6.3.902） Wav2Lip基于DevServer适配PyTorch NPU推理指导（6.3.906）	Wav2Lip，人脸说话视频模型，训练、推理	Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。 案例主要介绍如何基于ModelArts DevServer上的昇腾NPU资源进行模型训练推理。

ModelArts Standard 权限配置

样例	对应功能	场景	说明
ModelArts Standard权限管理	IAM权限配置、权限管理	为子用户配置权限	当一个华为云账号下需创建多个IAM用户（即子用户）时，可参考此样例，为IAM用户赋予使用ModelArts所需的权限。避免IAM用户因权限问题导致使用时出现异常。

ModelArts Standard 自动学习案例

表 1-1 自动学习样例列表

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

ModelArts Standard 开发工具案例

表 1-2 Notebook 样例列表

样例	镜像	对应功能	场景	说明
使用ModelArts PyCharm插件调试训练ResNet50图像分类模型	MindSpore	PyCharm ToolKit工具	目标检测	本案例介绍如何在本地进行MindSpore模型开发，并将模型迁移至ModelArts训练。
使用ModelArts VSCode插件调试训练ResNet50图像分类模型	MindSpore	VS Code Toolkit工具	目标检测	本案例以Ascend Model Zoo为例，介绍如何通过VS Code插件及ModelArts Notebook进行云端数据调试及模型开发。

ModelArts Standard 模型训练案例

表 1-3 自定义算法样例列表

样例	镜像	对应功能	场景	说明
使用ModelArts Standard自定义算法实现手写数字识别	PyTorch	自定义算法	手写数字识别	使用用户自己的算法，训练得到手写数字识别模型，并部署后进行预测。
从0制作自定义镜像并用于训练（PyTorch+CPU/GPU）	PyTorch	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。
从0制作自定义镜像并用于训练（MPI+CPU/GPU）	MPI	镜像制作 自定义镜像训练	-	此案例介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

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

ModelArts Standard 推理部署

表 1-4 推理部署列表

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

第三方案例列表

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

表 1-5 第三方案例列表

分类	文章名称	作者
Standard自动学习	2步打通ModelArts和Astro实现AI应用落地	胡琦
Standard开发环境	想不想让一张静态的照片动起来	林欣

分类	文章名称	作者
	基于TensorFlow训练轻量化ssdlite_mbv2人脸手机检测模型	AI练习生
	基于ModelArts的手写数字识别	AXYZdong
	AI 文字编辑图片 instruct-pix2pix 案例	AXYZdong
Standard推理部署	上线二维码检测识别服务	林欣
	使用ModelArts对8类常见生活垃圾进行分类	福州司马懿
	使用ModelArts搭建"花卉种类识别"服务	福州司马懿

2 昇腾能力应用地图

ModelArts支持如下开源模型基于Ascend卡进行训练和推理。

主流三方大模型

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

表 2-1 LLM 模型训练能力

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
Llama2	Llama2-7b Llama2-13b Llama2-70b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	• 主流开源大模型基于DevServer适配ModelLink PyTorch NPU训练指导（6.3.907） • 主流开源大模型基于DevServer适配LlamaFactory PyTorch NPU训练指导（6.3.907） • 主流开源大模型基于Standard+OBS适配PyTorch NPU训练指导（6.3.907） • 主流开源大模型基于Standard+OBS+SFS适配PyTorch NPU训练指导（6.3.907）
Llama3	Llama3-8b Llama3-70b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
Qwen	qwen-7b qwen-14b qwen-72b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
Qwen1.5	qwen1.5-7b qwen1.5-14b qwen1.5-32b qwen1.5-72b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
Qwen2	qwen2-0.5b qwen2-1.5b qwen2-7b qwen2-72b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
Yi	yi-6b yi-34b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
ChatGLM v3	glm3-6b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
GLMv4	glm4-9b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	
Baichuan 2	baichuan2-13b	预训练、SFT全参微调、LoRA微调	ModelLInk LlamaFactory	

表 2-2 LLM 模型推理能力

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
Llama	Llama-7b Llama-13b Llama-65b	推理	Ascend-vLLM	• 主流开源大模型基于Standard适配PyTorch NPU推理指导（6.3.907） • 主流开源大模型基于DevServer适配PyTorch NPU推理指导（6.3.907）
Llama2	Llama2-7b Llama2-13b Llama2-70b	推理	Ascend-vLLM	
Llama3	Llama3-8b Llama3-70b	推理	Ascend-vLLM	
Yi	yi-6b yi-9b yi-34b	推理	Ascend-vLLM	
deepseek	deepseek-llm-7b deepseek-llm-67b deepseek-coder-instruct-33b	推理	Ascend-vLLM	
Qwen	qwen-7b qwen-14b qwen-72b	推理	Ascend-vLLM	
Qwen1.5	qwen1.5-0.5b qwen1.5-7b qwen1.5-1.8b qwen1.5-14b qwen1.5-32b qwen1.5-72b qwen1.5-110b	推理	Ascend-vLLM	
Qwen2	qwen2-0.5b qwen2-1.5b qwen2-7b qwen2-72b	推理	Ascend-vLLM	
Baichuan 2	baichuan2-7b baichuan2-13b	推理	Ascend-vLLM	

支持模型	支持模型参数量	应用场景	软件技术栈	指导文档
gemmma	gemmma-2b gemmma-7b	推理	Ascend-vLLM	
ChatGLM2	chatglm2-6b	推理	Ascend-vLLM	
ChatGLM4	chatglm3-6b	推理	Ascend-vLLM	
GLMv4	glm4-9b	推理	Ascend-vLLM	
mistral	mistral-7b mistral-8x7b	推理	Ascend-vLLM	

AIGC 模型开箱

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

表 2-3 AIGC 模型

模型名称	应用场景	软件技术栈	指导文档
Stable Diffusion 1.5 Stable Diffusion XL Stable Diffusion 3	SFT全量微调训练 LoRA微调训练	Diffusers训练、Kohya训练、PyTorch	**SD1.5&SDXL Diffusers框架基于DevServer适配PyTorch NPU训练指导（6.3.907）** **SDXL基于Standard适配PyTorch NPU的LoRA训练指导（6.3.907）** **SD1.5&SDXL Koyha框架基于DevServer适配PyTorch NPU训练指导（6.3.907）**
	WebUI推理	WebUI推理、PyTorch	**SD WEBUI套件适配PyTorch NPU的推理指导（6.3.907）**
	Diffusers推理	diffusers推理、PyTorch	**SD1.5&SDXL Diffusers框架基于DevServer适配PyTorch NPU推理指导（6.3.907）**
Open-Sora	训练 推理	PyTorch	**Open-Sora1.2基于DevServer适配PyTorch NPU训练推理指导（6.3.907）**
Open-Sora-Plan	训练 推理	PyTorch	**Open-Sora-Plan1.0基于DevServer适配PyTorch NPU训练推理指导（6.3.907）**

模型名称	应用场景	软件技术栈	指导文档
Qwen-VL	训练 推理	PyTorch	Qwen-VL基于DevServer适配Pytorch NPU的Finetune训练指导(6.3.906) Qwen-VL基于DevServer适配Pytorch NPU的推理指导(6.3.906)
LLaVA	训练 推理	PyTorch	LLaVA模型基于DevServer适配PyTorch NPU预训练指导（6.3.906） LLaVA模型基于DevServer适配PyTorch NPU推理指导（6.3.906）
Open-clip	训练 推理	PyTorch	Open-Clip基于DevServer适配PyTorch NPU训练指导

表 2-4 数字人模型

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

3 LLM 大语言模型训练推理

3.1 主流开源大模型基于 DevServer 适配 ModelLink PyTorch NPU 训练指导（6.3.907）

3.1.1 场景介绍

方案概览

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

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

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表3-3**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite DevServer。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如**表3-1**所示。

表 3-1 支持的模型列表

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

序号	支持模型	支持模型参数量	权重文件获取地址
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**
18		qwen2-1.5b	**https://huggingface.co/Qwen/Qwen2-1.5B-Instruct**
19		qwen2-7b	**https://huggingface.co/Qwen/Qwen2-7B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 3-1 操作流程图

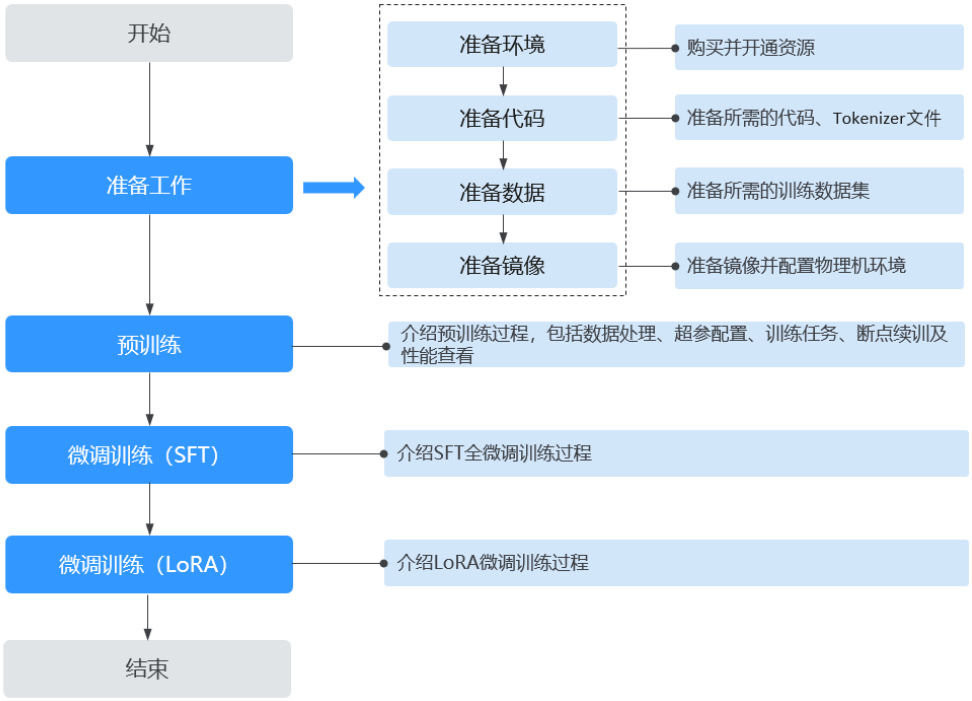


表 3-2 操作任务流程说明

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

阶段	任务	说明
微调训练	SFT全参微调	介绍如何进行SFT全参微调、超参配置、训练任务、性能查看。
	LoRA微调训练	介绍如何进行LoRA微调、超参配置、训练任务、性能查看。

3.1.2 准备工作

3.1.2.1 准备环境

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

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表3-11**。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.1.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表3-3**所示，模型列表、对应的开源权重获取地址如**表3-4**所示。

表 3-3 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

表 3-4 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
        |——AscendSpeed         # 基于AscendSpeed的训练代码
             |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
             |——scripts/             # 训练需要的启动脚本
                  |——llama2          # llama2系列模型执行脚本的文件夹
                  |——llama3          # llama3系列模型执行脚本的文件夹
                  |——qwen            # Qwen系列模型执行脚本的文件夹
                  |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                  |—— ...
                  |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                  |—— install.sh       # 环境部署脚本
             |——src/                  # 启动命令行封装脚本，在install.sh里面自动构建
   |——llm_inference                   # 推理代码包
   |——llm_tools                       # 推理工具
```

工作目录介绍

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

```
${workdir}（例如/home/ma-user/ws）
|——llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
```

```
    |── AscendSpeed              # 代码目录
          |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── data              # 预处理后数据
              |── pretrain    # 预训练加载的数据
              |── finetune    # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换magatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── logs              # 训练过程中日志（loss、吞吐性能）
                  |─── saved_models
                |── lora              # lora微调输出权重
                |── sft               # 增量训练输出权重
                |── pretrain          # 预训练输出权重
|── tokenizers                    #原始权重目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── models                        #tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                  #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

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

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

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

   具体步骤如下：

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

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

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

3.1.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

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

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

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

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

3.1.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-5 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a

表 3-6 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
驱动	23.0.5
PyTorch	2.1.0

Step1 检查环境

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

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

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

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
```

```
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
```
docker exec -it ${container_name} bash
```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```
4. 使用ma-user用户安装依赖包。
```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/AscendSpeed
#执行安装命令
sh scripts/install.sh
```
5. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：
```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
      |——AscendSpeed           # 基于AscendSpeed的训练代码
            |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
            |——scripts/             # 训练需要的启动脚本
            |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
      |——Megatron-LM/          # 适配昇腾的Megatron-LM训练框架
      |——MindSpeed/            # MindSpeed昇腾大模型加速库
      |——ModelLink/            # ModelLink端到端的大语言模型方案
            |——megatron/       # 注意：该文件夹从Megatron-LM中复制得到
            |——...
```

3.1.3 预训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**llama2-70b和llama2-13b预训练**为例，执行脚本为**0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-7**所示。其他超参均有默认值，可以参考**表3-10**按照实际需求修改。

表 3-7 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的CKPT、PLOG、LOG文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型CKPT文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。

参数	示例值	参数说明
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志LOG文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志PLOG文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

请根据**Step2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

示例:

```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_pretrain_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_pretrain_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

示例:

```
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_pretrain_13b.sh
```

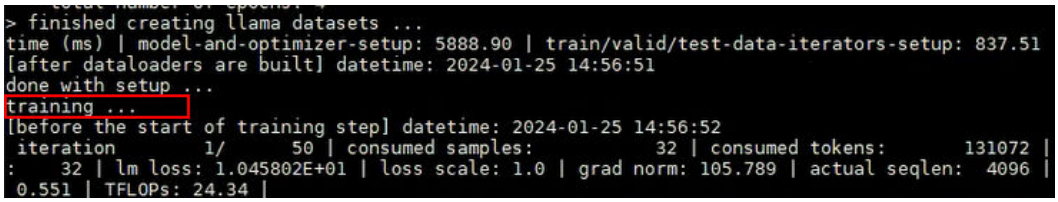
或者:

```
sh scripts/llama2/0_pl_pretrain_13b.sh
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-2 等待模型载入



最后，请参考**查看日志和性能**章节查看预训练的日志和性能。

3.1.4 SFT 全参微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-7**所示。其他超参均有默认值，可以参考**表3-10**按照实际需求修改。

表 3-8 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。

参数	示例值	参数说明
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

示例:

```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_sft_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_sft_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_sft_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_sft_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
示例:
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_sft_13b.sh
或者:
sh scripts/llama2/0_pl_sft_13b.sh
```

最后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

3.1.5 LoRA 微调训练

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-7**所示。其他超参均有默认值，可以参考**表3-10**按照实际需求修改。

表 3-9 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/models/llama2-13B	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13B	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。

参数	示例值	参数说明
INPUT_PROCESSED_DIR	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/ws/llm_train/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表3-11**所示。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

示例:

```
# 第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=0 sh scripts/llama2/0_pl_lora_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=1 sh scripts/llama2/0_pl_lora_70b.sh
```

```
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=2 sh scripts/llama2/0_pl_lora_70b.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=4 NODE_RANK=3 sh scripts/llama2/0_pl_lora_70b.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

```
示例:
MASTER_ADDR=localhost NNODES=1 NODE_RANK=0 sh scripts/llama2/0_pl_lora_13b.sh
或者:
sh scripts/llama2/0_pl_lora_13b.sh
```

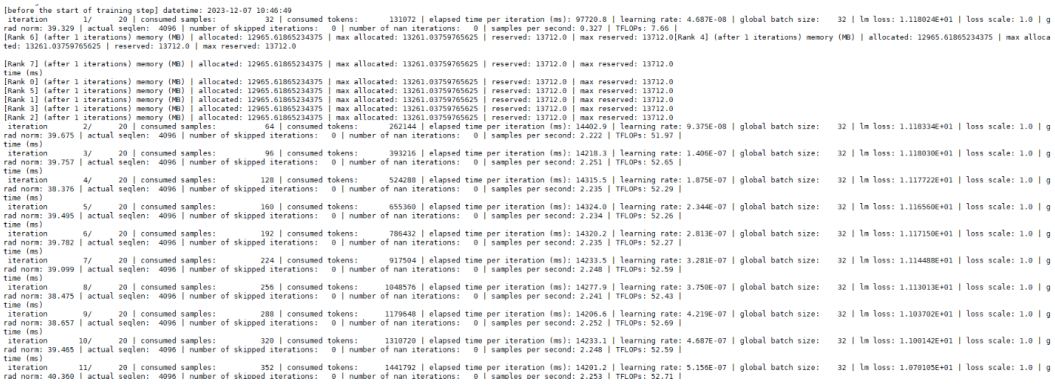
最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。

3.1.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-3 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

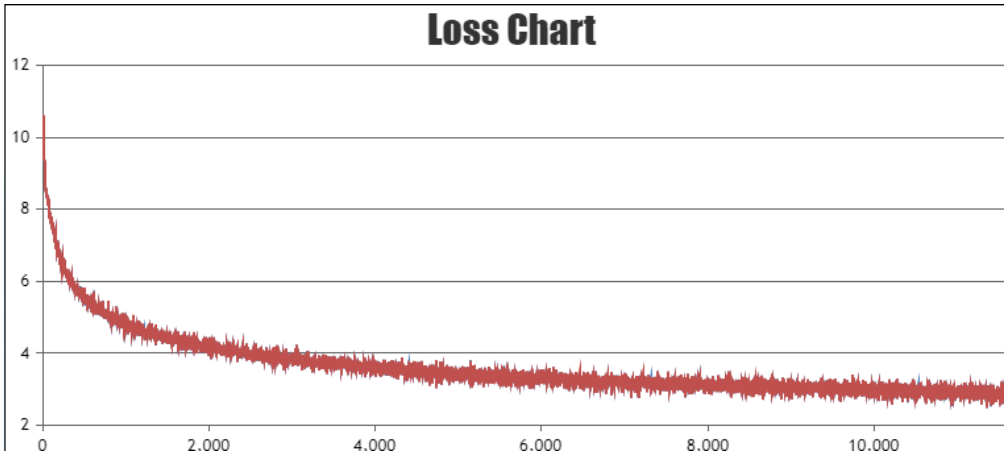
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表3-10**。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-4**所示。

单节点训练：训练过程中的loss直接打印在窗口上。

多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-4 Loss 收敛情况（示意图）



3.1.7 训练脚本说明

3.1.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 3-10 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。

参数	示例值	参数说明
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表3-11所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-11 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

3.1.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/fintune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的hanler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

  - 先调用self.get_tokenized_data()对数据集进行encode
  - self.get_tokenized_data()中调用self._filter方法处理每一个sample
  - self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

  - 对数据集 full_prompt 中的 user_prompt 进行 mask 操作。

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
```

```
    "input_ids": [input_ids],
    "attention_mask": [attention_mask],
    "labels": [labels]
}
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

i. 对user和assiant的文本进行清洗

ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assiantant_ids

iii. input_ids是user_ids和assiantant_ids的拼接

iv. labels与input_ids对应，用-100替换user_ids的token,只保留assiantant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 3-12 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。

环境变量	示例	参数说明
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.1.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋

值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 3-13 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face

参数	示例	参数说明
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.1.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图3-5所示。

图 3-5 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-6所示。

图 3-6 修改 ChatGLMv3-6B tokenizer 文件

```
270         # Load from model_defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-7所示。

图 3-7 修改 ChatGLMv3-6B tokenizer 文件

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

294行要添加注释，修改后如图3-8所示。

图 3-8 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model_defaults
294         #assert self.padding_side == "left"
295
```

314至323行要修改，修改后如图3-9所示。

图 3-9 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-10所示。

图 3-10 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.1.8 常见错误原因和解决方法

3.1.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表3-11进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

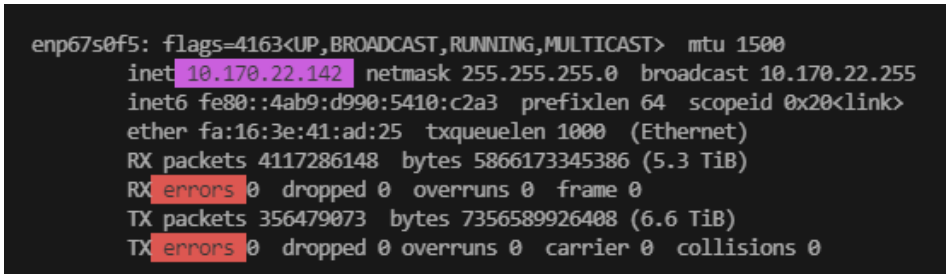
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

3.1.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 3-11 网卡名称错误



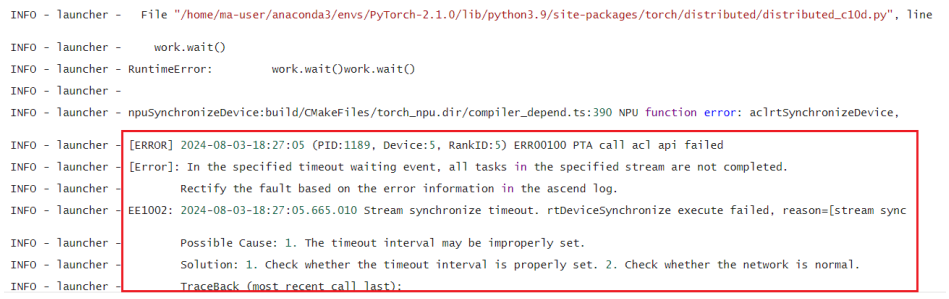

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

3.1.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 3-12 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

3.2 主流开源大模型基于 DevServer 适配 LlamaFactory PyTorch NPU 训练指导（6.3.907）

3.2.1 场景介绍

方案概览

本文档利用训练框架LlamaFactory+华为自研Ascend Snt9B硬件，为用户提供了常见主流开源大模型在ModelArts Lite DevServer上的微调方案，包括sft全参和lora 微调。

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

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表3-16**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite DevServer。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。
- DevServer驱动版本要求23.0.5

训练支持的模型列表

本方案支持以下模型的训练，如**表3-14**所示。

表 3-14 支持的模型列表及权重文件地址

支持模型	Template	支持模型参数量	权重文件获取地址
Llama3	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
Qwen1.5	qwen	qwen1.5-0.5b	**https://huggingface.co/Qwen/Qwen1.5-0.5B**
		qwen1.5-1.8b	**https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat**
		qwen1.5-4b	**https://huggingface.co/Qwen/Qwen1.5-4B**
		qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**

支持模型	Template	支持模型参数量	权重文件获取地址
		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
Yi	yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
Qwen2	qwen	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
Falcon2	falcon	falcon-11B	https://huggingface.co/tiiuae/falcon-11B

表 3-15 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite DevServer运行的，需要购买并开通DevServer资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
微调训练	指令监督微调训练	介绍如何进行SFT全参微调/lora微调、训练任务、性能查看。

3.2.2 准备工作

3.2.2.1 准备环境

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

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表3-22**。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.2.2.2 准备代码

本教程中用到的训练、推理代码如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表3-16**所示，模型列表、对应的开源权重获取地址如**表3-14**所示。

表 3-16 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
|——AscendCloud-LLM
   |——llm_train                    # 模型训练代码包
      |——LLaMAFactory              # 基于LLaMAFactory的训练代码
         |——ascendcloud_patch/     # 针对昇腾云平台适配的功能补丁包
         |——demo.yaml              # 样例yaml配置文件
```

```
        |——demo.sh              # 指令微调启动shell脚本
        |——intall.sh           # 需要的依赖包
        |——LLaMA-Factory        # LLaMAFactory的代码目录
    |——AscendSpeed              # 基于AscendSpeed的训练代码
```

工作目录介绍

详细的工作目录参考如下，建议参考以下要求设置工作目录。

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                  #解压代码包后自动生成的代码目录，无需用户创建
    |—— LLaMAFactory              # 代码目录
        |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |——demo.sh              # 指令微调启动shell脚本
        |——demo.yaml            # 样例yaml配置文件
        |——intall.sh           # 需要的依赖包
        |——LLaMA-Factory        # 执行install.sh后生成此目录,容器内执行参考Step3 启动容器镜像
            |——data             # 原始数据目录，如使用自定义数据，参考准备数据（可选）
|—— tokenizers                  #原始权重/tokenizer目录，用户手动创建，用户根据实际规划目录修改，后续操作步骤中会提示
    |—— Qwen2-72B
# 输出权重及日志路径，用户可根据实际自行规划，无需手动创建，此路径对应表3-19表格中output_dir参数值
|—— saved_dir_for_output_lf          # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |—— ${model_name}               # 模型名称,根据实际训练模型创建，训练完成权重文件及日志目录
```

上传代码和权重文件到工作环境

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

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

3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/{Model_Name}目录，用户根据自己实际规划路径修改；如Qwen2-72B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen2-72B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen2-72B
   ```

3.2.2.3 准备镜像环境

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-17 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a

表 3-18 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
驱动	23.0.5
PyTorch	2.1.0

Step1 检查环境

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

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

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

2. 检查docker是否安装。

```
docker -v   #检查docker是否安装
```

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
```

```
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
--cpus 192 \
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如llamafactory。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. **修改**目录**权限**，上传代码和数据到宿主机时使用的是root用户，如**用ma-user用户**训练，此处需要执行如下命令统一文件权限。

```
#统一文件权限
chmod -R 777  ${work_dir}
# ${work_dir}:/home/ma-user/ws 宿主机代码和数据目录
#例如：  chmod -R 777  /home/ma-user/ws
```

3. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。

```
docker exec -it ${container_name} bash
```

4. 使用ma-user用户安装依赖包。

```
#进入scripts目录换
cd /home/ma-user/ws/llm_train/LLaMAFactory
#执行安装命令,安装依赖包及/LLaMAFactory代码包
sh install.sh
```

3.2.2.4 准备数据（可选）

说明

此小节为**自定义数据**集执行过程，如非自定义数据集此小节忽略。

本教程使用到的是LLamaFactory代码包自带数据集。您也可以自行准备数据集，目前指令微调数据集我们支持alpaca格式和sharegpt格式的数据集；使用自定义数据集时，请更新代码目录下data/dataset_info.json文件；请务必在dataset_info.json文件中添加*数据集描述*。

关于数据集文件的格式及配置，请参考**data/README_zh.md**的内容。可以使用HuggingFace/ModelScope上的数据集或加载本地数据集。

上传自定义数据到指定目录

将下载的原始数据存放在{work_dir}/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data目录下。
   ```
   cd /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
   ```
2. 将自定义原始数据如demo.json放置在此处。

   数据存放参考目录结构如下：
   ```
   ${workdir}（例如/home/ma-user/ws/llm_train ）
    |── LLaMAFactory/data
        |── alpaca_en_demo.json                # 代码原有数据集
        |── identity.json                      # 代码原有数据集
        ...
        |── demo.json                          # 自定义数据集
   ```
3. 更新代码目录下 data/dataset_info.json 文件。关于数据集文件的格式及配置，请参考 **data/README_zh.md** 的内容。
   ```
   vim dataset_info.json
   ```

3.2.3 指令监督微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段**已经上传**权重文件、自定义数据集，可以**忽略**此步骤。

- **未上传**训练权重文件，具体参考**上传代码和权重文件到工作环境**。
- 使用**自定义**数据集训练**未上传**自定义数据集。具体参考**上传自定义数据到指定目录**章节并**更新**dataset_info.json 文件。

Step2 修改训练 yaml 文件配置

LlamaFactroy配置文件主要为yaml文件为主，启动训练前需修改样例yaml配置文件配置，样例yaml配置文件在代码目录下的{work_dir}/llm_train/LLaMAFactory/demo.yaml。修改详细步骤如下所示：

步骤1 选择指令微调类型

- sft，复制**sft_yaml样例模板**内容覆盖demo.yaml文件内容。
- lora，复制**lora_yaml样例模板**内容覆盖demo.yaml文件内容。

步骤2 修改**yaml文件**(demo.yaml)的参数如**表3-19**所示

表 3-19 修改重要参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时存放目录绝对或相对路径。请根据实际规划修改。
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用Qwen模板进行训练，模板选择可参照**表3-14**中的**template列**

参数	示例值	参数说明
output_dir	/home/ma-user/ws/Qwen2-72B/sft-4096	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下。用户根据自己实际要求适配。
per_device_train_batch_size	1	指定每个设备的训练批次大小
gradient_accumulation_steps	8	指定梯度累积的步数，这可以增加批次大小而不增加内存消耗。可根据自己要求适配
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。可根据自己要求适配
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
dataset	identity,alpaca_en_demo	【**可选**】指定用于训练的数据集，数据集都放置在此处为identity，alpaca_en_demo表示使用了两个数据集，一个是 identity，一个是alpaca_en_demo。如选用定义数据请参考**准备数据（可选）**配置dataset_info.json文件
dataset_dir	/home/ma-user/ws/LLaMAFactory/LLaMA-Factory/data	【**可选**】自定义数据集dataset_info.json配置文件**绝对路径**；如使用自定义数据集，yaml配置文件需添加此参数。

步骤3 是否选择加速深度学习训练框架Deepspeed，可参考**表3-21**选择不同的框架

- 是，选用ZeRO (Zero Redundancy Optimizer)优化器
  - ZeRO-0，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z0_config.json
    ```
  - ZeRO-1，配置以下参数，并复制**ds_z1_config.json样例模板**至工作目录/home/ma-user/LLaMAFactory/LLaMA-Factory/examples/deepspeed
    ```
    deepspeed: examples/deepspeed/ds_z1_config.json
    ```
  - ZeRO-2，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z2_config.json
    ```
  - ZeRO-3，配置以下参数
    ```
    deepspeed: examples/deepspeed/ds_z3_config.json
    ```
- 否，默认选用Accelerate加速深度学习训练框架，**注释**掉deepspeed参数。

步骤4 是否使用固定句长

- 是，配置以下参数
  ```
  packing: true
  ```
- 否，默认使用动态句长，**注释**掉packing参数。

步骤5 选用数据精度格式，以下参数二选一。

- bf16，配置以下参数
  ```
  bf16: true
  ```
- fp16，配置以下参数
  ```
  fp16: true
  ```

步骤6 是否使用自定义数据集

- 是，参考**准备数据（可选）**后，填写自定义注册后数据集前缀名称及数据集**绝对路径**，参考**表3-19**dataset_dir行，如demo.json数据集前缀则为demo
  ```
  dataset: demo
  dataset_dir: /home/ma-user/ws/llm_train/LLaMAFactory/LLaMA-Factory/data
  ```
- 否，使用代码包自带数据集，配置参数如
  ```
  dataset: identity,alpaca_en_demo
  ```

步骤7 如需其他配置参数，可参考**表3-20**按照实际需求修改

----结束

Step3 启动训练脚本

说明

- **启动训练前**需**修改**启动训练脚本**demo.sh 内容**。具体请参考**•修改启动脚本**。
- 对于**falcon-11B**训练任务开始前，需**手动替换tokenizer**中的config.json，具体请参见**falcon-11B模型**。

修改完yaml配置文件后，启动训练脚本；模型不同最少npu卡数不同，npu卡数建议值可参考**模型NPU卡数取值表**。

- **修改启动脚本**

进入代码目录{work_dir}/llm_train/LLaMAFactory 下**修改启动脚本**，其中{work_dir}为容器挂载路径；修改demo.sh 最后一行代码：

将demo.yaml配置文件路径**修改**为自己**实际绝对路径:**{work_dir}/llm_train/LLaMAFactory/demo.yaml，例如将以下命令：

```
FORCE_TORCHRUN=1 llamafactory-cli train /data/openllm_gy1/user/lmz/poc_package/LLaMAFactory/demo.yaml
```

修改为：

```
FORCE_TORCHRUN=1 llamafactory-cli train  /home/ma-user/ws/llm_train/LLaMAFactory/demo.yaml
```

- **多机启动**

多台机器执行训练启动命令如下。

```
多机执行命令为：sh demo.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
#第一台节点
sh demo.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh demo.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh demo.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh demo.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

- **单机启动**

一般小于等于14B模型可选择单机启动，操作过程与多机启动相同，只需修改对应参数即可，可以选用单机启动。

进入代码目录**/home/ma-user/ws/llm_train/LLaMAFactory**下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh  demo.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh demo.sh localhost 1 0
```

单机如需指定**训练卡数**训练可使用ASCEND_RT_VISIBLE_DEVICES变量指定卡ID，使用执行命令如下：

```
ASCEND_RT_VISIBLE_DEVICES=0,1,2,3 sh demo.sh localhost 1 0
```

其中ASCEND_RT_VISIBLE_DEVICES=0,1,2,3指使用0-3卡执行训练任务

- **训练成功标志**

“******* train metrics *******”关键字打印

```
  warnings.warn(
[INFO|tokenization_utils_base.py:2513] 2024-08-02 19:19:18,468 >> tok
[INFO|tokenization_utils_base.py:2522] 2024-08-02 19:19:18,468 >> Spe
***** train metrics *****
  epoch                    =     4.9863
  num_input_tokens_seen    =    1013520
  total_flos               = 32944743GF
  train_loss               =     0.9493
  train_runtime            = 0:58:44.11
  train_samples_per_second =      1.548
  train_steps_per_second   =      0.193
  train_tokens_per_second  =    288.277
```

训练完成后，请参考**查看日志和性能**章节查看指令微调的日志和性能。

说明

1、如训练过程中遇到“NPU out of memory”“Permission denied” 问题可参考 **附录：指令微调训练常见问题**解决

2、训练中遇到"**ImportError:** This modeling file requires the following packages that were not found in your environment: **flash_attn**. Run `pip install flash_attn`"请参考**附录：指令微调训练常见问题**问题3小节。

3.2.4 查看日志和性能

查看日志

训练过程中，训练日志会在第一个的Rank节点打印。

图 3-13 打印训练日志

```
  0%|              | 0/70 [00:00<?, ?it/s][W compiler_depend.ts:103] Warning: Non finite check and unscale on NPU device! (function operator())
Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32768.0

  1%|              | 1/70 [00:10<11:45, 10.23s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16384.0

  3%|▏             | 2/70 [00:19<10:42,  9.45s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8192.0

  4%|▍             | 3/70 [00:28<10:16,  9.20s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4096.0

  6%|▌             | 4/70 [00:36<09:59,  9.08s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2048.0

  7%|▋             | 5/70 [00:45<09:45,  9.02s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 1024.0

  9%|▊             | 6/70 [00:54<09:34,  8.98s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 512.0

 10%|▉             | 7/70 [01:03<09:23,  8.95s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 256.0

 11%|█             | 8/70 [01:12<09:14,  8.94s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 128.0

 13%|█▏            | 9/70 [01:21<09:04,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 64.0

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]

{'loss': 0.0, 'grad_norm': nan, 'learning_rate': 0.0, 'epoch': 1.43}

 14%|█▎            | 10/70 [01:30<08:55,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 32.0

 16%|█▍            | 11/70 [01:39<08:46,  8.92s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 16.0

 17%|█▌            | 12/70 [01:48<08:36,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 8.0

 19%|█▋            | 13/70 [01:57<08:27,  8.91s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 4.0

 20%|█▊            | 14/70 [02:05<08:18,  8.90s/it]Gradient overflow. Skipping step
Loss scaler reducing loss scale to 2.0
```

训练完成后，如果需要单独获取训练日志文件，日志存放在第一个的Rank节点中；日志存放路径为：对应**表3-19**表格中**output_dir**参数值路径下的**trainer_log.jsonl**文件

查看性能

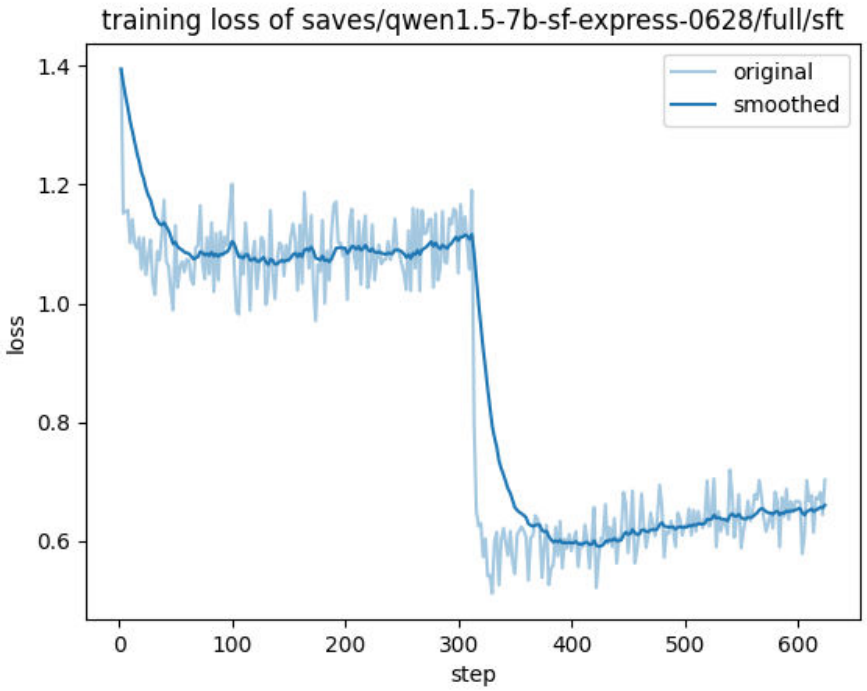
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：可通过**表3-19**表格中**output_dir**参数值路径下的**train_results.json**查看性能。吞吐计算公式为"num_input_tokens_seen / train_runtime / 训练卡数"。相关参数可查看**表3-19**。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。loss收敛图存放路径对应**表3-19**表格中**output_dir**参数值路径下的**training_loss.png**中也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-14**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在第一个节点上。

图 3-14 Loss 收敛情况（示意图）



3.2.5 训练脚本说明

3.2.5.1 yaml 配置文件参数配置说明

本小节主要详细描述demo_yaml样例配置文件、配置参数说明，用户可根据实际自行选择其需要的参数。

表 3-20 模型训练脚本参数

参数	示例值	参数说明
model_name_or_path	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放绝对或相对路径。请根据实际规划修改。
do_train	true	指示脚本执行训练步骤，用来控制是否进行模型训练的。如果设置为true，则会进行模型训练；如果设置为false，则不会进行模型训练。
cutoff_len	4096	文本处理时的最大长度，此处为4096，用户可根据自己要求适配。
packing	true	**可选项**。当选用**静态数句长度**时，可将不足于文本处理时的最大长度数据弥补到文本处理时的最大长度;当选用**动态数句长度**则**去掉**此参数。

参数	示例值	参数说明
deepspeed	examples/deepspeed/ds_z3_config.json	**可选项。**用于指定DeepSpeed的配置文件相对或绝对路径。DeepSpeed是一个开源库，用于加速深度学习训练。通过使用DeepSpeed，可以实现如混合精度训练、ZeRO内存优化等高级特性，以提高训练效率和性能
stage	sft	表示训练类型。可选择值：[pt、sf、rm、ppo]，pt代表预训练，sft代表指令监督微调，rm代表奖励模型训练，ppo代表PPO训练。
finetuning_type	full	用于指定微调的类型，可选择值【full、lora】如果设置为"full"，则对整个模型进行微调。这意味着在微调过程中，除了输出层外，模型的所有参数都将被调整以适应新的任务。
dataset	identity,alpaca_en_demo	指定用于训练的数据集，数据集都放置在此处为identity，alpaca_en_demo表示使用了两个数据集，一个是identity，一个是alpaca_en_demo。如选用定义数据请参考**准备数据（可选）**
template	qwen	**必须修改**。用于指定模板。如果设置为"qwen"，则使用QWEN模板进行训练，模板选择可参照**表3-14**中的**template列**
max_samples	1000	用于指定训练过程中使用的最大样本数量。如果设置了这个参数，训练过程将只使用指定数量的样本，而忽略其他样本。这可以用于控制训练过程的规模和计算需求
overwrite_cache	true	用于指定是否覆盖缓存。如果设置为"overwrite_cache"，则在训练过程中覆盖缓存。这通常在数据集发生变化，或者需要重新生成缓存时使用
preprocessing_num_workers	16	用于指定**预处理数据的工作线程数**。随着线程数的增加，预处理的速度也会提高，但也会增加内存的使用。
per_device_train_batch_size	1	**必须修改**，指定每个设备的训练批次大小。
gradient_accumulation_steps	8	指定梯度累积的步数,这可以增加批次大小而不增加内存消耗。

参数	示例值	参数说明
output_dir	/home/ma-user/ws/tokenizers/Qwen2-72B	**必须修改**。指定输出目录。训练过程中生成的模型参数和日志文件将保存在这个目录下
logging_steps	2	用于指定模型训练过程中，多少步输出一次日志。日志包括了训练进度、学习率、损失值等信息。建议设置
save_steps	500	指定模型训练过程中，每多少步保存一次模型。保存的模型可以用于后续的训练或推理任务
plot_loss	true	用于指定是否绘制损失曲线。如果设置为"true"，则在训练结束后，将损失曲线保存为图片
overwrite_output_dir	true	是否覆盖输出目录。如果设置为"true"，则在每次训练开始时，都会清空输出目录，以便保存新的训练结果。
num_train_epochs	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
fp16/bf16	true	使用混合精度格式，减少内存使用和计算需求。**二者选其一**
learning_rate	1.0e-5	指定学习率

sft_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: full
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
template: qwen
cutoff_len: 4096
packing: true
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/sft
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
```

```
warmup_ratio: 0.1
fp16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

lora_yaml 样例模板

```
### model
model_name_or_path: /home/ma-user/ws/tokenizers/Qwen2-72B
### method
stage: sft
do_train: true
finetuning_type: lora
lora_target: all
deepspeed: examples/deepspeed/ds_z3_config.json
### dataset
dataset: identity,alpaca_en_demo
template: qwen
cutoff_len: 4096
packing: true
max_samples: 1000
overwrite_cache: true
preprocessing_num_workers: 16
### output
output_dir: /home/ma-user/ws/tokenizers/Qwen2-72B/lora
logging_steps: 2
save_steps: 5000
plot_loss: true
overwrite_output_dir: true
### train
per_device_train_batch_size: 1
gradient_accumulation_steps: 8
learning_rate: 1.0e-5
num_train_epochs: 10.0
lr_scheduler_type: cosine
warmup_ratio: 0.1
fp16: true
ddp_timeout: 180000000
include_tokens_per_second: true
include_num_input_tokens_seen: true
```

ds_z1_config.json 样例模板

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "zero_optimization": {
    "stage": 1,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "overlap_comm": true,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "contiguous_gradients": true,
```

```
    "round_robin_gradients": true
  }
}
```

3.2.5.2 各个模型深度学习训练加速框架的选择

1. LlamaFactory框架使用两种训练框架：
   - DeepSpeed和Accelerate都是针对深度学习训练加速的工具，但是它们的实现方式和应用场景有所不同。
     - DeepSpeed是一种深度学习加速框架，主要针对大规模模型和大规模数据集的训练。DeepSpeed的核心思想是在单个GPU上实现大规模模型并行训练，从而提高训练速度。DeepSpeed提供了一系列的优化技术，如ZeRO内存优化、分布式训练等，可以帮助用户更好地利用多个GPU进行训练
     - Accelerate是一种深度学习加速框架，主要针对分布式训练场景。Accelerate的核心思想是通过模型并行和数据并行来实现分布式训练，从而提高训练速度。Accelerate提供了一系列的优化技术，如模型切分、梯度累积等，可以帮助用户更好地利用多个节点进行训练。
2. 各个模型选用加速框架

表 3-21 模型加速框架建议表

序号	模型参数量	文本序列长度	优化工具(Deepspeed&Accelerator)
0	小于4B	cutoff_len=4096	Deepspeed-ZeRO-0
		cutoff_len=8192	Deepspeed-ZeRO-0
1	小于7B	cutoff_len=4096	Deepspeed-ZeRO-1
		cutoff_len=8192	Deepspeed-ZeRO-1
2	7B至13B	cutoff_len=4096	Deepspeed-ZeRO-2
		cutoff_len=8192	Deepspeed-ZeRO-2
3	14B-72B	cutoff_len=4096	Deepspeed-ZeRO-3
		cutoff_len=8192	Deepspeed-ZeRO-3

说明

以上为建议值，上述参数值仅供参考，如需配置其他加速框架或ZeRO (Zero Redundancy Optimizer)优化器用户可自行选用配置。

3.2.5.3 模型 NPU 卡数取值表

不同模型推荐的训练参数和计算规格要求如表3-22所示。规格与节点数中的**1*节点 & 4*Ascend**表示**单机4卡**，以此类推

表 3-22 模型 NPU 卡数取值表

支持模型	支持模型参数量	文本序列长度	训练类型	Zero并行	规格与节点数
llama 3	70B	cutoff_len=4096	lora	per_device_train_batch_size=1	2*节点 & 8*Ascend
			sft	per_device_train_batch_size=1	8*节点 & 8*Ascend
		cutoff_len=8192	lora	per_device_train_batch_size=1	2*节点 & 8*Ascend
			sft	per_device_train_batch_size=1	8*节点 & 8*Ascend
	8B	cutoff_len=4096/8192	lora sft	per_device_train_batch_size=1	1*节点 & 1*Ascend 1*节点 & 4*Ascend
Qwen 2	72B	cutoff_len=4096	lora sft	per_device_train_batch_size=1	2*节点 & 8*Ascend 4*节点 & 8*Ascend
		cutoff_len=8192	lora sft	per_device_train_batch_size=1	2*节点 & 8*Ascend 8*节点 & 8*Ascend
	7B	cutoff_len=4096	lora/sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=8192	lora/sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
	0.5/1.5B	*cutoff_len=4096/8192*	lora/sft	per_device_train_batch_size=1	1*节点 & 1*Ascend
Qwen 1.5	0.5B/1.8B	cutoff_len=4096/8192	lora/sft	per_device_train_batch_size=1	1*节点 & 1*Ascend
	4B	*cutoff_len=4096/8192*	sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
	7B	cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
		cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend

支持模型	支持模型参数量	文本序列长度	训练类型	Zero并行	规格与节点数
	14B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
falcon 2	11B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 8*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
Yi	6B	cutoff_len=4096/8192	sft	per_device_train_batch_size=1	1*节点 & 4*Ascend
		cutoff_len=4096/8192	lora	per_device_train_batch_size=1	1*节点 & 1*Ascend
	34B	cutoff_len=4096	sft lora	per_device_train_batch_size=1	1*节点 & 8*Ascend 1*节点 & 2*Ascend
		cutoff_len=8192	sft lora	per_device_train_batch_size=1	2*节点 & 8*Ascend 1*节点 & 4*Ascend

3.2.5.4 各个模型训练前文件替换

在训练开始前，因模型权重文件可能与训练框架不匹配或有优化，因此需要针对模型的tokenizer文件进行修改或替换，不同模型的tokenizer文件修改内容如下。

falcon-11B 模型

在训练开始前，针对falcon-11B模型中的tokenizer文件，需要替换代码。替换文件{work_dir}/tokenizers/falcon-11B/config.json，具体步骤如下：

复制代码包目录下config.json至falcon-11B的tokenizer目录下，样例命令：

- 进入到代码目录下{work_dir}/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/如：

```
cd /home/ma-user/ws/llm_train/LLaMAFactory/ascendcloud_patch/models/falcon2/
```

- 复制config.json文件至加载的权重文件/tokenizer目录下，参考路径**上传代码和权重文件到工作环境**中的步骤3。

```
cp -f config.json {work_dir}/tokenizers/falcon-11B/
```

3.2.6 附录：指令微调训练常见问题

问题1：在训练过程中遇到NPU out of memory

解决方法：

步骤1 将yaml文件中的per_device_train_batch_size调小，重新训练如未解决则执行下一步。

步骤2 替换深度学习训练加速的工具或增加zero等级，可参考**各个模型深度学习训练加速框架的选择**，如原使用Accelerator可替换为Deepspeed-ZeRO-1，Deepspeed-ZeRO-1替换为Deepspeed-ZeRO-2以此类推，重新训练如未解决则执行下一步。

- - ZeRO-0 数据分布到不同的NPU
- - ZeRO-1 Optimizer States分布到不同的NPU
- - ZeRO-2 Optimizer States、Gradient分布到不同的NPU
- - ZeRO-3 Optimizer States、Gradient、Model Parameter分布到不同的NPU

步骤3 增加卡数重新训练，未解决找相关人员定位。

----结束

问题2：访问容器目录时提示Permission denied

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

问题3：训练过程报错：ImportError: This modeling file requires the following packages that were not found in your environment: **flash_attn**

根因：昇腾环境暂时不支持**flash_attn**接口

规避措施：修改dynamic_module_utils.py文件，将180-184行代码注释掉

```
vim /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/
dynamic_module_utils.py
```

```
177         except ImportError:
178             missing_packages.append(imp)
179
180 #     if len(missing_packages) > 0:
181 #         raise ImportError(
182 #             "This modeling file requires the following packages that were not found in your environment: "
183 #             f"{', '.join(missing_packages)}. Run `pip install {' '.join(missing_packages)}`"
184 #         )
185
186     return get_relative_imports(filename)
187
188
189 def get_class_in_module(class_name: str, module_path: Union[str, os.PathLike] ) -> typing.Type:
```

3.3 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907）

3.3.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 支持FP16和BF16数据类型推理。
- DevServer驱动版本要求23.0.6。

资源规格要求

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

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

说明

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

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-23 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-24**所示。

表 3-24 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如**表3-25**所示。

表 3-25 支持的模型列表和权重获取地址

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

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemmma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemmma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon2-11b	√	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
```

```
        ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
        ├── build.sh                # 推理构建脚本
        ├── vllm_install.patch      # 社区昇腾适配的补丁包
        ├── Dockerfile              # 推理构建镜像dockerfile
        ├── build_image.sh          # 推理构建镜像启动脚本
├──llm_tools          # 推理工具包
  ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
  ├──AutoAWQ            # W4A16量化工具
     ├──convert_awq_to_npu.py   # awq权重转换脚本
     ├──quantize.py   # 昇腾适配的量化转换脚本
     ├──build.sh      # 安装量化模块的脚本
  ├──llm_evaluation    # 推理评测代码包
     ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
     ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──install.sh           #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py            #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型（PyTorch）基于DevServer训练指导**。

3.3.2 部署推理服务

本章节介绍如何使用vLLM 0.5.0框架部署并启动推理服务。

前提条件

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

Step1 检查环境

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

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

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

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

2. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表3-23**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip到主机中，包获取路径请参见**表3-24**。
2. 将权重文件上传到DevServer机器中。权重文件的格式要求为Huggface格式。开源权重文件获取地址请参见**表3-25**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

   3.权重要求放在磁盘的指定目录，并做目录大小检查，参考命令如下：

   ```
   df -h
   ```

Step4 制作推理镜像

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保机器环境可以访问公网。

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

参数说明：

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

运行完后，会生成推理所需镜像。

Step5 启动容器镜像

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

```
docker run -itd \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
-v /etc/localtime:/etc/localtime  \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /etc/ascend_install.info:/etc/ascend_install.info \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
```

```
-v /var/log/npu/:/usr/slog \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v ${dir}:${container_work_dir} \
--net=host \
--name ${container_name} \
${image_id} \
/bin/bash
```

参数说明：

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

  说明

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

Step6 启动推理服务

1. 评估推理资源。运行如下命令，返回NPU设备信息可用的卡数。

   ```
   npu-smi info      # 启动推理服务之前检查卡是否被占用、端口是否被占用，是否有对应运行的进程
   ```

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

   驱动版本要求是23.0.6。如果不符合要求请参考**安装NPU设备和驱动**章节升级驱动。启动后容器默认端口是8080。
2. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

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

说明

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

图 3-15 查询结果

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

3. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型和Qwen2-57b模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

4. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**或**使用SmoothQuant量化**章节对模型做量化处理。
5. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务**

  在llm_inference/ascend_vllm/目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

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

- **方式二：通过vLLM服务API接口启动服务**

  在llm_inference/ascend_vllm/目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

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

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。若使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后在等待池等候处理。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM不同模型推理支持最小卡数和最大序列说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**2**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址，默认为127.0.0.1。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。

- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表3-26**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。如果启动服务未添加served-model-name参数，${container_model_path}的值请与model参数的值保持一致，如果使用了served-model-name参数，${container_model_path}请替换为实际使用的模型名称。

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

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
```

```
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

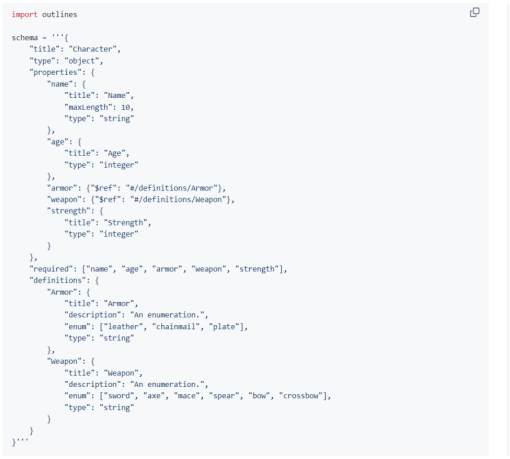
服务的API与vLLM官网相同，此处介绍关键参数。详细参数解释请参见官网**https://docs.vllm.ai/en/stable/dev/sampling_params.html**。

表 3-26 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

<table>
<tr><th>参数</th><th>是否必选</th><th>默认值</th><th>参数类型</th><th>描述</th></tr>
<tr><td>guided_json</td><td>否</td><td>None</td><td>Union[str, dict, BaseModel]</td><td>使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。
JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必需属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。
若希望使用JSON Schema，guided_json的写法可参考outlines: Structured Text Generation中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。

图 3-16 guided_json 样例
 

若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。
curl -X POST http://${docker_ip}:8080/v1/completions \
-H "Content-Type: application/json" \
-d '{
"model": "${container_model_path}",
"prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength",
"max_tokens": 200,
"temperature": 0,
"guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\": {\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength</td></tr>
</table>

参数	是否必选	默认值	参数类型	描述
				\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

3.3.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

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

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

代码目录如下:

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

目前性能测试还不支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step4 制作推理镜像**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，切换一个conda环境。
   ```
   cd benchmark_tools
   conda activate python-3.9.10
   ```
3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```

参数说明

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host ${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口8080。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。
- --epochs：测试轮数，默认取值为5
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。
- --served-model-name： 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

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

图 3-17 静态 benchmark 测试结果（示意图）

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

动态 benchmark

本章节介绍如何进行动态benchmark验证。

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

   方法一：使用公开数据集

   - ShareGPT下载地址: **https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json**

- Alpaca下载地址: **https://github.com/tatsu-lab/stanford_alpaca/blob/main/alpaca_data.json**

方法二：使用generate_dataset.py脚本生成数据集方法：

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

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

generate_dataset.py脚本执行参数说明如下：

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

2. 进入benchmark_tools目录下，切换一个conda环境。

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

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

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

- --backend：服务类型，如tgi，vllm，mindspore、openai。
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：推理服务端口。
- --dataset：数据集路径。
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是HuggingFace的权重路径，backend取值是openai时，tokenizer路径需要和推理服务启动时--model路径保持一致，比如--model /data/nfs/model/llama_7b， --tokenizer也需要为/data/nfs/model/llama_7b，两者要完全一致。
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。

- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
- --benchmark-csv：结果保存路径，如benchmark_serving.csv。
- --served-model-name： 选择性添加， 选择性添加，在接口中使用的模型名；如果没有配置，则默认为tokenizer。

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

图 3-18 动态 benchmark 测试结果（示意图）

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

3.3.4 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen、math_gen、gsm8k_gen、humaneval_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

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

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
├──vllm_ppl.py          #ppl精度测试脚本
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
```

3. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤4进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

4. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

5. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen --debug -w ${output_path}
```

output_path: 要保存的结果路径。

6. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。
   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算着n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预制变量

```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
```

执行脚本如下：

```
python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
```

output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集mmlu为例，配置如下：

表 3-27 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

7. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下：

   - --datasets：评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法。
   - --hf-type：HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择。
   - --hf-path：本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B。
   - --max-seq-len：模型的最大序列长度。
   - --max-out-len：模型的最大输出长度。
   - --hf-num-gpus：需要使用的卡数。
   - --batch-size：推理每次处理的输入数目。
   - -w：存放输出结果的目录。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1）认为NPU精度和GPU对齐。NPU和GPU的评分结果和社区的评分不能差太远(小于10）认为分数有效。

3.3.5 推理模型量化

3.3.5.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表3-25**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：per-group

Step1 模型量化

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

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1、在容器中使用ma-user用户运行以下命令下载并安装AutoAWQ源码。

```
bash build.sh
```

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

```
export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup，注意需指定到val.jsonl的上一级目录。

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

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

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

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

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

3.3.5.2 使用 SmoothQuant 量化

SmootQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmootQuant(W8A8)量化的模型列表请参见**表3-25**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

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

代码目录如下:

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

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。

- --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step6 启动推理服务**，启动推理服务时添加如下命令。

```
-q smoothquant 或者 --quantization smoothquant
--dtype=float16
```

3.3.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表3-25**。

Step1 使用 tensorRT 量化工具进行模型量化，必须在 GPU 环境

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

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

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

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

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

Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

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

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


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


注意：

1. 抽取完成后，可能提取不到model_type信息，需要手动将model_type修改为指定模型，如"llama"。
2. 当前社区vllm只支持float8的kv_cache量化，抽取脚本中dtype类型是"float8_e4m3fn"。dtype类型不影响int8的scale系数的抽取和加载。

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

3.3.6 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

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

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

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

表 3-28 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

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

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
38	falcon-11b	1	8	1	64

3.3.7 附录：大模型推理常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

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

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

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

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

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

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmootQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

3.4 主流开源大模型基于 Standard+OBS 适配 PyTorch NPU 训练指导（6.3.907）

3.4.1 场景介绍

方案概览

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

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

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

提示：本文档适用于仅使用OBS对象存储服务（Object Storage Service）作为存储的方案，OBS用于存储模型文件、训练数据、代码、日志等，提供了高可靠性的数据存储解决方案。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表3-29所示。

表 3-29 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 3-19 操作流程图

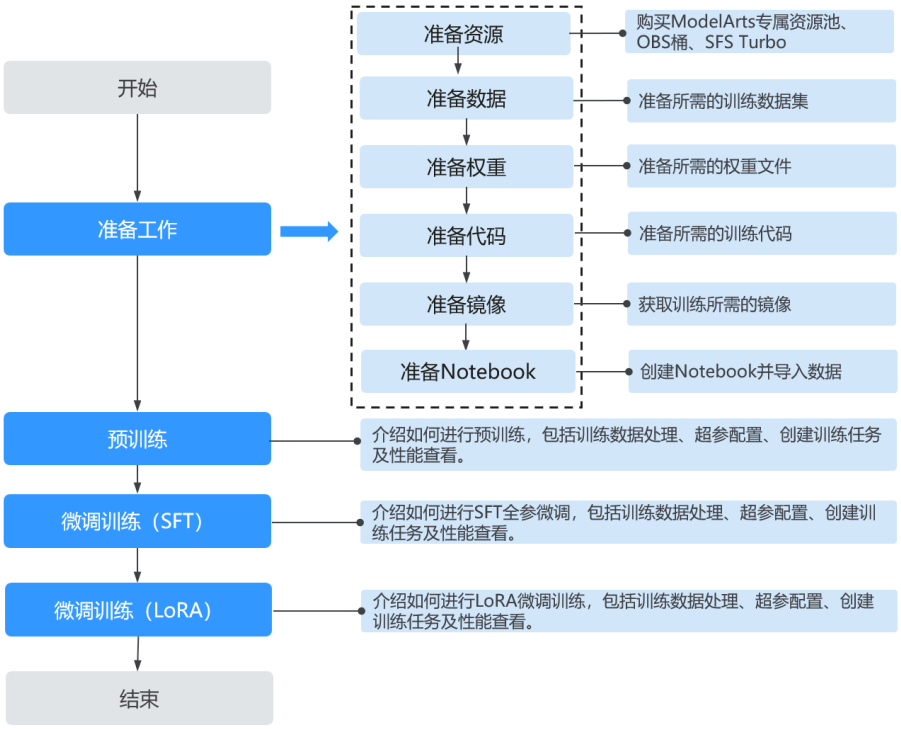


表 3-30 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

3.4.2 准备工作

3.4.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表3-36**。

硬盘空间：至少200GB。

昇腾资源规格：

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

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

3.4.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

- 预训练使用的Alpaca数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT和LoRA微调使用的Alpaca数据集下载：**https://huggingface.co/datasets/QingyiSi/Alpaca-CoT/blob/main/alpacaGPT4/alpaca_gpt4_data.json**，数据大小：43.6 MB。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3.4.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-29**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

3.4.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表3-31**所示。

表 3-31 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.907代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|——llm_train                # 模型训练代码包
        |——AscendSpeed        # 基于AscendSpeed的训练代码
              |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |——scripts/             # 训练需要的启动脚本
                   |——llama2          # llama2系列模型执行脚本的文件夹
                   |——llama3          # llama3系列模型执行脚本的文件夹
                   |——qwen            # Qwen系列模型执行脚本的文件夹
                   |——qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   |—— ...
                   |—— dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   |—— install.sh       # 环境部署脚本
              |———src/                # 启动命令行封装脚本，在install.sh里面自动构建
|——llm_inference              # 推理代码包
|——llm_tools                  # 推理工具
```

代码上传至 OBS

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|——llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |—— AscendSpeed               # 代码目录
          |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |——scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|—— training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |—— train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
```

```
    |── alpaca_gpt4_data.json                    #微调数据文件
|── model                                       #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
```

3.4.2.5 准备镜像

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-32 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像（二选一）**、**ECS中构建新镜像（二选一）**的方式（**二选一**）来部署训练环境。方案的区别如下：

- **使用基础镜像（二选一）**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像（二选一）**：在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。
  - 若用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

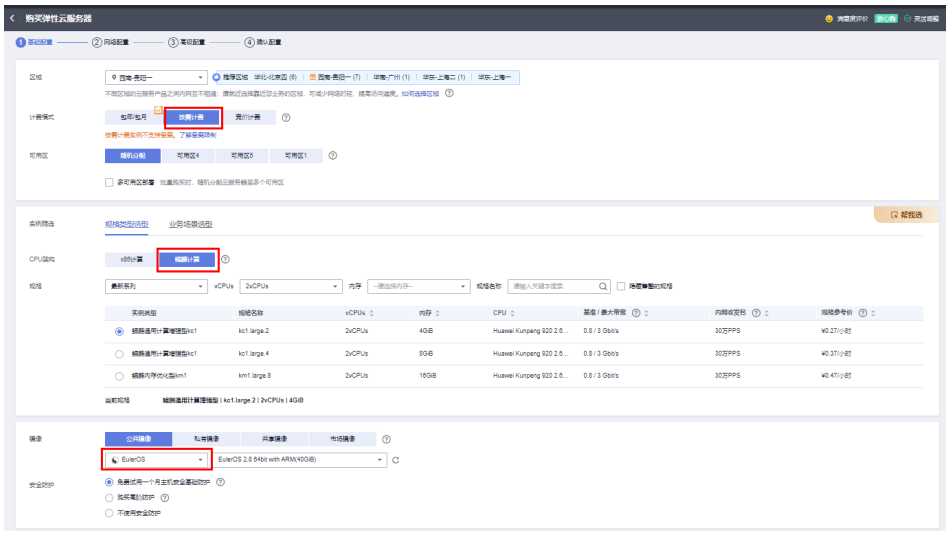
ECS 获取和上传基础镜像

步骤1 创建ECS。

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

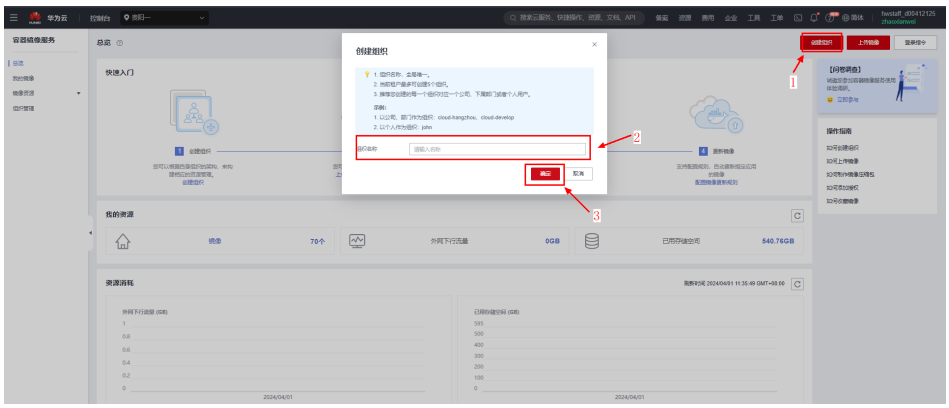
图 3-20 购买 ECS



步骤2 创建镜像组织。

在SWR服务页面创建镜像组织。

图 3-21 创建镜像组织



步骤3 安装Docker。

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

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

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

步骤4 获取训练镜像。

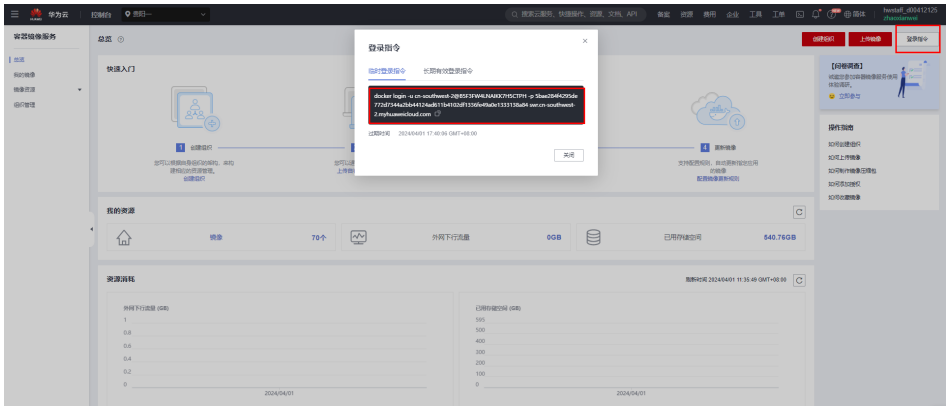
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表3-32**。

```
docker pull {image_url}
```


步骤5 在ECS中Docker登录。

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-22 复制登录指令



步骤6 修改并上传镜像。

1. 登录指令输入之后，使用下列示例命令：
   ```
   docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   参数说明：
   - <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
   - <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

   docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606
2. 上传镜像至镜像仓库。
   ```
   docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   示例：

   docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

----结束

使用基础镜像（二选一）

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

由于基础镜像内需要安装固定版本依赖包，若直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图3-23**中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。命令如下：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而 install.sh 则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 3-23 训练作业启动命令

ECS 中构建新镜像（二选一）

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

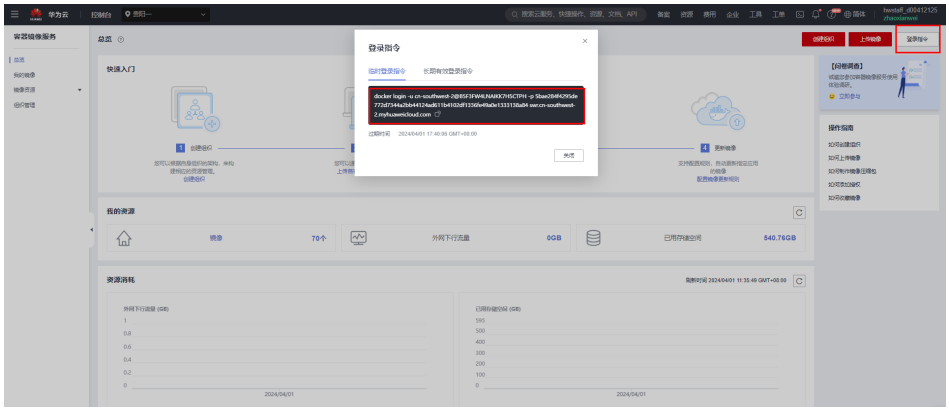
步骤1 获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表3-31**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.907-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   若无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

步骤2 在ECS中Docker登录。

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-24 复制登录指令



步骤3 修改并上传镜像。

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：
   ```
   docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   参数说明：

   ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。

   <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

   <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group

   <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

   示例：

   docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606
2. 上传镜像至镜像仓库。
   ```
   docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
   ```
   示例：

   docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

----结束

3.4.2.6 准备 Notebook（可选）

本步骤为可选操作。ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。

本案例中，若用户需要自定义开发，可通过Notebook环境进行数据预处理、权重转换等操作。并且Notebook环境具有一定的存储空间，可与OBS中的数据相互传递。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 3-25 Notebook 中选择自定义镜像与规格

云硬盘EVS是Notebook开发环境内存的存储硬盘，作为持久化存储挂载在/home/ma-user/work目录下，该目录下的内容在实例停止后会被保留。可以自定义磁盘空间，若需要存储数据集、模型等大型文件，建议申请规格300GB+。存储支持在线按需扩容。

图 3-26 自定义存储配置

使用 Notebook 将 OBS 数据导入云硬盘 EVS

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至云硬盘EVS中，并可通过Notebook随时访问并编辑云硬盘EVS中的数据

3.4.3 预训练

前提条件

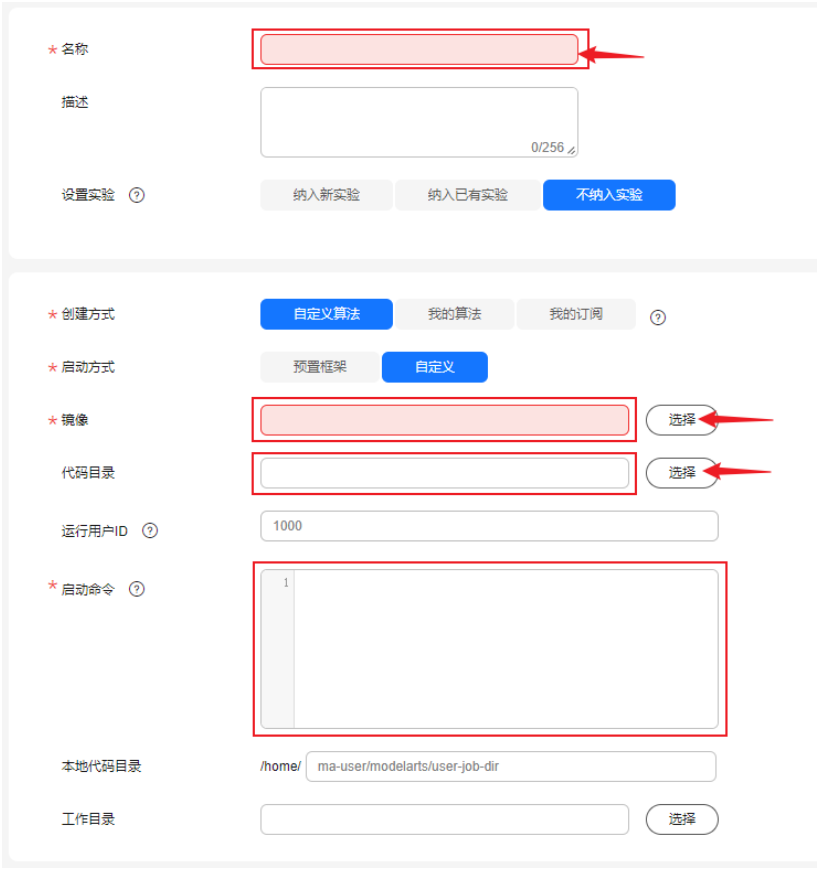
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 3-27 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

点击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。

2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。
   - **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
   - **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。
3. 分别点击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

点击“增加环境变量”，在增加的环境变量填写框中，按照表3-33表格中的配置进行填写。

表 3-33 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralPretrainHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	5120	用户指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	1	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-36**进行配置。

图 3-28 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

3.4.4 SFT 全参微调训练

前提条件

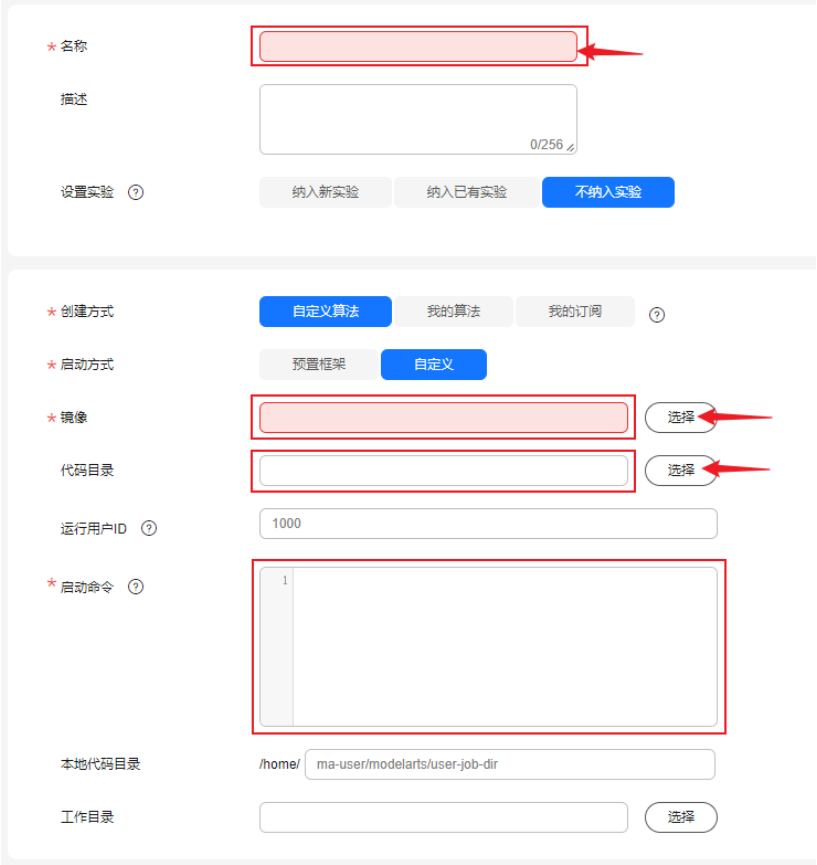
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 3-29 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

点击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别点击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

点击“增加环境变量”，在增加的环境变量填写框中，按照表3-33表格中的配置进行填写。

图 3-30 环境变量

表 3-34 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	sft	表示训练类型。可选择值：[pretrain, sft, lora]。

环境变量	示例值	参数说明
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	5120	用户指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	1	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-36**进行配置。

图 3-31 选择资源池规格

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

3.4.5 LoRA 微调训练

前提条件

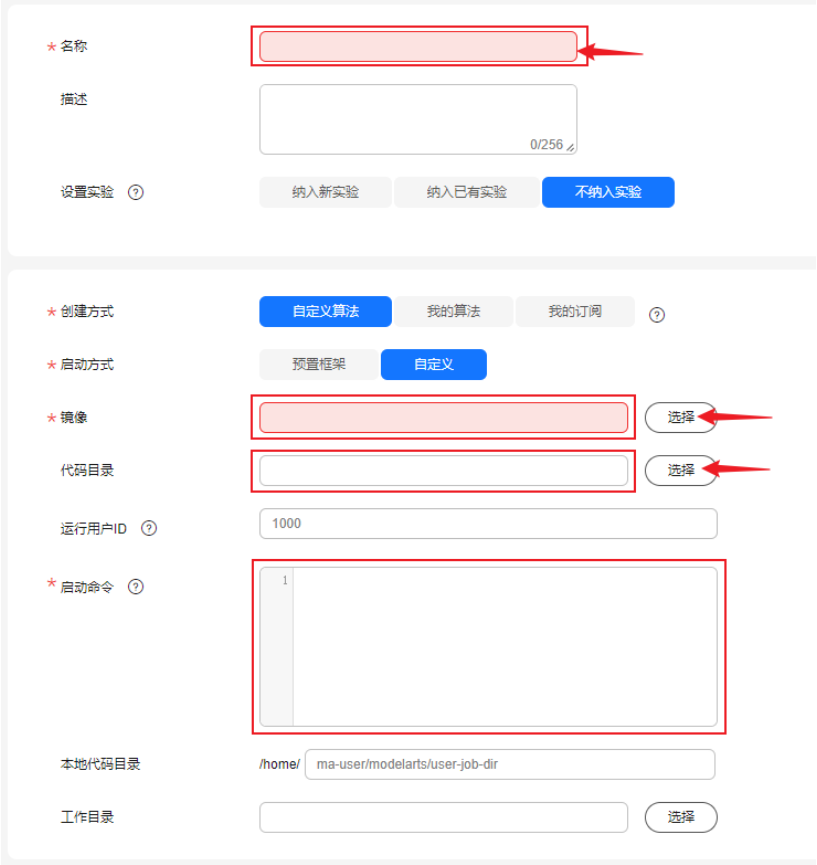
已上传训练代码、训练权重文件和数据集到OBS中，具体参考**代码上传至OBS**。

Step1 创建训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及选择上传的镜像。

代码目录选择：OBS桶路径下的 llm_train/AscendSpeed 代码目录。

图 3-32 创建训练作业



若镜像使用**使用基础镜像（二选一）**中的基础镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/obs_pipeline.sh
```

若镜像使用**ECS中构建新镜像（二选一）**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/modelarts/user-job-dir/AscendSpeed;
sh ./scripts/obs_pipeline.sh
```

Step2 配置数据输入和输出

点击“增加训练输入”和“增加训练输出”，用于配置训练作业开始时需要输入数据的路径和训练结束后输出数据的路径。

1. 在“输入”的输入框内设置变量：ORIGINAL_TRAIN_DATA_PATH、ORIGINAL_HF_WEIGHT。
   - **ORIGINAL_TRAIN_DATA_PATH**：训练时指定的输入数据集路径。
   - **ORIGINAL_HF_WEIGHT**：加载tokenizer与Hugging Face权重时，对应的存放地址。
2. 在“输出”的输入框内设置变量：OUTPUT_SAVE_DIR、HF_SAVE_DIR。

- **OUTPUT_SAVE_DIR**：训练完成后指定的输出模型路径。
- **HF_SAVE_DIR**：训练完成的权重文件自动转换为Hugging Face格式权重输出的路径（确保添加CONVERT_MG2HF环境变量并设置为True）。

3. 分别点击“输入”和“输出”的数据存储位置，如图所示，选择OBS桶中指定的目录。ORIGINAL_TRAIN_DATA_PATH中则直接选中数据集文件。
4. “输入”和“输出”中的获取方式全部选择为：环境变量。
5. “输出”中的预下载至本地目标选择：下载，此时输出路径中的数据则会下载至OBS中。

Step3 配置环境变量

点击“增加环境变量”，在增加的环境变量填写框中，按照**表3-33**表格中的配置进行填写。

表 3-35 需要填写的环境变量

环境变量	示例值	参数说明
MOUNT	OBS	**默认必须填写。**表示代码根据OBS存储方式运行。
MODEL_NAME	llama2-13b	输入选择训练的模型名称。
RUN_TYPE	lora	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	GeneralInstructionHandler	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。 • MOSSMultiTurnHandler：使用微调的moss数据集。

环境变量	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	5120	用户指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	1	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	表示训练step迭代次数，根据实际需要修改。
SAVE_INTERVAL	10	表示训练间隔多少step，则会保存一次权重文件。
SEED	1234	随机种子数。每次数据采样时，保持一致。
CONVERT_MG2HF	True	表示训练完成的权重文件会自动转换为Hugging Face格式权重。若不需要自动转换，则删除该环境变量。

对于ChatGLMv3-6B、GLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step4 其他配置

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-36**进行配置。

图 3-33 选择资源池规格

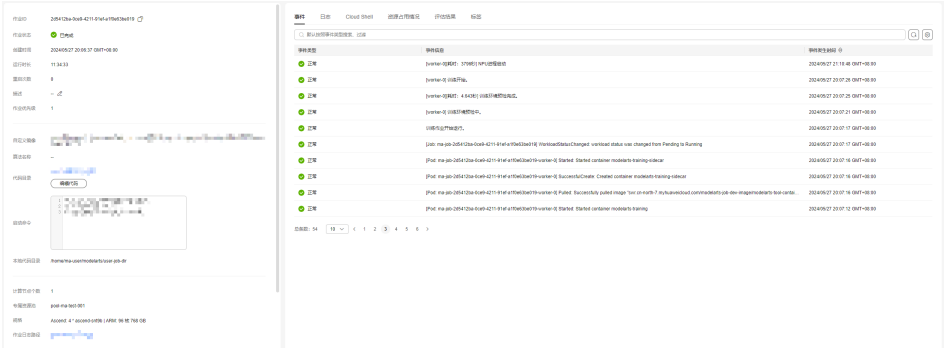
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型训练**。

3.4.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

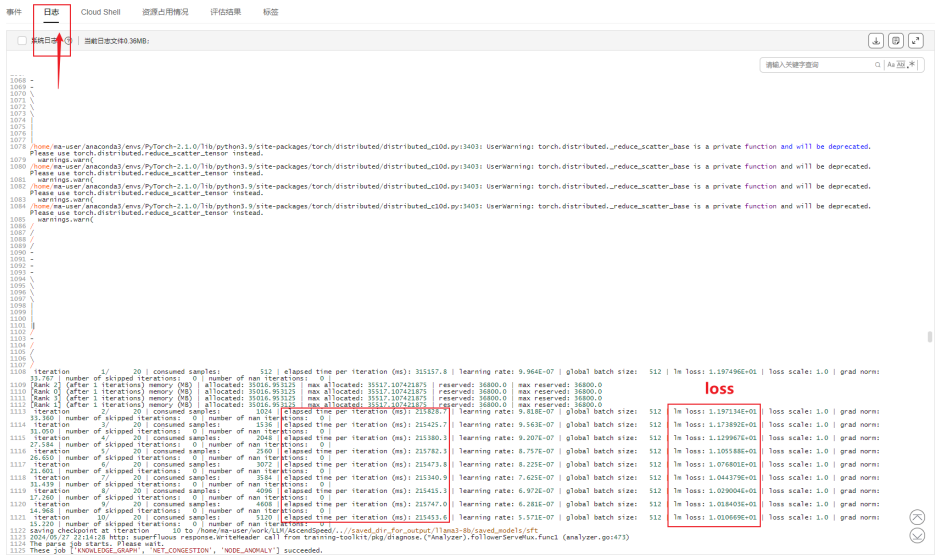
图 3-34 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size×seq_length/(总卡数×elapsed time per iteration)×1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 3-35 查看日志和性能



3.4.7 训练脚本说明

3.4.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过统一的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。用户可通过**Notebook**中创建.ipynb文件，并编辑以下代码可实现Notebook环境中的数据与OBS中的数据进行相互传递。

```
import moxing as mox
# OBS存放数据路径
obs_data_dir= "obs://<bucket_name>/data"
# NoteBook存放数据路径
local_data_dir= "/home/ma-user/work/data"
# OBS数据上传至Notebook
mox.file.copy_parallel(obs_data_dir, local_data_dir)
# Notebook数据上传至OBS
mox.file.copy_parallel(local_data_dir, obs_data_dir)
```

不同模型推荐的训练参数和计算规格要求如表3-36所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-36 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

3.4.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**obs_pipeline.sh** 训练脚本后，脚本自动执行数据集预处理，并检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）

- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-37 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。

环境变量	示例	参数说明
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.4.7.3 训练的权重转换说明

以 **llama2-13b** 举例，使用训练作业运行 **obs_pipeline.sh** 脚本后，脚本自动执行权重转换，并检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

Megatron 转 HuggingFace 参数说明

若用户需要自动转换，则在训练作业中，添加变量**CONVERT_MG2HF**并赋值**True**。若用户后续不需要自动转换，则在环境变量中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

注意：权重转换完成后，需要将转换后的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-38 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.4.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图3-36所示。

图 3-36 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-37所示。

图 3-37 修改 ChatGLMv3-6B tokenizer 文件

```
270         # Load from model_defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-38所示。

图 3-38 修改 ChatGLMv3-6B tokenizer 文件

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

294行要添加注释，修改后如图3-39所示。

图 3-39 修改 ChatGLMv4-9B tokenizer 文件

```
293         # Load from model defaults
294         #assert self.padding_side == "left"
```

314至323行要修改，修改后如图3-40所示。

图 3-40 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-41所示。

图 3-41 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41
42
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.4.8 常见错误原因和解决方法

3.4.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照**表3-36**进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

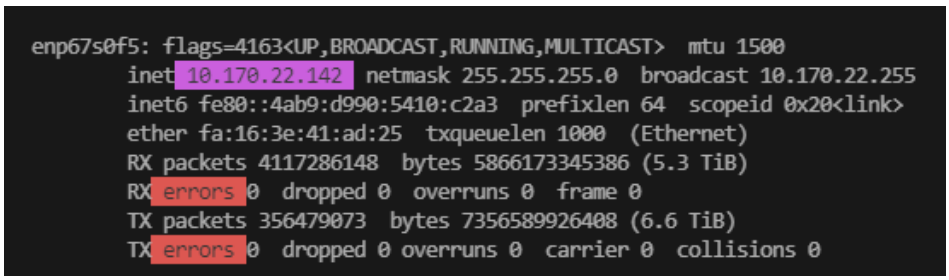
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

3.4.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 3-42 网卡名称错误



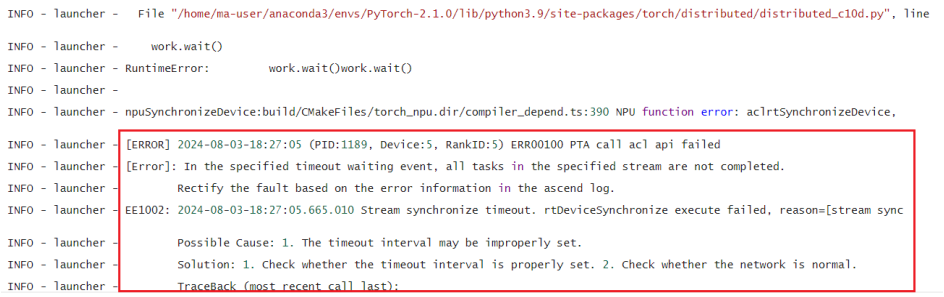

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

3.4.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 3-43 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

3.5 主流开源大模型基于 Standard+OBS+SFS 适配 PyTorch NPU 训练指导（6.3.907）

3.5.1 场景介绍

方案概览

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

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

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

提示：本文档适用于OBS+SFS Turbo的数据存储方案，不适用于仅OBS存储方案。通过OBS对象存储服务（Object Storage Service）与SFS Turbo文件系统联动，可以实现灵活数据管理、高性能读取等。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如**表3-39**所示。

表 3-39 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**
17	Qwen2	qwen2-0.5b	**https://huggingface.co/Qwen/Qwen2-0.5B-Instruct**

序号	支持模型	支持模型参数量	权重文件获取地址
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 3-44 操作流程图

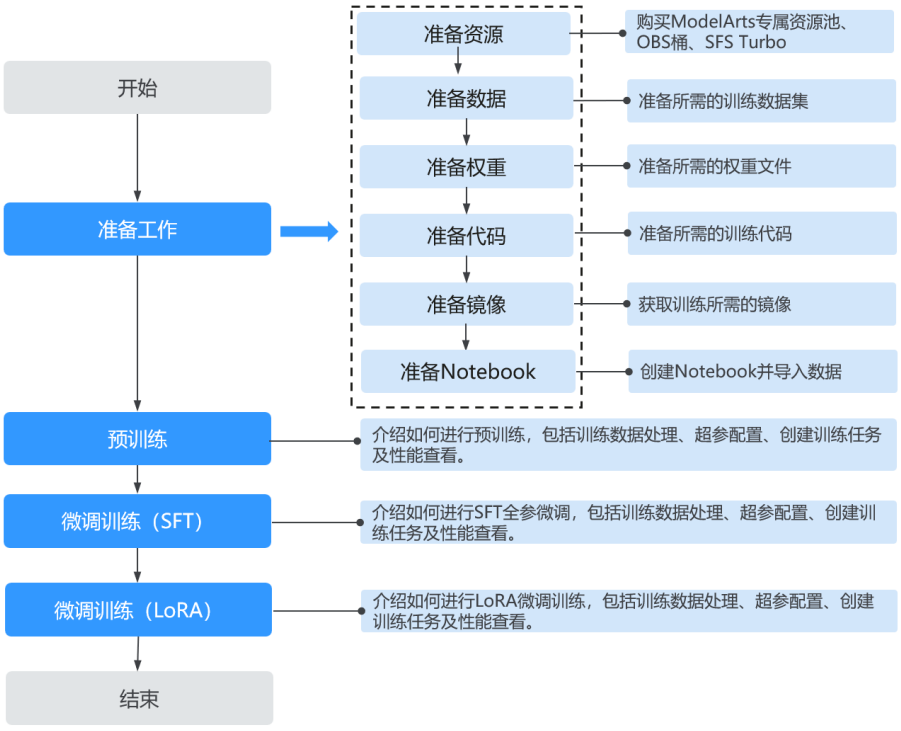


表 3-40 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。

阶段	任务	说明
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

3.5.2 准备工作

3.5.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表3-47**。

硬盘空间：至少200GB。

昇腾资源规格：

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

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

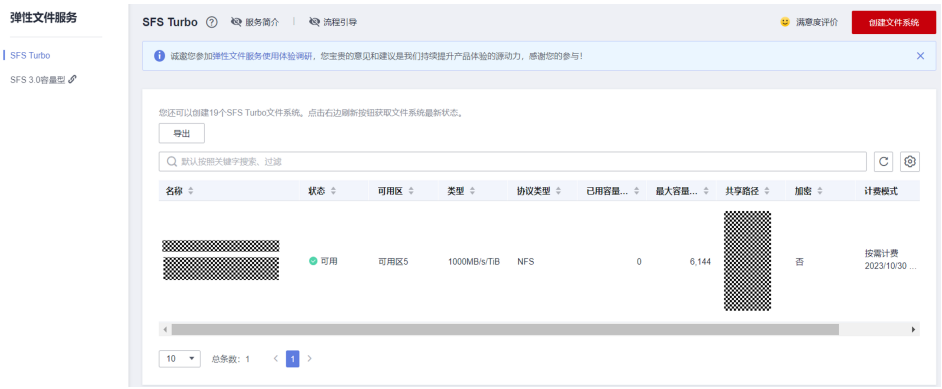
SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统前提条件：

1. 创建SFS Turbo文件系统前，确认已有可用的VPC。
2. 需要由IAM用户设置SFS Turbo FullAccess权限，用于授权ModelArts云服务使用SFS Turbo。

详细操作指导请参考**创建SFS Turbo文件系统**。

图 3-45 创建 SFS Turbo



其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 3-46 SFS 类型和容量选择

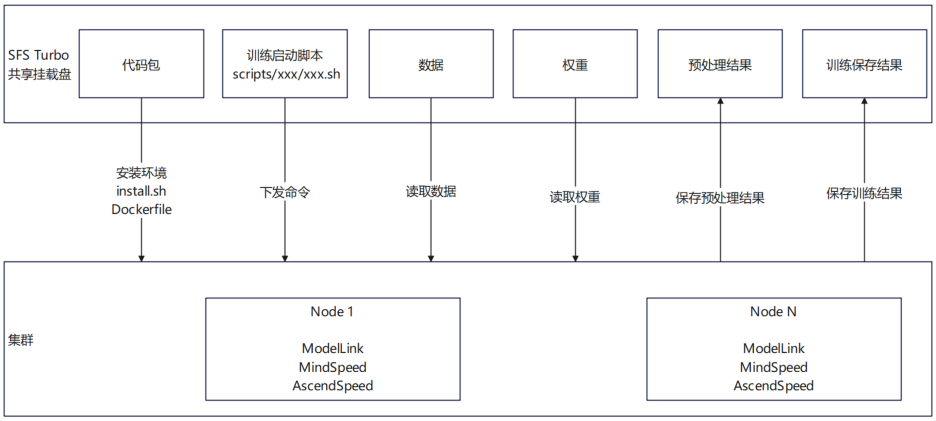
ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。

若ModelArts网络关联SFS Turbo失败，则需要授权ModelArts云服务使用SFS Turbo，具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 3-47 ModelArts 网络关联 SFS Turbo

SFS Turbo 模式下执行流程



SFS Turbo作为完全托管的共享文件存储系统，在本方案中作为主要的存储介质应用于训练作业。因此，后续需要准备的**原始数据集**、**原始Hugging Face权重文件**以及**训练代码**都需要上传至SFS Turbo中。而基于SFS Turbo所执行的训练流程如下：

1. 将SFS Turbo挂载至ECS服务器后，可直接访问SFS Turbo。通过SSH连接ECS将代码包上传至SFS Turbo中。
2. 在**表3-42**获取基础镜像，随后通过**镜像方案说明**中的步骤执行代码包中llm_train/AscendSpeed/Dockerfile文件，构建新的镜像，并上传至SWR中。
3. 新构建的镜像中，包含有ModelLink、MindSpeed、Megatron-LM等代码，在集群中启动容器即可通过/home/ma-user/AscendSpeed路径访问。
4. 在ModelArts中创建训练作业如：**预训练**，执行代码包中例如：scripts/llama2/0_pl_pretrain_13b.sh 的脚本，开始训练。
5. 在训练中，程序会自动执行对数据集预处理、权重转换、执行训练等操作，具体可通过**训练启动脚本说明和参数配置**、**训练的数据集预处理说明**、**训练的权重转换说明**了解其中的操作。
6. 训练完成后在SFS Turbo中保存训练的模型结果。（多机情况下，只有在rank_0节点进行数据预处理，权重转换等工作，所以原始数据集和原始权重，包括保存结果路径，都应该在共享目录下）

3.5.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

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

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3.5.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-39**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
```

```
├── tokenizer.model
└── USE_POLICY.md
```

3.5.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表3-41**所示。

表 3-41 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.906代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|──llm_train                # 模型训练代码包
        |──AscendSpeed        # 基于AscendSpeed的训练代码
              |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |──scripts/             # 训练需要的启动脚本
                  |──llama2           # llama2系列模型执行脚本的文件夹
                  |──llama3           # llama3系列模型执行脚本的文件夹
                  |──qwen             # Qwen系列模型执行脚本的文件夹
                  |──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                  |── ...
                  |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                  |── install.sh       # 环境部署脚本
              |──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|──llm_inference              # 推理代码包
|──llm_tools                  # 推理工具
```

代码上传至 OBS

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                  # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed             # 代码目录
          |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |──scripts/             # 训练需要的启动脚本
```

```
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
          |── ${model_name}            # 模型名称
                |── data             # 预处理后数据
             |── pretrain   # 预训练加载的数据
             |── finetune   # 微调加载的数据
       |──converted_weights  # HuggingFace格式转换magatron格式后权重文件
    |── saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
          |── ${model_name}            # 模型名称
                |── logs             # 训练过程中日志（loss、吞吐性能）
                   |─── saved_models
                |── lora             # lora微调输出权重
                |── sft              # 增量训练输出权重
                |── pretrain         # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会
提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                          #微调数据文件
|── model                                          #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作
步骤中会提示
    |── llama2-13b-hf
```

3.5.2.5 准备镜像

3.5.2.5.1 镜像方案说明

准备大模型训练适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

基础镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-42 基础容器镜像地址

镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

基础镜像的使用

用户通过**ECS获取和上传基础镜像**步骤拉取基础镜像并上传至SWR中。随后可通过**使用基础镜像**、**ECS中构建新镜像**、**Notebook中构建新镜像**的方式（**三选一**）来部署训练环境。方案的区别如下：

- **直接使用基础镜像方案**：用户可在训练作业中直接选择基础镜像作为运行环境。但基础镜像中pip依赖包缺少或版本不匹配，因此每次创建训练作业时，训练作业的启动命令中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。
- **ECS中构建新镜像方案：**在ECS中，通过运行Dockerfile文件会在基础镜像上创建新的镜像。新镜像命名可自定义。Dockerfile会下载Megatron-LM、MindSpeed、ModelLink源码，并将以上源码打包至镜像环境中。

  - 若用户希望修改源码，则需要使用新镜像创建容器，在容器内的/home/ma-user工作目录中访问并编辑以上源码文件。编辑完成后重新构建新镜像。

- **Notebook中构建新镜像方案：**首先需要ECS将基础镜像上传至SWR中。随后在Notebook环境中，通过运行scripts/install.sh文件会安装必要的依赖包以及下载Megatron-LM、MindSpeed、ModelLink源码。若Notebook环境挂载了SFS Turbo，则源码文件会下载至SFS Turbo中。最后选择Notebook中“保存镜像”，则可以得到新的镜像环境。

  若用户希望修改源码，则需要在Notebook环境中直接访问并编辑源码文件。

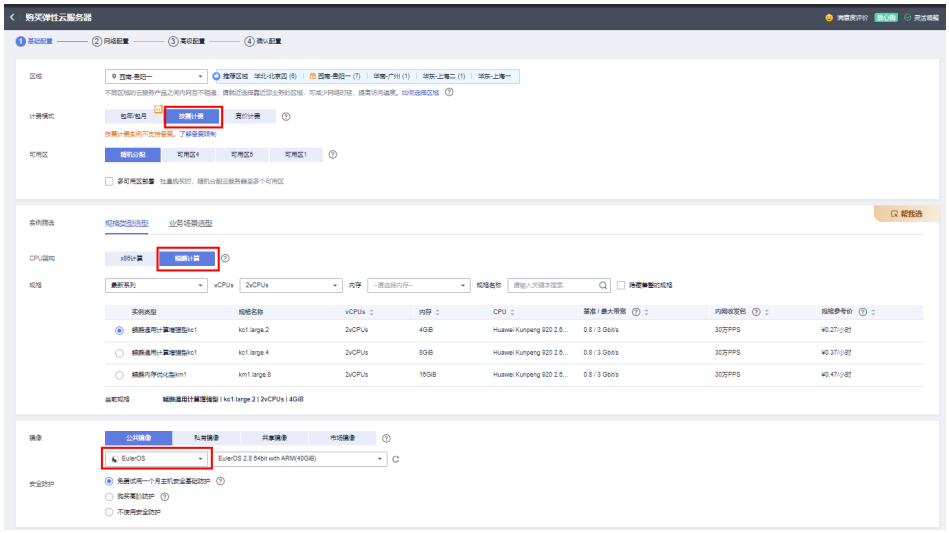
3.5.2.5.2 ECS 获取和上传基础镜像

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

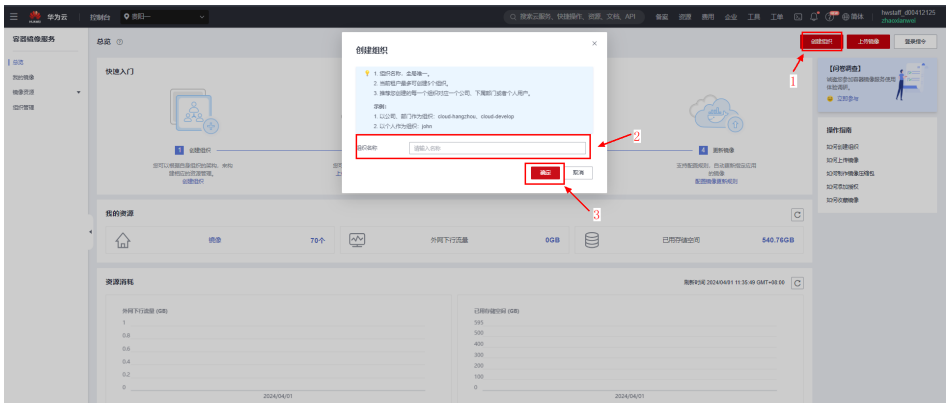
图 3-48 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 3-49 创建镜像组织



Step3 安装 Docker

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

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

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

Step4 获取训练镜像

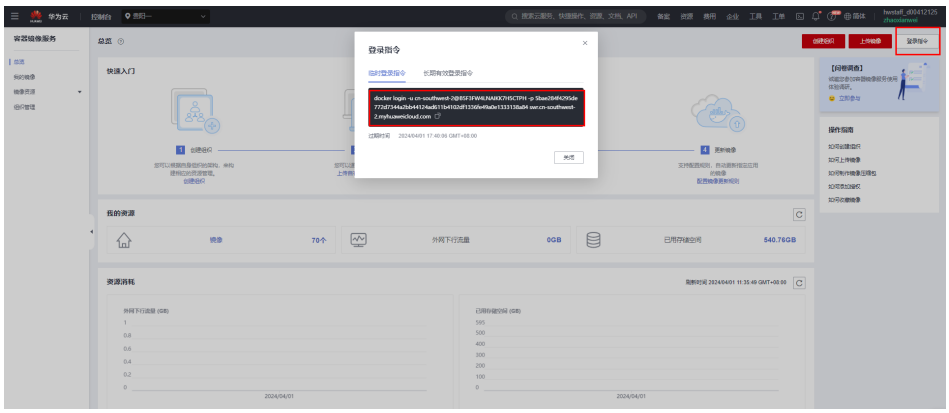
请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表3-42**。

```
docker pull {image_url}
```

Step5 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-50 复制登录指令



Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

3.5.2.5.3 使用基础镜像

通过**ECS获取和上传基础镜像**将镜像上传至SWR服务后，可创建训练作业，在“选择镜像”中选择SWR中基础镜像。

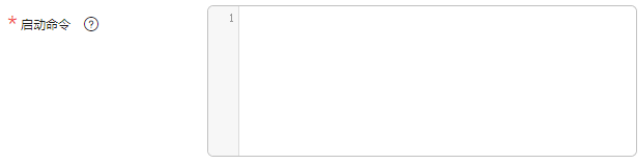
由于基础镜像内需要安装固定版本依赖包，若直接使用基础镜像进行训练，每次创建训练作业时，训练作业的**图3-51**中都需要执行 install.sh 文件，来安装依赖以及下载完整代码。

以创建llama2-13b预训练作业为例，执行脚本0_pl_pretrain_13b.sh时，命令如下：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业后，会在节点机器中使用基础镜像创建docker容器，并在容器内进行分布式训练。而 install.sh 则会在容器内安装依赖以及下载完整的代码。当训练作业结束后，对应的容器也会同步销毁。

图 3-51 训练作业启动命令



3.5.2.5.4 ECS 中构建新镜像

通过**ECS获取和上传基础镜像**获取基础镜像后，可通过ECS运行Dockerfile文件，在镜像的基础上构建新镜像。

Step1 构建新 ModelArts Standard 训练镜像

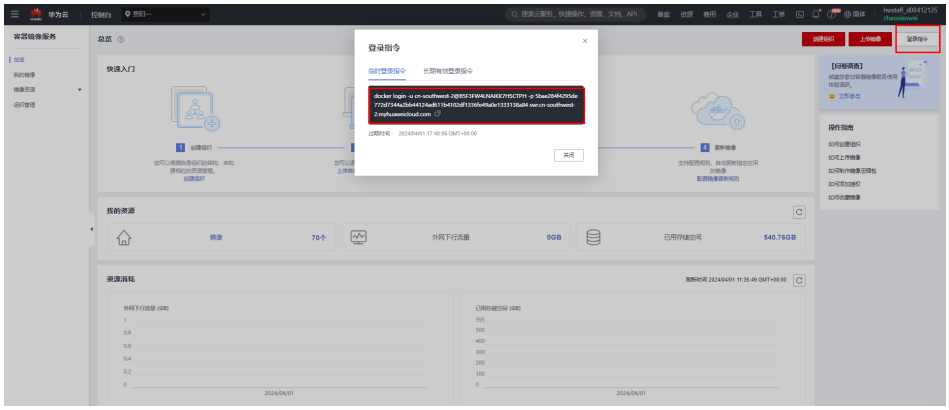
获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表3-41**。

1. 解压AscendCloud压缩包及该目录下的训练代码AscendCloud-LLM-6.3.907-xxx.zip，并直接进入到llm_train/AscendSpeed文件夹下面
   ```
   unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_train/AscendSpeed
   ```
2. 编辑llm_train/AscendSpeed中的Dockerfile文件，修改git命令，填写自己的git账户信息。
   ```
   git config --global user.email "you@example.com" &&  \
   git config --global user.name "Your Name" &&  \
   ```
3. 执行以下命令制作训练镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网
   ```
   docker build -t <镜像名称>:<版本名称>  .
   ```
   若无法访问公网，则可以配置代理，增加`--build-arg`参数指定代理地址，可访问公网。
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host -t <镜像名称>:<版本名称>  .
   ```
   - <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606
   - 记住使用Dockerfile创建的新镜像名称， 后续使用 ${dockerfile_image_name} 进行表示。

Step2 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-52 复制登录指令



Step3 修改并上传镜像

1. 在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR：

```
docker tag ${dockerfile_image_name} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- ${dockerfile_image_name}：在step5中，使用Dockerfile创建的新镜像名称。
- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag {image_url} swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

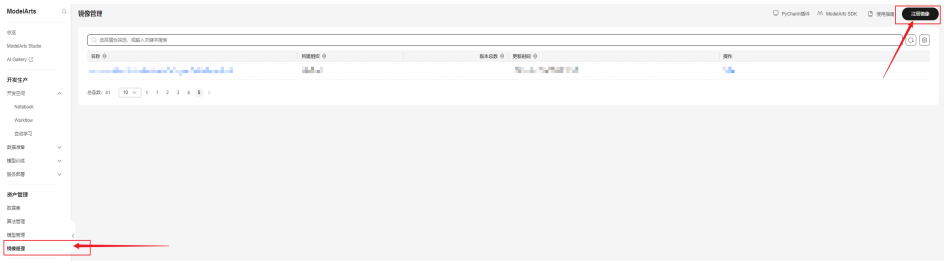
3.5.2.5.5 Notebook 中构建新镜像

ModelArts 中注册镜像

通过**ECS获取和上传基础镜像**将基础镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。

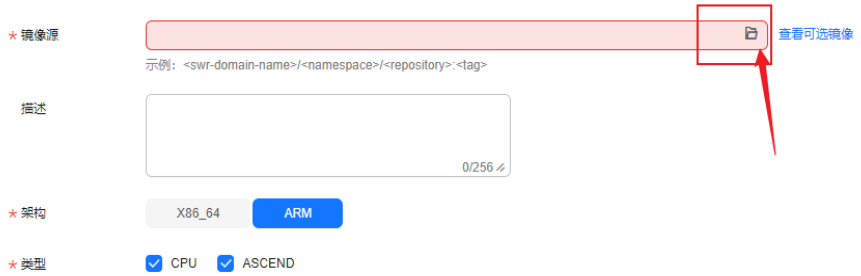
访问ModelArts，在镜像管理中选择注册镜像，如图所示：

图 3-53 注册镜像



选择已上传的镜像源，架构选择ARM，类型勾选CPU和ASCEDN，完成镜像注册。

图 3-54 选择已上传的镜像源



Notebook 介绍

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

Step1 创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 3-55 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例，子目录挂载可选择默认不填写。

如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 3-56 Notebook 中选择弹性文件服务

Step2 使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
```

```
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Step3 Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包。

通过运行install.sh脚本，会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
           |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
           |——scripts/             # 训练需要的启动脚本
           |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/           # MindSpeed昇腾大模型加速库
       |——ModelLink/           # ModelLink端到端的大语言模型方案
           |——megatron/        # 注意：该文件夹从Megatron-LM中复制得到
           |——...
```

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

图 3-57 安装依赖包

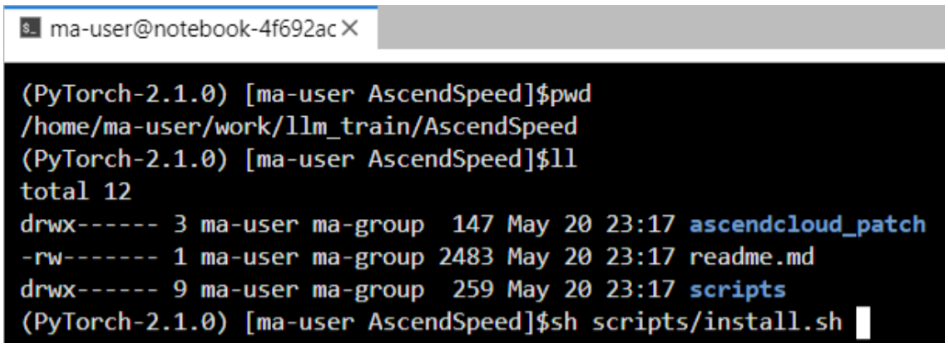


图 3-58 保存镜像

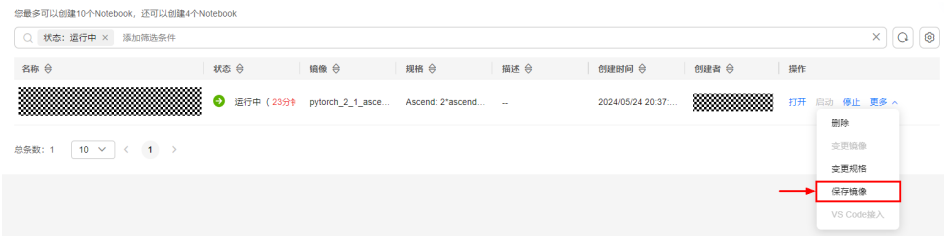


图 3-59 填写保存镜像相关参数

3.5.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如表3-43所示。其他超参均有默认值，可以参考表3-46按照实际需求修改。

表 3-43 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。

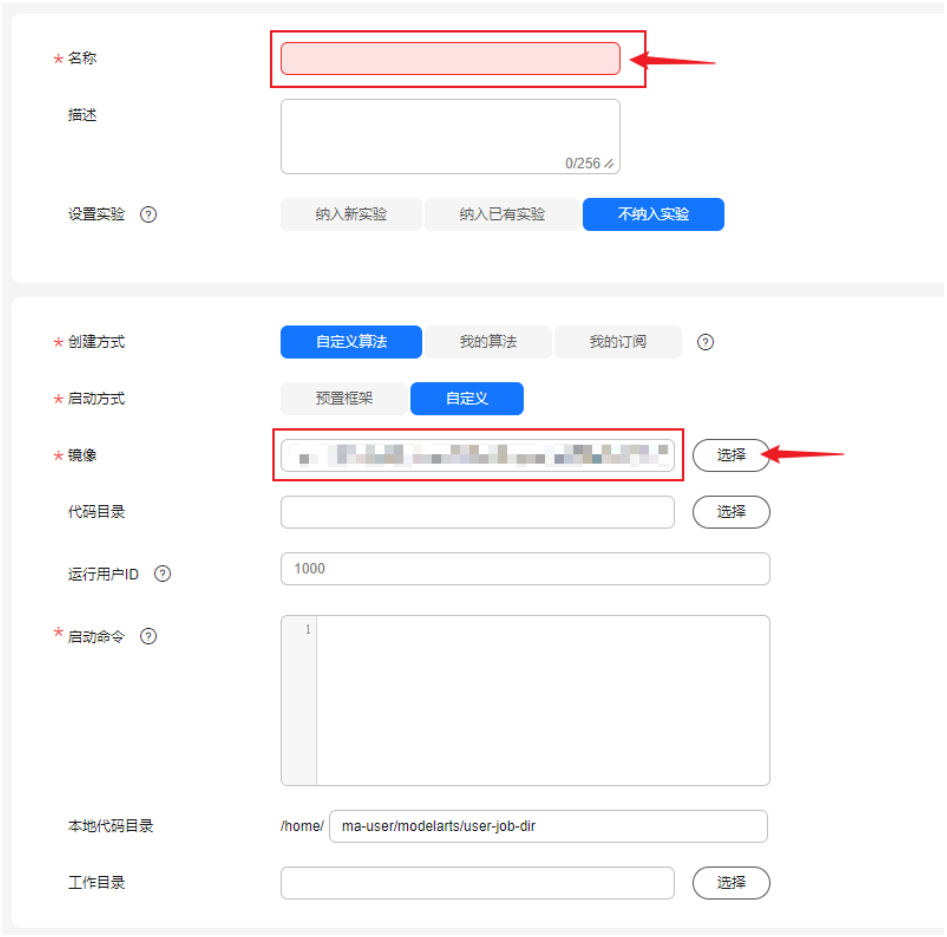
参数	示例值	参数说明
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-60 选择镜像



若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

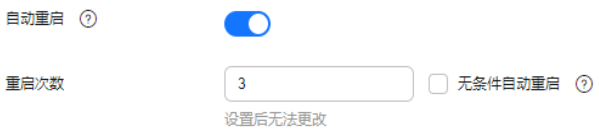
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 3-61 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

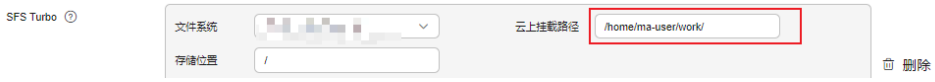
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-47**进行配置。

图 3-62 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 3-63 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.5.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如表3-44所示。其他超参均有默认值，可以参考表3-46按照实际需求修改。

表 3-44 训练超参配置说明

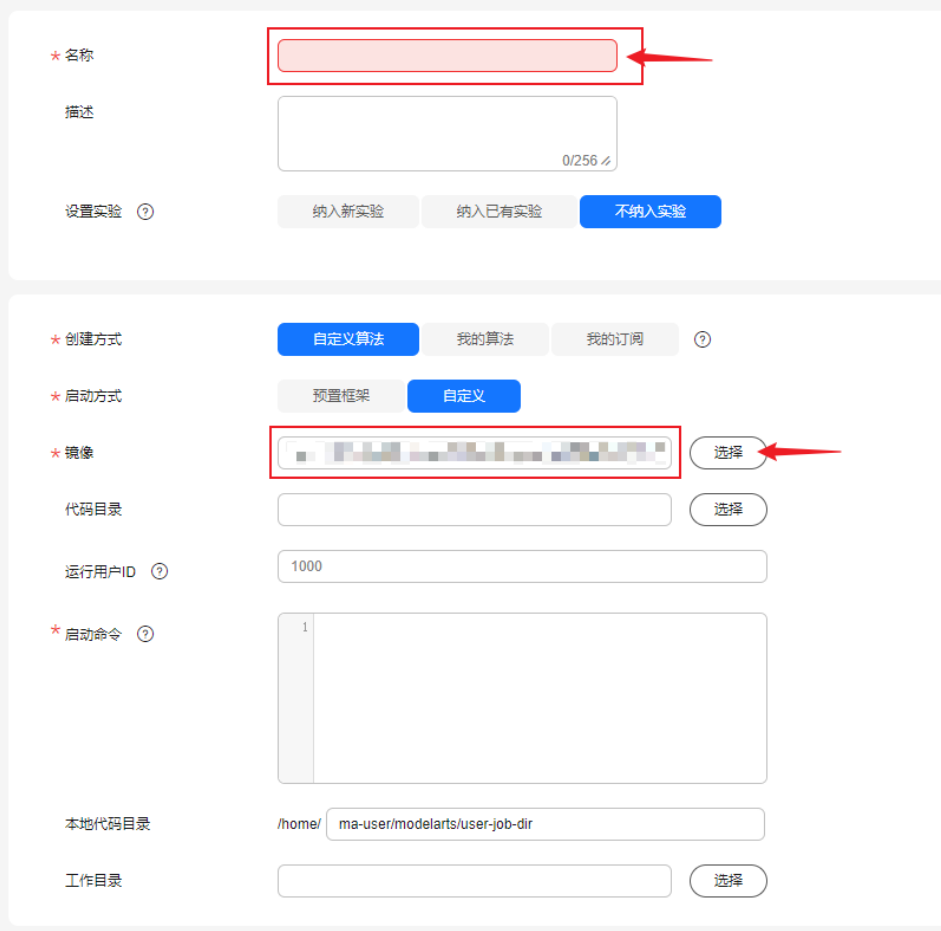
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-64 选择镜像



若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

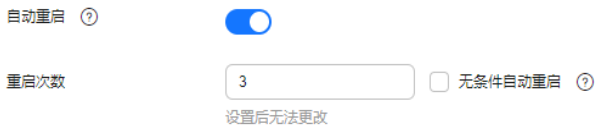
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 3-65 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

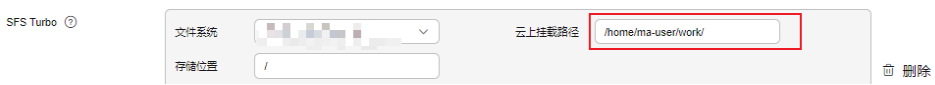
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-47**进行配置。

图 3-66 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 3-67 选择 SFS Turbo



作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.5.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-45**所示。其他超参均有默认值，可以参考**表3-46**按照实际需求修改。

表 3-45 训练超参配置说明

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载Hugging Face权重（可与tokenizer相同文件夹）时，对应的存放地址。请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	该参数为tokenizer文件的存放地址。默认与ORIGINAL_HF_WEIGHT路径相同。若用户需要将Hugging Face权重与tokenizer文件分开存放时，则需要修改参数。
INPUT_PROCESSED_DIR	/home/ma-user/work/AscendSpeed/processed_for_input/llama2-13b	该路径下保存“数据转换”和“权重转换”的结果。示例中，默认生成在“processed_for_input”文件夹下。若用户需要修改，可添加并自定义该变量。
OUTPUT_SAVE_DIR	/home/ma-user/work/AscendSpeed/saved_dir_for_output/	该路径下统一保存生成的 CKPT、PLOG、LOG 文件。示例中，默认统一保存在“saved_dir_for_output”文件夹下。若用户需要修改，可添加并自定义该变量。
CKPT_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b	保存训练生成的模型 CKPT 文件。示例中，默认保存在“saved_dir_for_output/saved_models”文件夹下。若用户需要修改，可添加并自定义该变量。
LOG_SAVE_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/saved_models/llama2-13b/log	保存训练过程记录的日志 LOG 文件。示例中，默认保存在“saved_models/llama2-13b/log”文件夹下。若用户需要修改，可添加并自定义该变量。

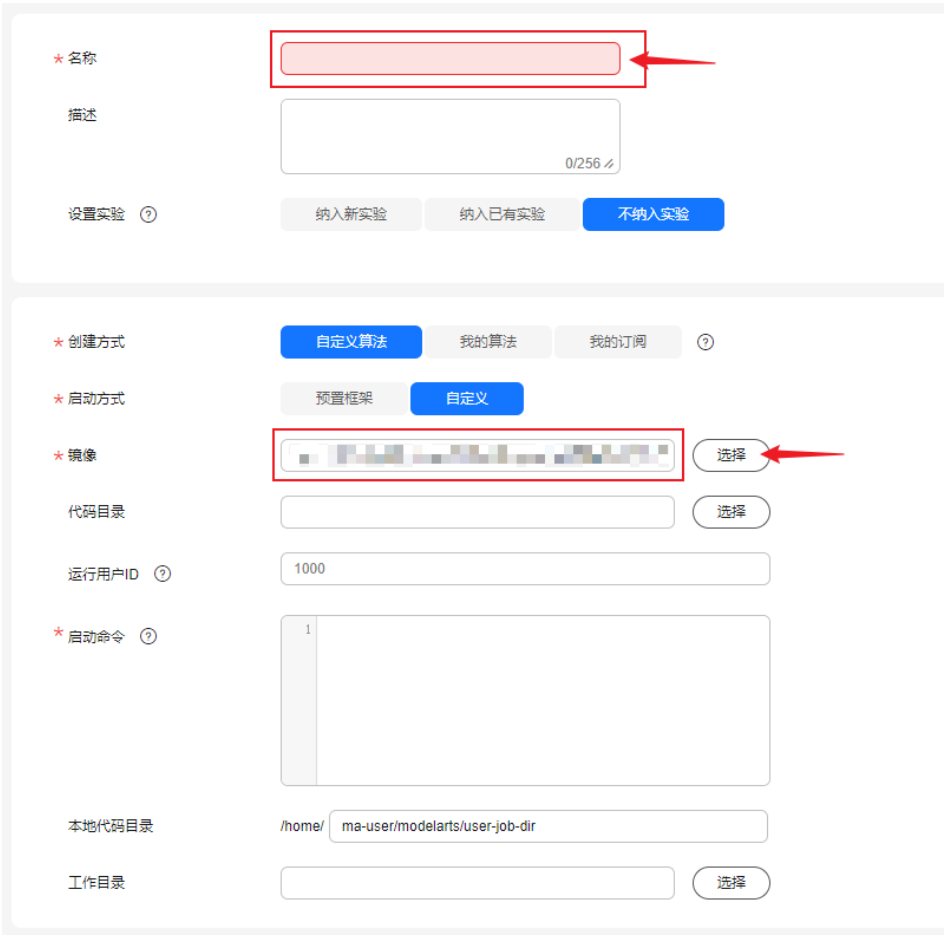
参数	示例值	参数说明
ASCEND_PROCESS_LOG_PATH	/home/ma-user/work/AscendSpeed/saved_dir_for_output/plog	保存训练过程中记录的程序堆栈信息日志 PLOG 文件。示例中，默认保存在“saved_dir_for_output/plog”文件夹下。若用户需要修改，可添加并自定义该变量。

对于Yi系列模型、ChatGLMv3-6B和Qwen系列模型，还需要手动修改训练参数和tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-68 选择镜像



若镜像使用**使用基础镜像**中的基础镜像时，训练作业启动命令中输入：

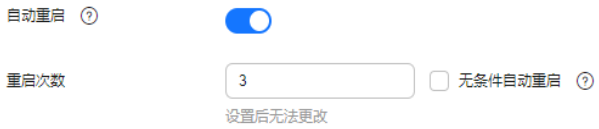
```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

若镜像使用**ECS中构建新镜像**和**Notebook中构建新镜像**构建的新镜像时，训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 3-69 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数。

说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。

注：训练作业中的训练故障自动恢复功能包括：

- 训练容错检查（自动重启），帮助用户隔离故障节点，优化用户训练体验。详细可了解：**训练容错检查**
- 无条件自动重启，不管什么原因系统都会自动重启训练作业，提高训练成功率和提升作业的稳定性。详细可了解：**无条件自动重启**。

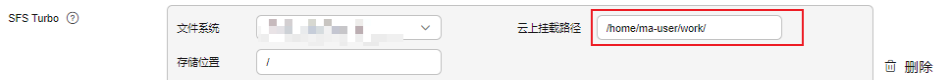
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-47**进行配置。

图 3-70 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在**创建Notebook**的“子目录挂载”路径。若默认没有填写，则忽略。

图 3-71 选择 SFS Turbo



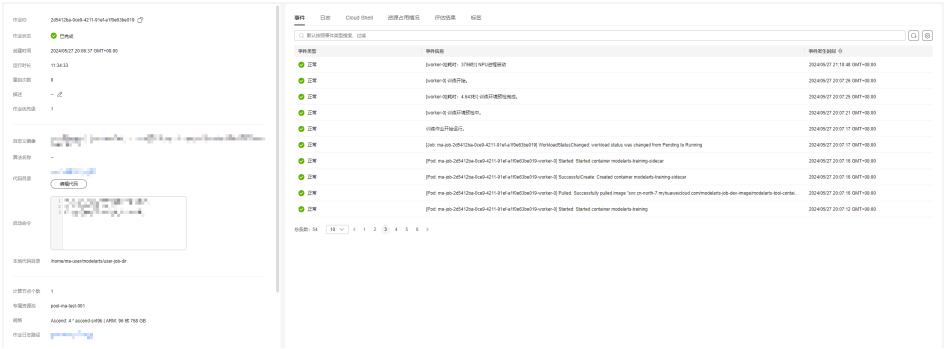
作业日志选择OBS中的路径，ModelArts的训练作业的日志信息则保存该路径下。

最后，请参考**查看日志和性能**章节查看LoRA微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.5.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

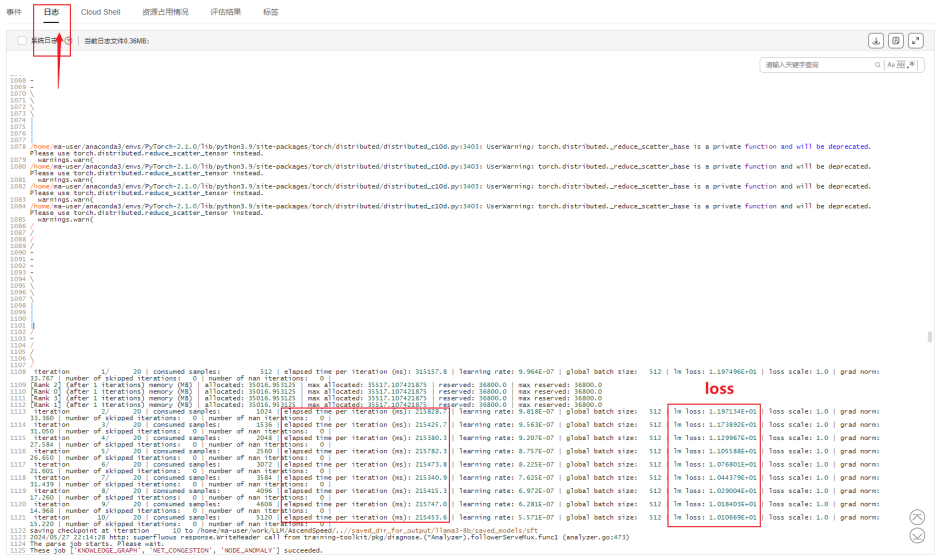
图 3-72 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 3-73 查看日志和性能



3.5.7 训练脚本说明

3.5.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 3-46 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
SHELL_FOLDER	$(dirname $(readlink -f "$0"))	表示执行脚本时的路径。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。

参数	示例值	参数说明
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表3-47所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-47 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

3.5.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。

- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/fintune/**

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-48 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。

环境变量	示例	参数说明
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.5.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。

- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-49 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径

参数	示例	参数说明
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.5.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

Yi 模型

在使用Yi模型的chat版本时，由于transformer 4.38版本的bug，导致在读取tokenizer文件时，加载的vocab_size出现类似如下尺寸不匹配的问题。

```
RuntimeError: Error(s) in loading state_dict for VocabParallelEmbedding:
 size mismatch for weight: copying a param with shape torch.Size([64000, 4096]) from checkpoint, the shape in current model is torch.Size([63992, 4096]).
```

需要在训练开始前，修改llm_train/AscendSpeed/yi/3_training.sh文件，并添加--tokenizer-not-use-fast参数。修改后如图3-74所示。

图 3-74 修改 Yi 模型 3_training.sh 文件

```
if [ ${MODEL_TYPE} == "yi-6b" ]; then
    model_args="
        --num-layers 32 \
        --hidden-size 4096 \
        --num-attention-heads 32 \
        --ffn-hidden-size 11008 \
        --group-query-attention \
        --num-query-groups 4 \
        --tokenizer-not-use-fast \
    "
elif [ ${MODEL_TYPE} == "yi-34b" ]; then
    model_args="
        --num-layers 60 \
        --hidden-size 7168 \
        --num-attention-heads 56 \
        --ffn-hidden-size 20480 \
        --group-query-attention \
        --num-query-groups 8 \
        --tokenizer-not-use-fast \
    "
```

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-75所示。

图 3-75 修改 ChatGLMv3-6B tokenizer 文件

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-76所示。

图 3-76 修改 ChatGLMv3-6B tokenizer 文件

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

294行要添加注释，修改后如图3-77所示。

图 3-77 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
```

314至323行要修改，修改后如图3-78所示。

图 3-78 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-79所示。

图 3-79 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41
42
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.5.8 常见错误原因和解决方法

3.5.8.1 显存溢出错误

在训练过程中，常见显存溢出报错，示例如下：

RuntimeError: NPU out of memory. Tried to allocate 1.04 GiB (NPU 4; 60.97 GiB total capacity; 56.45 GiB already allocated; 56.45 GiB current active; 1017.81 MiB free; 56.84 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.

解决方法

- 通过npu-smi info查看是否有进程资源占用NPU，导致训练时显存不足。解决可通过kill掉残留的进程或等待资源释放。
- 可调整参数：TP张量并行（tensor-model-parallel-size）和PP流水线并行（pipeline-model-parallel-size），可以尝试增加 TP和PP的值，一般TP×PP≤NPU数量，并且要被整除，具体调整值可参照表3-11进行设置。
- 可调整参数：MBS指最小batch处理的样本量（micro-batch-size）、GBS指一个iteration所处理的样本量（global-batch-size）。可将MBS参数值调小至1，但需要遵循GBS/MBS的值能够被NPU/(TP×PP)的值进行整除。
- 可调整参数：SEQ_LEN要处理的最大的序列长度（seq-length），参数值过大很容易发生显存溢出的错误。
- 可添加参数：在3_training.sh文件中添加开启重计算的参数。其中recompute-num-layers的值为模型网络中num-layers的参数值。

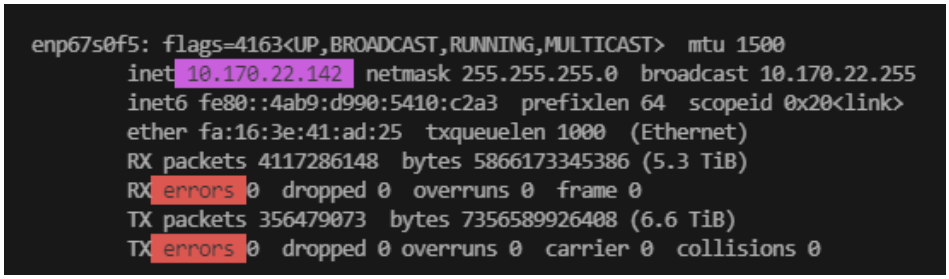
```
--recompute-granularity full \
--recompute-method block \
--recompute-num-layers {NUM_LAYERS} \
```

3.5.8.2 网卡名称错误

当训练开始时提示网卡名称错误。或者通信超时。可以使用ifconfig命令检查网卡名称配置是否正确。

比如，ifconfig看到当前机器IP对应的网卡名称为enp67s0f5，则可以设置环境变量指定该值。

图 3-80 网卡名称错误



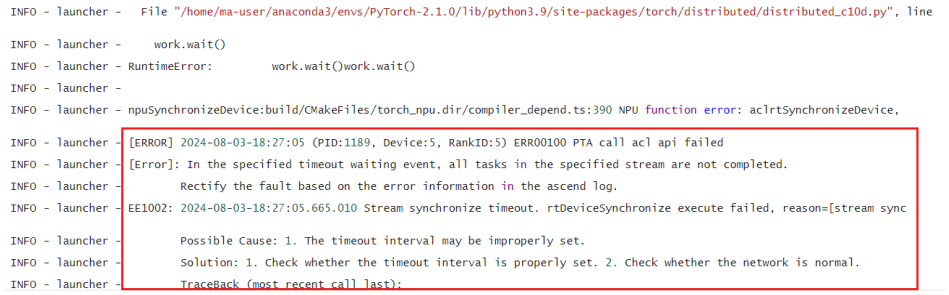

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # 多机之间使用gloo通信时需要指定网口名称，
export TP_SOCKET_IFNAME=enp67s0f5    # 多机之间使用TP通信时需要指定网口名称
export HCCL_SOCKET_IFNAME=enp67s0f5  # 多机之间使用HCCL通信时需要指定网口名称
```

关于环境变量的解释可以参考：**Distributed communication package - torch.distributed — PyTorch 2.3 documentation**

3.5.8.3 保存 ckpt 时超时报错

在多节点集群训练完成后，只有部分节点会保存权重，而其他节点会一直在等待通信。当等待时间超过36分钟时，会发生超时的错误。

图 3-81 报错提示



解决方法

1. 需要保证磁盘IO带宽正常，可以在36分钟内将文件保存到磁盘。单个节点内，最大只有60G（实际应该在40G以下）的文件内容，只要在36分钟内保存完成，就不会报超时错误。

2. 忽略该报错，因为报错不影响实际报错的权重。

3.6 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.907）

3.6.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台

的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

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

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.5.0版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。
- 专属资源池驱动版本要求23.0.6。

支持的模型列表和权重文件

本方案支持vLLM的v0.5.0版本。不同vLLM版本支持的模型列表有差异，具体如**表3-50**所示。

表 3-50 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**
3	llama-65b	√	√	√	√	**https://huggingface.co/huggyllama/llama-65b**
4	llama2-7b	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
5	llama2-13b	√	√	√	√	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	x	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemmma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
32	gemmma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
38	falcon2-11b	√	x	x	x	https://huggingface.co/tiiuae/falcon-11B/tree/main
39	qwen2-57b-a14b	√	x	x	x	https://huggingface.co/Qwen/Qwen2-57B-A14B-Instruct
40	llama3.1-8b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
41	llama3.1-70b	√	x	x	x	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

操作流程

图 3-82 操作流程图

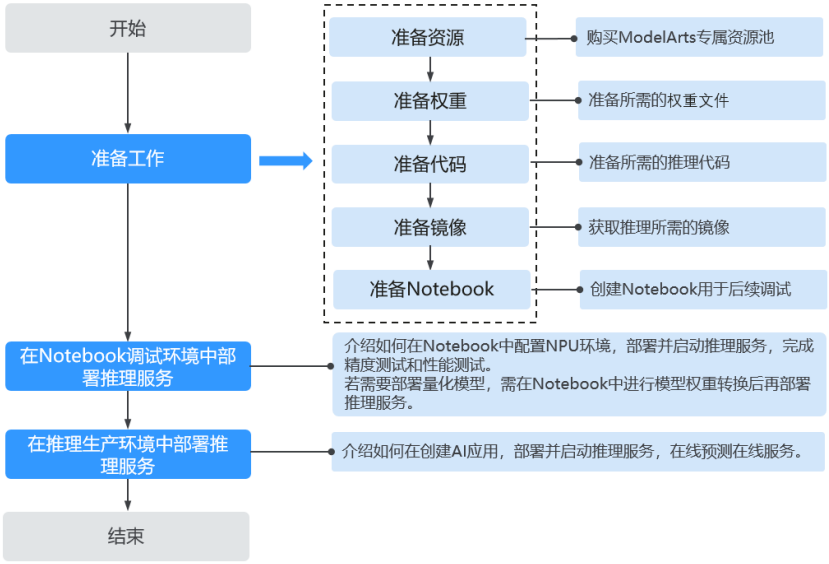


表 3-51 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.907-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 若需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

3.6.2 准备工作

3.6.2.1 准备资源

创建专属资源池

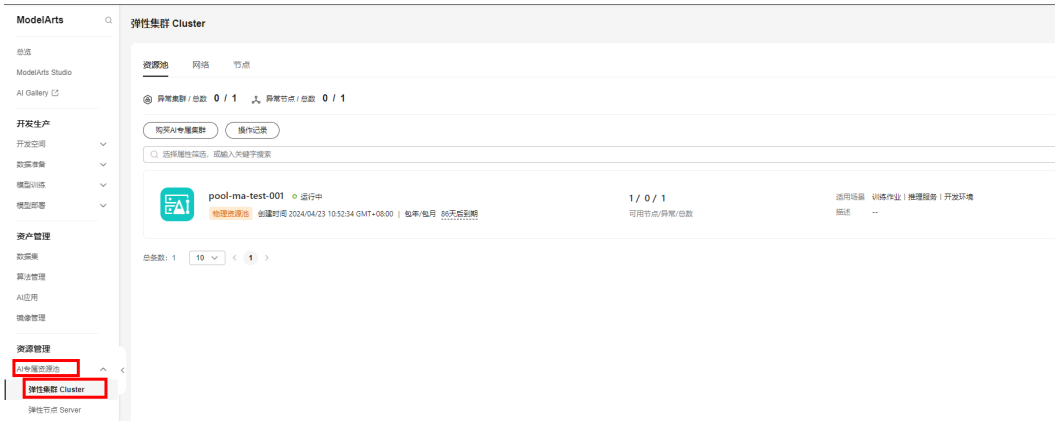
本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

专属资源池驱动检查

登录ModelArts控制台，单击“专属资源池 > 弹性集群”，选择创建的专属资源池。

图 3-83 查看专属资源池



在专属池详情页可查看驱动及固件版本。如下图显示Ascend驱动为7.1.0.7.220-23.0.5，表示固件版本为7.1.0.7.220，驱动版本为23.0.5。

图 3-84 查看专属池驱动

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

3.6.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**支持的模型列表和权重文件**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

3.6.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-52**所示。

表 3-52 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.907-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.907中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.5.0-py3-none-any.whl   # 推理安装包
       ├── build.sh                # 推理构建脚本
       ├── vllm_install.patch       # 社区昇腾适配的补丁包
       ├── Dockerfile              # 推理构建镜像dockerfile
       ├── build_image.sh          # 推理构建镜像启动脚本
 ├──llm_tools          # 推理工具包
```

```
├──AutoSmoothQuant  # W8A8量化工具
    ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
    ├── autosmoothquant                 # 量化代码
    ├── build.sh                        # 安装量化模块的脚本
├──AutoAWQ              # W4A16量化工具
    ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──quantize.py   # 昇腾适配的量化转换脚本
    ├──build.sh      # 安装量化模块的脚本
├──llm_evaluation    # 推理评测代码包
  ├──benchmark_tools    #性能评测
    ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
    ├── benchmark_parallel.py   # 评测静态性能脚本
    ├── benchmark_serving.py    # 评测动态性能脚本
    ├── benchmark_utils.py      # 抽离的工具集
    ├── generate_datasets.py    # 生成自定义数据集的脚本
    ├── requirements.txt        # 第三方依赖
  ├──benchmark_eval #精度评测
    ├──opencompass.sh       #运行opencompass脚本
    ├──install.sh           #安装opencompass脚本
    ├──vllm_api.py          #启动vllm api服务器
    ├──vllm.py              #构造vllm评测配置脚本名字
```

3.6.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-53 基础容器镜像地址

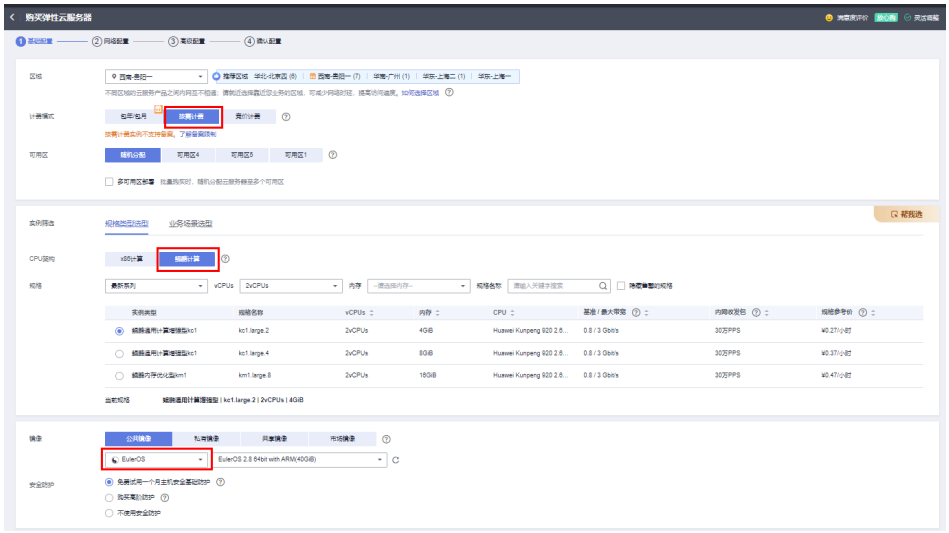
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 3-85 购买 ECS



Step2 创建镜像组织

在SWR服务页面创建镜像组织。

图 3-86 创建镜像组织

Step3 安装 Docker

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

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

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

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包，并上传到ECS的目录下（可自定义路径），获取地址参考**表3-52**。

解压AscendCloud压缩包及该目录下的推理代码AscendCloud-LLM-6.3.907-xxx.zip和算子包AscendCloud-OPP-6.3.907-xxx.zip，并执行build_image.sh脚本制作推理镜像。安装过程需要连接互联网git clone，请确保ECS可以访问公网。

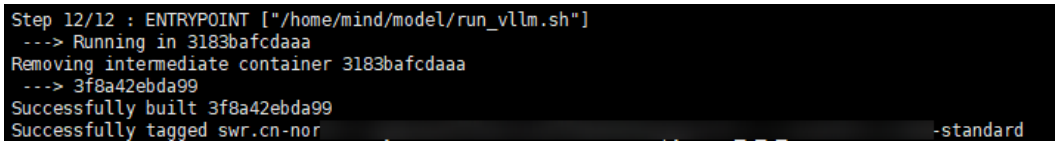
```
unzip AscendCloud-*.zip -d ./AscendCloud && unzip ./AscendCloud/AscendCloud-OPP-*.zip -d ./AscendCloud/AscendCloud-OPP && unzip ./AscendCloud/AscendCloud-LLM-*.zip -d ./AscendCloud/AscendCloud-LLM && cd ./AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/ && sh build_image.sh --base-image=${base_image}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

参数说明：

- ${base_image}为基础镜像；
- ${image_name}为推理镜像名称，示例：swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>。<组织名称>为**Step2 创建镜像组织**中创建的组织名称，<镜像名称>:<tag>为自定义镜像名称。

打印如下信息，表示构建镜像成功。

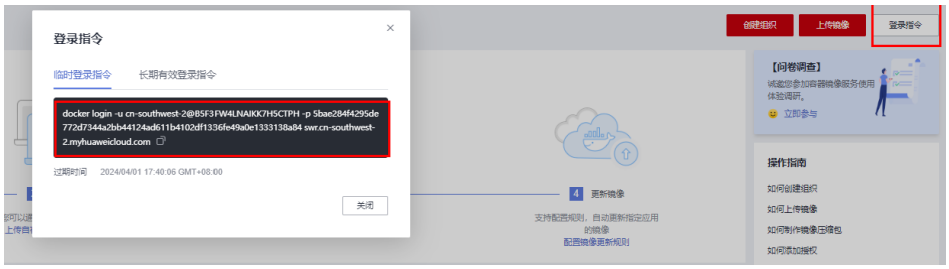
图 3-87 成功构建镜像



Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-88 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

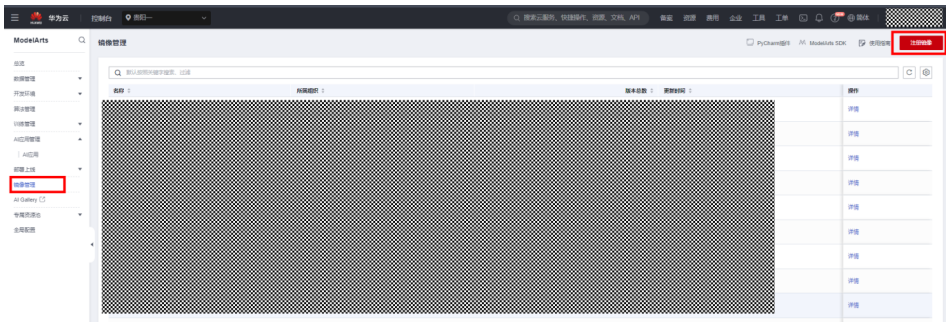
打印如下信息，表示上传镜像成功。

图 3-89 成功上传镜像

Step8 注册镜像

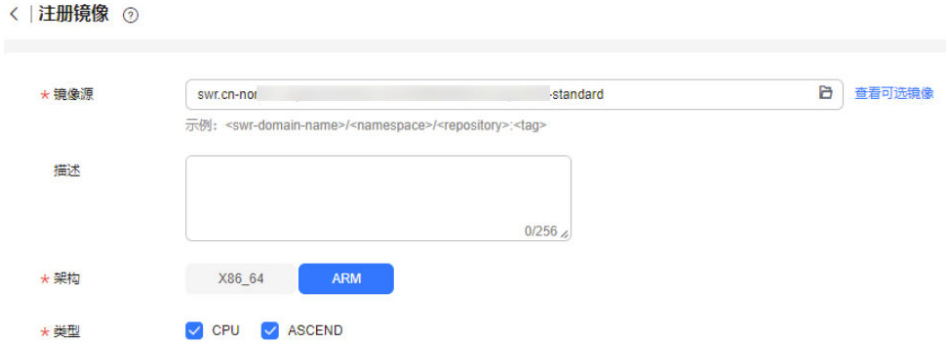
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中点击“注册镜像”。

图 3-90 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 3-91 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

3.6.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，点击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

图 3-92 创建 Notebook

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 3-93 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Asecnd snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。。

3.6.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

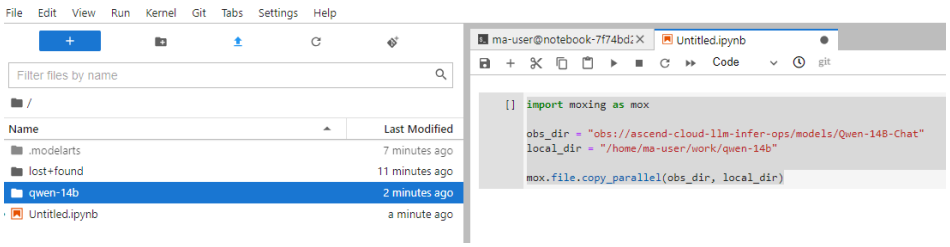
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 3-94 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡为容器中的第几张卡。例如：实际使用的是容器中第1张卡，此处填写“0”。

```
export ASCEND_RT_VISIBLE_DEVICES=0
```

如果启动服务需要使用多张卡，则按容器中的卡号依次编排。例如：实际使用的是容器中第1张和第2张卡，此处填写为“0,1”，以此类推。

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

说明

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

图 3-95 查询结果

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

2. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **通过vLLM服务API接口启动服务**

  在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model="${model_path}" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：/home/ma-user/work/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
        "prompt": "hello",
        "max_tokens": 100,
        "temperature": 0,
        "ignore_eos": false,
        "presence_penalty":2
 }'
```

vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**

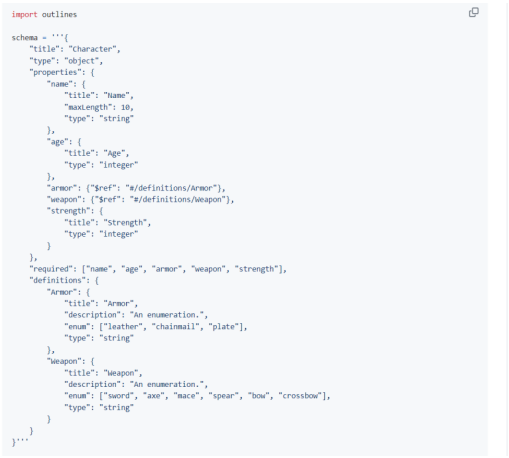
- 方式二：使用OpenAI接口请求服务，命令参考如下。

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

表 3-54 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

参数	是否必选	默认值	参数类型	描述
guided_json	否	None	Union[str, dict, BaseModel]	使用openai启动服务，若需要使用JSON Schema时要配置guided_json参数。 JSON Schema使用专门的关键字来描述数据结构，例如标题title、 类型type、属性properties，必需属性required 、定义definitions等，JSON Schema通过定义对象属性、类型、格式的方式来引导模型生成一个包含用户信息的JSON对象。 若希望使用JSON Schema，guided_json的写法可参考**outlines: Structured Text Generation**中的“Efficient JSON generation following a JSON Schema”样例，如下图所示。 **图 3-96** guided_json 样例  若想在发送的请求中包含上述guided_json架构，可参考以下代码。如果prompt未提供充足信息可能导致返回的json文件部分结果为空。 curl -X POST http://${docker_ip}:8080/v1/completions \ -H "Content-Type: application/json" \ -d '{ "model": "${container_model_path}", "prompt": "Meet our valorous character, named Knight, who has reached the age of 32. Clad in impenetrable plate armor, Knight is well-prepared for any battle. Armed with a trusty sword and boasting a strength score of 90, this character stands as a formidable warrior on the field.Please provide details for this character, including their Name, Age, preferred Armor, Weapon, and Strength", "max_tokens": 200, "temperature": 0, "guided_json": "{\"title\": \"Character\", \"type\": \"object\", \"properties\": {\"name\": {\"title\": \"Name\", \"maxLength\": 10, \"type\": \"string\"}, \"age\":

参数	是否必选	默认值	参数类型	描述
				{\"title\": \"Age\", \"type\": \"integer\"}, \"armor\": {\"$ref\": \"#/definitions/Armor\"}, \"weapon\": {\"$ref\": \"#/definitions/Weapon\"}, \"strength\": {\"title\": \"Strength\", \"type\": \"integer\"}}, \"required\": [\"name\", \"age\", \"armor\", \"weapon\", \"strength\"], \"definitions\": {\"Armor\": {\"title\": \"Armor\", \"description\": \"An enumeration.\", \"enum\": [\"leather\", \"chainmail\", \"plate\"], \"type\": \"string\"}, \"Weapon\": {\"title\": \"Weapon\", \"description\": \"An enumeration.\", \"enum\": [\"sword\", \"axe\", \"mace\", \"spear\", \"bow\", \"crossbow\"], \"type\": \"string\"}}}" }'

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9。

表 3-55 不同模型推理支持的 max-model-len 长度

模型名	280T		313T	
	最小卡数	最大序列(K)	最小卡数	最大序列(K)
llama-7b	1	16	1	32
llama-13b	2	16	1	16
llama-65b	8	16	4	16
llama2-7b	1	16	1	32
llama2-13b	2	16	1	16
llama2-70b	8	32	4	64
llama3-8b	1	32	1	128
llama3-70b	8	32	4	64
qwen-7b	1	8	1	32
qwen-14b	2	16	1	16
qwen-72b	8	8	4	16

模型名	280T		313T	
	最小卡数	最大序列(K)	最小卡数	最大序列(K)
qwen1.5-0.5b	1	128	1	256
qwen1.5-7b	1	8	1	32
qwen1.5-1.8b	1	64	1	128
qwen1.5-14b	2	16	1	16
qwen1.5-32b	4	32	2	64
qwen1.5-72b	8	8	4	16
qwen1.5-110b	oom		8	128
qwen2-0.5b	1	128	1	256
qwen2-1.5b	1	64	1	128
qwen2-7b	1	32	1	64
qwen2-72b	8	32	4	64
chatglm2-6b	1	64	1	128
chatglm3-6b	1	64	1	128
glm-4-9b	1	32	1	128
baichuan-7b	1	16	1	32
baichuan-13b	2	4	1	4
baichuan2-7b	1	8	1	32
baichuan2-13b	2	4	1	4
yi-6b	1	64	1	128
yi-9b	1	32	1	64
yi-34b	4	32	2	64
deepseek-llm-7b	1	16	1	32
deepseek-coder-instruct-33b	4	32	2	64
deepseek-llm-67b	8	32	4	64
mistral-7b	1	32	1	128
mixtral-8x7b	4	8	2	32
gemma-2b	1	64	1	128
gemma-7b	1	8	1	32

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

3.6.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**支持的模型列表和权重文件**。

  说明

  若需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。

- 推理启动脚本run_vllm.sh制作请参见下文**创建推理脚本文件run_vllm.sh**的介绍。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 3-97 准备模型文件和权重文件

创建推理脚本文件run_vllm.sh

run_vllm.sh脚本示例如下。

- **通过vLLM服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
python -m vllm.entrypoints.api_server --model="${model_path}" \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
python -m vllm.entrypoints.openai.api_server --model="${model_path}" \
--ssl-keyfile="/home/mind/model/key.pem" \
--ssl-certfile="/home/mind/model/cert.pem" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

参数说明：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：home/mind/model/chatglm3-6b。
- --tensor-parallel-size：并行卡数。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。如果不指定，则根据输入数据自动匹配数据类型。
- --distributed-executor-backend：多卡推理启动后端，可选值为"ray"或者"mp"，其中"ray"表示使用ray进行启动多卡推理，"mp"表示使用python多进程进行启动多卡推理。默认使用"mp"后端启动多卡推理。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。

- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。
```

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 3-98 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 3-99 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

   首次创建AI应用预计花费40~60分钟，之后每次构建AI应用花费时间预计5分钟。

图 3-100 创建完成

说明

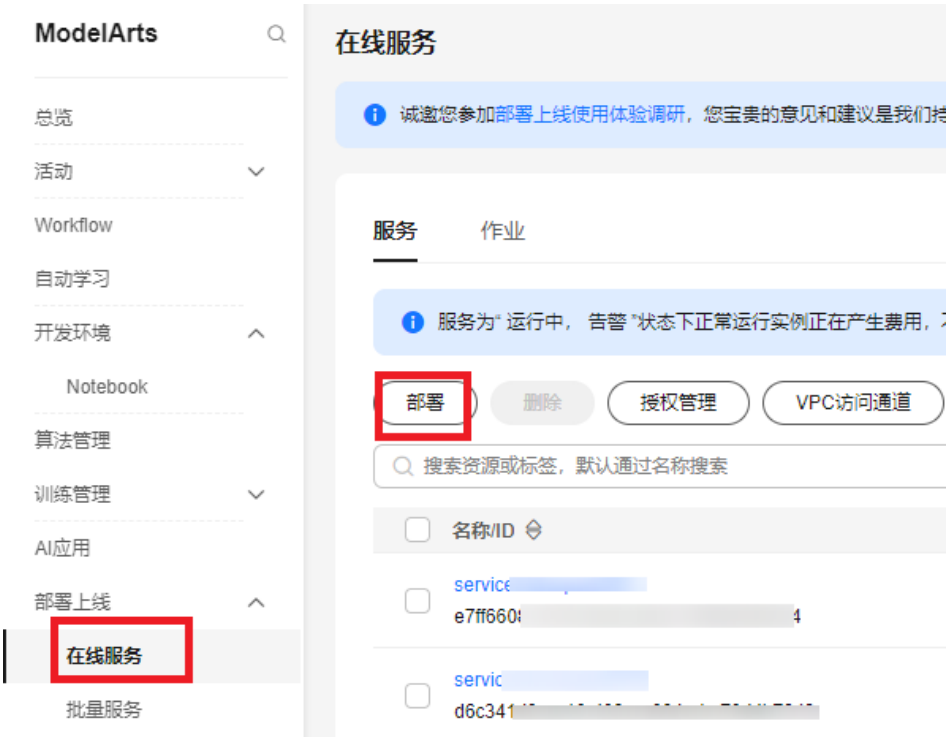
若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台中，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 3-101 部署在线服务



2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 3-102 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 3-103 服务部署完成

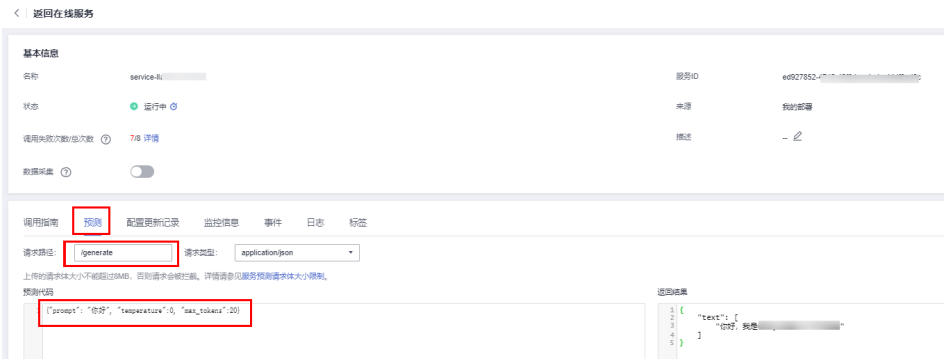
注：若部署在线服务出现报错starting container process caused "exec: \"/home/mind/model/run_vllm.sh\": permission denied"，请参考**附录：大模型推理standard常见问题**问题6重新构建镜像。

Step4 调用在线服务

进入在线服务详情页面，选择“预测”。

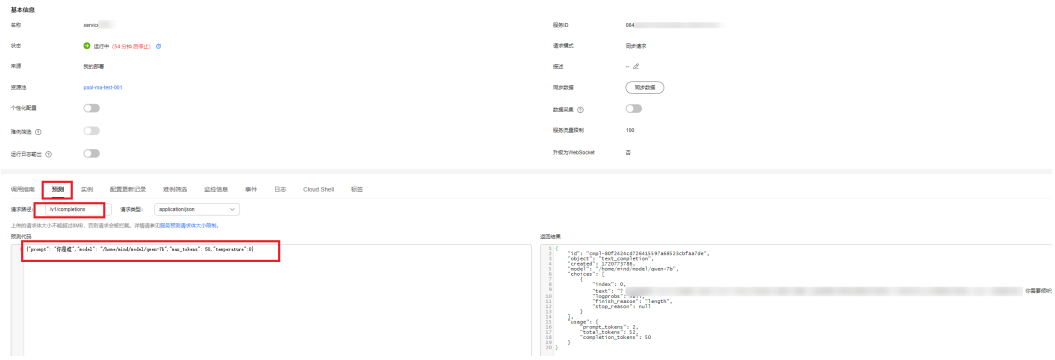
若以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”既可看到预测结果。

图 3-104 预测-vllm



若以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”既可看到预测结果。

图 3-105 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

3.6.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

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

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py             #构造vllm评测配置脚本名字
```

2. 精度评测切换conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。

```
conda activate python-3.9.10
```

3. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**4**进行评测。

```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```

4. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。

```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
```

```
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```

参数说明:

- vllm_path：构造vllm评测配置脚本名字，默认为vllm。
- service_port：服务端口，与启动服务时的端口保持，比如8080。
- max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen，不同数据集可以详见opencompass下面data目录。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：作为一个保存log结果中的一个变量名，默认选eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

5. （可选）如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

   ```
   cd opencompass
   python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
   ```

   output_path: 要保存的结果路径。

6. （可选）创建新conda环境，安装vllm和opencompass。执行完之后，在opencompass/configs/models/vllm/vllm_ppl.py 里是ppl的配置项。由于离线执行推理，消耗的显存相当庞大。其中以下参数需要根据实际来调整。

   - batch_size, 推理时传入的 prompts 数量，可配合后面的参数适当减少
   - offline，是否启动离线模型，使用 ppl 时必须为 True
   - tp_size，使用推理的卡数
   - max_seq_len，推理的上下文长度，和消耗的显存直接相关，建议稍微高于prompts。其中，mmlu和ceval 建议 3200

   另外，在 opencompass/opencompass/models/vllm_api.py 中，可以适当调整gpu_memory_utilization。如果还是 oom，建议适当往下调整。

   最后，如果执行报错提示oom，建议修改数据集的shot配置。例如mmlu，可以修改文件 opencompass/configs/datasets/mmlu/mmlu_ppl_ac766d.py 中的

   fix_id_list, 将最大值适当调低。

   ppl困惑度评测一般用于base权重测评，会将n个选项上拼接上下文，形成n个序列，再计算着n个序列的困惑度(perplexity)。其中，perplexity最小的序列所对应的选项即为这道题的推理结果。运行时间比较长，例如llama3_8b 跑完mmlu要2~3小时。

   在npu卡上，使用多卡进行推理时，需要预制变量

   ```
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:False
   ```

   执行脚本如下：

   ```
   python run.py --models vllm_ppl --datasets mmlu_ppl -w ${output_path}
   ```

   output_path 指定保存结果的路径。

参考模型llama3系列模型，数据集 mmlu 为例，配置如下：

表 3-56 参数配置

模型	max_seq_len	batch_size	shot数
llama3_8b	3200	8	采用默认值
llama3_70b	3200	4	[0, 1, 2]

7. (可选) opencompass也支持通过本地权重来进行ppl精度测试。本质上使用transformers进行推理，因为没有框架的优化，执行时间最长。另一方面，由于是使用transformers推理，结果也是最稳定的。对单卡运行的模型比较友好，算力利用率比较高。对多卡运行的推理，缺少负载均衡，利用率低。

   在昇腾卡上执行时，需要在 opencompass/opencompass/runners/local.py 中添加如下代码

   ```
   import torch
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   执行脚本如下

   ```
   # for llama3_8b
   python run.py --datasets mmlu_ppl \
   --hf-type base --hf-path {hf-path} \
   --max-seq-len 3200 --max-out-len 16 --hf-num-gpus 1 --batch-size 4 \
   -w {output_path} --debug
   ```

   参数说明如下:

   - --datasets, 评测的数据集及评测方法，其中 mmlu 是数据集，ppl 是评测方法
   - --hf-type, HuggingFace模型权重类型(base,chat), 默认为chat, 依据实际的模型选择
   - --hf-path, 本地 HuggingFace 权重的路径，比如/home/ma-user/nfs/model/Meta-Llama-3-8B
   - --max-seq-len, 模型的最大序列长度
   - --max-out-len, 模型的最大输出长度
   - --hf-num-gpus, 需要使用的卡数
   - --batch-size, 推理每次处理的输入数目
   - -w 存放输出结果的目录

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

```
npu：
mmlu：46.6
gpu：
mmlu：47
```

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1%以内（计算公式：(47-46.6)/47*100=0.85%）认为NPU精度和GPU对齐。

3.6.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，若静态benchmark的并发数（parallel-num参数）或动态benchmark的请求频率（request-rate参数）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（例如输出大于1k），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

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

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

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

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
conda activate python-3.9.10
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

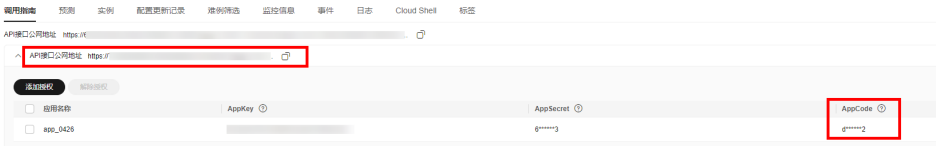
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 3-106 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

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

图 3-107 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57295766
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

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

   公开数据集下载地址：

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

   使用generate_dataset.py脚本生成数据集方法：

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

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

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。

- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

   Notebook中进行测试：

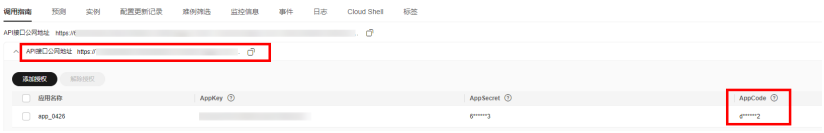
   ```
   conda activate python-3.9.10
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 3-108 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --dataset：数据集路径。
   - --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
   - --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
   - --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
   - --benchmark-csv：结果保存路径，如benchmark_serving.csv。

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

图 3-109 动态 benchmark 测试结果（示意图）

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

3.6.7 推理模型量化

3.6.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化，量化方法为per-group。

Step1 模型量化

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

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

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

```
export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
python examples/quantize.py --model-path /home/ma-user/llama-2-7b/ --quant-path /home/ma-user/llama-2-7b-awq/ --calib-data /home/ma-user/mit-han-lab/pile-val-backup
```

参数说明:

- --model-path：原始模型权重路径。
- --quan-path：转换后权重保存路径。
- --calib-data：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst，注意需指定到val.jsonl的上一级目录。

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

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools/AutoAWQ代码目录下执行以下脚本：

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

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

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

3.6.7.2 使用 SmoothQuant 量化工具转换权重

SmootQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmootQuant(W8A8)量化的模型列表请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

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

代码目录如下:

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

具体操作如下：

1. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
2. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。
   - --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
   - --scale-output：量化系数保存路径。
   - --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
   - --model-output：量化模型权重保存路径。
   - --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
   - --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。

- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

3. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

3.6.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**支持的模型列表和权重文件**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

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

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

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

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

Step2 抽取 kv-cache 量化系数

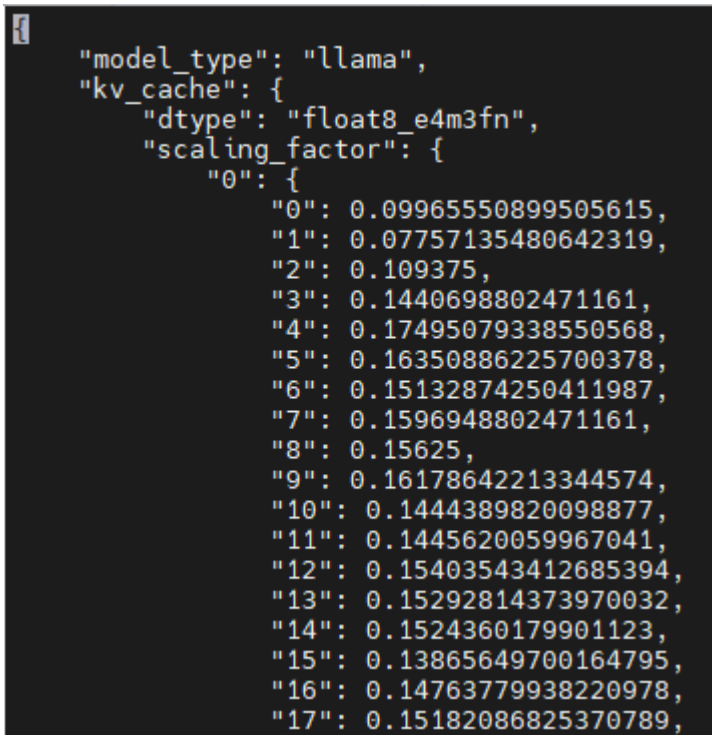
该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

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

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

图 3-110 抽取 kv-cache 量化系数



注意：

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

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

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

3.6.8 附录：基于 vLLM 不同模型推理支持最小卡数和最大序列说明

基于vLLM（v0.5.0）部署推理服务时，不同模型推理支持的最小昇腾卡数和对应卡数下的max-model-len长度说明，如下面的表格所示。

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

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

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

表 3-57 基于 vLLM 不同模型推理支持最小卡数和最大序列说明

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

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
20	qwen2-1.5b	1	64	1	128
21	qwen2-7b	1	8	1	32
22	qwen2-72b	8	32	4	64
23	chatglm2-6b	1	64	1	128
24	chatglm3-6b	1	64	1	128
25	glm-4-9b	1	32	1	128
26	baichuan2-7b	1	8	1	32
27	baichuan2-13b	2	4	1	4
28	yi-6b	1	64	1	128
29	yi-9b	1	32	1	64
30	yi-34b	4	32	2	64
31	deepseek-llm-7b	1	16	1	32
32	deepseek-coder-instruct-33b	4	32	2	64
33	deepseek-llm-67b	8	32	4	64
34	mistral-7b	1	32	1	128
35	mixtral-8x7b	4	8	2	32
36	gemma-2b	1	64	1	128
37	gemma-7b	1	8	1	32

序号	模型名	32GB显存		64GB显存	
		最小卡数	最大序列(K) max-model-len	最小卡数	最大序列(K) max-model-len
38	falcon-11b	1	8	1	64

3.6.9 附录：大模型推理 standard 常见问题

- 问题1：在推理预测过程中遇到NPU out of memory。

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

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

  解决方法：修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

  config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

- 问题3：使用离线推理时，性能较差或精度异常。

  解决方法：将block_size大小设置为128。

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

- 问题4：使用llama3.1系模型进行推理时，报错：ValueError: 'rope_scaling' must be a dictionary with two fields, 'type' and 'factor', got {'factor': 8.0, 'low_freq_factor': 1.0, 'high_freq_factor': 4.0, 'original_max_position_embeddings': 8192, 'rope_type': 'llama3'}

  解决方法：升级transformers版本到4.43.1：pip install transformers --upgrade

- 问题5：使用SmootQuant进行W8A8进行模型量化时，报错：AttributeError: type object 'LlamaAttention' has no attribute '_init_rope'

  解决方法：降低transformers版本到4.42：pip install transformers==4.42 --upgrade

- 问题6：部署在线服务报错starting container process caused "exec: \"/home/mind/model/run_vllm.sh\": permission denied"

  解决方法：修改AscendCloud-6.3.907-xxx.zip压缩包中llm_inference/ascend_vllm/build_image.sh内容，将'ENTRYPOINT ["/home/mind/model/run_vllm.sh"]'修改为'ENTRYPOINT sh /home/mind/model/run_vllm.sh'，并重新构建镜像。

  见如下示例：

图 3-111 修改 build_images.sh

```
#!/bin/bash

OPTIONS=$(getopt -n "$0" -o i:e:n --long base-image:,specify-enrtypoint:,image-name: -- "$@")

if [ $? != 0 ]; then
    echo "args error"
    exit 1
fi

eval set -- "$OPTIONS"

while true; do
    case "$1" in
        -i|--base-image)
            base_image="$2"
            shift 2
            ;;
        -e|--specify-enrtypoint)
            specify_enrtypoint="$2"
            shift 2
            ;;
        -n|--image-name)
            image_name="$2"
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            echo "unknown options"
            exit 1
            ;;
    esac
done

if [ -z "$base_image" ]; then
    echo "--base-image not specified"
    exit 1
fi

if [ -z "$image_name" ]; then
    echo "--image-name not specified"
    exit 1
fi

if [ -n "$specify_enrtypoint" ]; then
    echo 'ENTRYPOINT sh /home/mind/model/run_vllm.sh' >> Dockerfile
fi

cd ../../../../
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile ./
cp AscendCloud/AscendCloud-LLM/llm_inference/ascend_vllm/.dockerignore ./

docker build -t $image_name --build-arg BASE_IMAGE=$base_image .

rm -f Dockerfile
rm -f .dockerignore

```

3.7 主流开源大模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.906）

3.7.1 场景介绍

方案概览

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

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

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**表3-60**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite DevServer。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如**表3-58**所示。

表 3-58 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 3-112 操作流程图

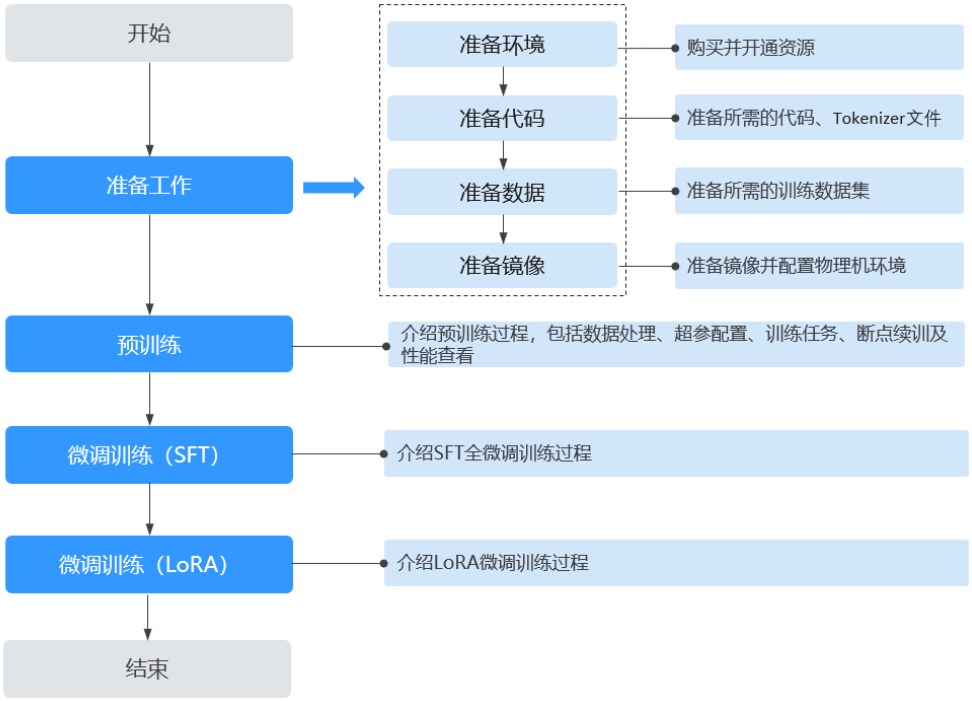


表 3-59 操作任务流程说明

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

3.7.2 准备工作

3.7.2.1 准备环境

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

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表3-68**。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.7.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表3-60**所示，模型列表、对应的开源权重获取地址如**表3-61**所示。

表 3-60 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

表 3-61 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**

序号	支持模型	支持模型参数量	权重文件获取地址
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

模型软件包结构说明

AscendCloud代码包结构介绍如下：

```
├──AscendCloud-LLM
  ├──llm_train                # 模型训练代码包
     ├──AscendSpeed           # 基于AscendSpeed的训练代码
        ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
        ├──scripts/             # 训练需要的启动脚本
           ├──llama2            # llama2系列模型执行脚本的文件夹
           ├──llama3            # llama3系列模型执行脚本的文件夹
           ├──qwen              # Qwen系列模型执行脚本的文件夹
           ├──qwen1.5           # Qwen1.5系列模型执行脚本的文件夹
           ├── ...
           ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
           ├── install.sh       # 环境部署脚本
        ├──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
  ├──llm_inference              # 推理代码包
```

```
   |——llm_tools                    # 推理工具
|——AscendCloud-OPP                 #依赖算子包
```

工作目录介绍

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

```
${workdir}（例如/home/ma-user/ws ）
|——llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |—— AscendSpeed               # 代码目录
        |——ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |——scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
    # 自动生成数据目录结构
    |—— processed_for_input          #目录结构会自动生成，无需用户创建
        |—— ${model_name}              # 模型名称
            |—— data              # 预处理后数据
          |—— pretrain   # 预训练加载的数据
          |—— finetune   # 微调加载的数据
      |——converted_weights  # HuggingFace格式转换magatron格式后权重文件
    |—— saved_dir_for_output           # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |—— ${model_name}              # 模型名称
            |—— logs              # 训练过程中日志（loss、吞吐性能）
              |—— saved_models
            |—— lora              # lora微调输出权重
            |—— sft               # 增量训练输出权重
            |—— pretrain          # 预训练输出权重
|—— tokenizers                    #原始权重及tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |—— Llama2-70B
|—— training_data                 #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |—— train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |—— alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

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

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

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

   具体步骤如下：

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

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

3.7.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

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

自定义数据

用户也可以自行准备训练数据。数据要求如下：

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

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

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

3.7.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-62 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

表 3-63 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

步骤 1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤 2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

步骤 3 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 200g \
```



```
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

1. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
3. 使用ma-user用户安装依赖包。
   ```
   #进入scripts目录换
   cd /home/ma-user/ws/llm_train/AscendSpeed
   #执行安装命令
   sh scripts/install.sh
   ```
4. 通过运行install.sh脚本，还会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：
   ```
   |——AscendCloud-LLM
     |——llm_train                # 模型训练代码包
       |——AscendSpeed            # 基于AscendSpeed的训练代码
         |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
         |——scripts/             # 训练需要的启动脚本
         |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/           # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/             # MindSpeed昇腾大模型加速库
       |——ModelLink/             # ModelLink端到端的大语言模型方案
         |——megatron/            # 注意：该文件夹从Megatron-LM中复制得到
         |——...
   ```

3.7.3 预训练任务

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-64**所示。其他超参均有默认值，可以参考**表3-67**按照实际需求修改。

表 3-64 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤 3 启动训练脚本

请根据**步骤2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为4机32卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NNODES、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
示例:
sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-113 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

3.7.4 SFT 全参微调训练任务

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-64**所示。其他超参均有默认值，可以参考**表3-67**按照实际需求修改。

表 3-65 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B、ChatGLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

步骤 3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4>
<NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NNODES、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

3.7.5 LoRA 微调训练

步骤 1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

步骤 2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-64**所示。其他超参均有默认值，可以参考**表3-67**按照实际需求修改。

表 3-66 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B、ChatGLMv4-9B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表3-68**所示。

步骤 3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=4> <NODE_RANK=0>
示例:
```

```
#第一台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  0
# 第二台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  1
# 第三台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  2
# 第四台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  4  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
```

训练完成后，生成的权重文件保存路径为：**/home/ma-user/ws/llm_train/saved_dir_for_output/llama2-13b/saved_models/**。

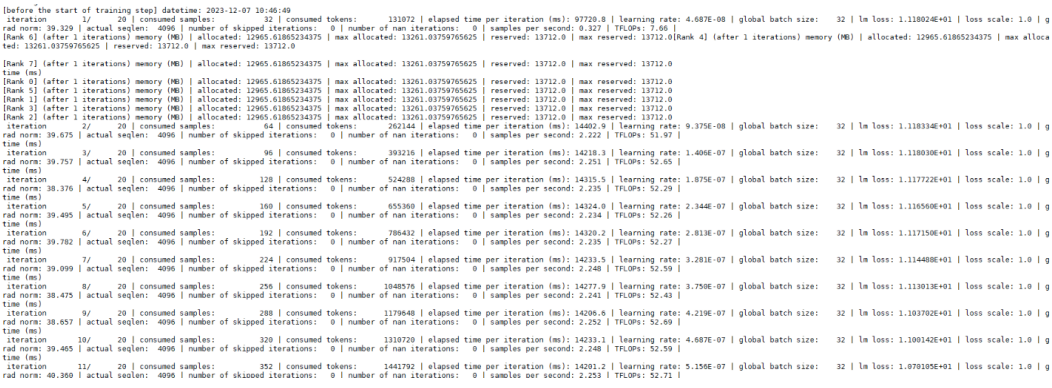
训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

3.7.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-114 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

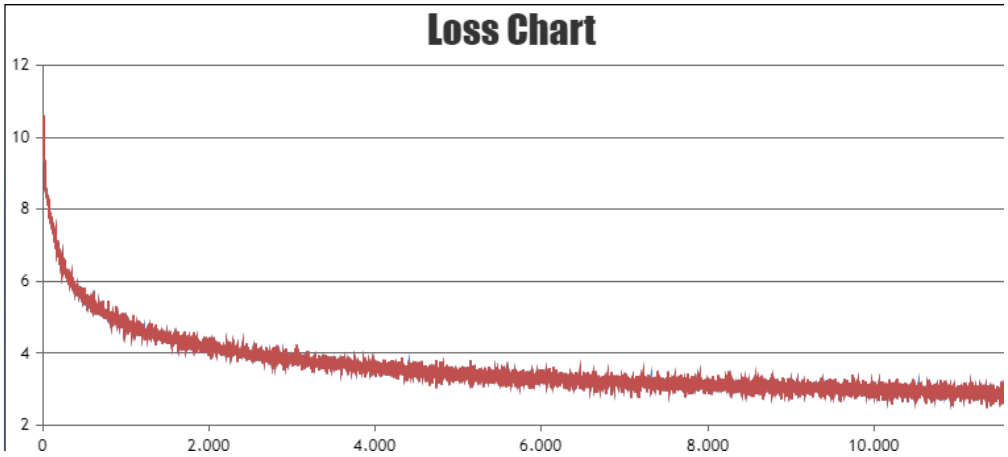
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表3-67**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-115**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-115 Loss 收敛情况（示意图）



3.7.7 训练脚本说明

3.7.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例。

表 3-67 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。

参数	示例值	参数说明
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSInstructionHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSInstructionHandler：使用微调的moss数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表3-68所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-68 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

3.7.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/fintune/**

handler-name 参数说明

数据集预处理中 --handler-name 都会传递参数，用于构建实际处理数据的hanler对象，并根据handler对象对数据集进行解析。文件路径在：ModelLink/modellink/data/data_handler.py。

- **基类BaseDatasetHandler解析**

  data_handler的基类是BaseDatasetHandler，其核心函数是serialize_to_disk：

```
def serialize_to_disk(self):
    """save idx and bin to disk"""
    startup_start = time.time()
    if not self.tokenized_dataset:
        self.tokenized_dataset = self.get_tokenized_data()
    output_bin_files = {}
    output_idx_files = {}
    builders = {}
    level = "document"
    if self.args.split_sentences:
        level = "sentence"
    logger.info("Vocab size: %s", self.tokenizer.vocab_size)
    logger.info("Output prefix: %s", self.args.output_prefix)
    for key in self.args.json_keys:
        ## 写入磁盘
```

- 先调用self.get_tokenized_data()对数据集进行encode
- self.get_tokenized_data()中调用self._filter方法处理每一个sample
- self._filter在基类中未定义，需要各个子类针对目标数据集格式进行实现

所有handler依据实际数据集实现self._filter方法，处理原始数据集中的单一sample，其余方法复用基类的实现。

- **GeneralPretrainHandler解析**

  GeneralPretrainHandler是处理预训练数据集的一个类，继承自BaseDatasetHandler，实现对alpaca格式预训练数据集的处理。

```
def _filter(self, sample):
    sample = self._pre_process(sample)
    for key in self.args.json_keys:
        text = sample[key]
        doc_ids = []
        for sentence in self.splitter.tokenize(text):
            if len(sentence) > 0:
                sentence_ids = self._tokenize(sentence)
                doc_ids.append(sentence_ids)
        if len(doc_ids) > 0 and self.args.append_eod:
            doc_ids[-1]['input_ids'].append(self.tokenizer.eod)
            doc_ids[-1]['attention_mask'].append(1)
            doc_ids[-1]['labels'].append(self.tokenizer.eod)
        sample[key] = doc_ids
        # for now, only input_ids are saved
        sample[key] = list(map(lambda x: x['input_ids'], sample[key]))
    return sample
```

- **GeneralInstructionHandler解析**

  GeneralInstructionHandler是处理微调数据集的一个基本类，继承自BaseDatasetHandler，实现对alpaca格式微调数据集的处理。

```
def _filter(self, sample):
    messages = self._format_msg(sample)
    full_prompt = self.prompter.generate_training_prompt(messages)
    tokenized_full_prompt = self._tokenize(full_prompt)
    if self.args.append_eod:
        tokenized_full_prompt["input_ids"].append(self.tokenizer.eod)
        tokenized_full_prompt["attention_mask"].append(1)
        tokenized_full_prompt["labels"].append(self.tokenizer.eod)
    if not self.train_on_inputs:
        user_prompt = full_prompt.rsplit(self.prompter.template.assistant_token, maxsplit=1)[0] + \
            self.prompter.template.assistant_token + "\n"
        tokenized_user_prompt = self._tokenize(user_prompt)
        user_prompt_len = len(tokenized_user_prompt["input_ids"])
        tokenized_full_prompt["labels"][:user_prompt_len] = [self.ignored_label] * user_prompt_len
    for key in self.args.json_keys:
        tokenized_full_prompt[key] = [tokenized_full_prompt[key]]
    return tokenized_full_prompt
```

- 对数据集 full_prompt 中的 user_prompt 进行 mask 操作。

- **MOSSMultiTurnHandler解析**

  MOSSMultiTurnHandler是处理微调数据集的一个类，继承自GeneralInstructionHandler，实现对moss格式微调数据集的处理。

```
def _filter(self, sample):
    input_ids, labels = [], []
    for turn in sample["chat"].values():
        if not turn:
            continue
        user = turn["Human"].replace("<eoh>", "").replace("<|Human|>: ", "").strip()
        assistant = turn["MOSS"].replace("<|MOSS|>:", "").replace("<eom>", "").strip()
        user_ids = self._unwrapped_tokenizer.encode(user)
        assistant_ids = self._unwrapped_tokenizer.encode(assistant)
        input_ids += self.user_token + user_ids + self.assistant_token + assistant_ids
        labels += [self._unwrapped_tokenizer.eos_token_id] + self.ignored_index * len(user_ids) +
```

```
self.ignored_index + assistant_ids
    input_ids.append(self._unwrapped_tokenizer.eos_token_id)
    labels.append(self._unwrapped_tokenizer.eos_token_id)
    attention_mask = [1 for _ in range(len(input_ids))]
    return {
        "input_ids": [input_ids],
        "attention_mask": [attention_mask],
        "labels": [labels]
    }
```

a. moss原始数据集是一个多轮对话的jsonl，filter的输入就是其中的一行

b. 循环处理其中的单轮对话

c. 在单轮对话中

   i. 对user和assiant的文本进行清洗

   ii. 分别encode处理后的文本，获得对应的token序列，user_ids和assiantant_ids

   iii. input_ids是user_ids和assiantant_ids的拼接

   iv. labels与input_ids对应，用-100替换user_ids的token,只保留assiantant_ids

d. attention_mask是和input_ids等长的全1序列

e. 返回input_ids\attention_mask\labels的字典

f. 处理完单一sample

注：labels中用-100填充的地方，表示会被loss_mask给mask掉

- **自定义handler**

  参考MOSSMultiTurnHandler的实现，继承想要的通用的父类，实现_filter方法，然后在数据预处理的参数里指定自己的handler名称即可

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值，进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 3-69 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**

环境变量	示例	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.7.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE。**若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。进入到 **/home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/ws/llm_train/AscendSpeed/ModelLink** 命令，随后运行该脚本。

其中环境变量详细介绍如下：

表 3-70 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.7.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-116所示。

图 3-116 修改 ChatGLMv3-6B tokenizer 文件

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-117所示。

图 3-117 修改 ChatGLMv3-6B tokenizer 文件

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

294行要添加注释，修改后如图3-118所示。

图 3-118 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
295 
```

314至323行要修改，修改后如图3-119所示。

图 3-119 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型（qwen-7b、qwen-14b、qwen-72b）中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-120所示。

图 3-120 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.8 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906）

3.8.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.4.2版本。
- 支持FP16和BF16数据类型推理。
- DevServer驱动版本要求23.0.5。

资源规格要求

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

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

说明

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

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-71 基础容器镜像地址

镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	cann_8.0.rc2

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表3-72所示。

表 3-72 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

支持的模型列表和权重文件

本方案支持vLLM的v0.4.2版本。不同vLLM版本支持的模型列表有差异，具体如表3-73所示。

表 3-73 支持的模型列表和权重获取地址

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

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen-72B-Chat

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
18	qwen1.5-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	√	√	x	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	qwen2-0.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
26	qwen2-1.5b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
27	qwen2-7b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-7B-Instruct
28	qwen2-72b	√	√	√	x	https://huggingface.co/Qwen/Qwen2-72B-Instruct
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemmma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
32	gemmma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

说明：当前版本中yi-34b、qwen1.5-32b模型暂不支持单卡启动。

支持的 rope scaling 类型

本方案支持的rope scaling类型包括linear、dynamic和yarn，其中linear方法只支持传入一个固定的scaling factor值，暂不支持传入列表。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.906中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
 ├──llm_inference  # 推理代码
   ├──ascend_vllm
       ├── vllm_npu       # 推理源码
       ├── ascend_vllm-0.4.2-py3-none-any.whl   # 推理安装包
       ├── build.sh               # 推理构建脚本
       ├── vllm_install.patch      # 社区昇腾适配的补丁包
 ├──llm_tools          # 推理工具包
   ├──AutoSmoothQuant  # W8A8量化工具
       ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
       ├── autosmoothquant                 # 量化代码
       ├── build.sh                    # 安装量化模块的脚本
   ├──awq            # W4A16量化工具
      ├──convert_awq_to_npu.py   # awq权重转换脚本
   ├──llm_evaluation    # 推理评测代码包
```

```
├──benchmark_tools    #性能评测
    ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
    ├── benchmark_parallel.py   # 评测静态性能脚本
    ├── benchmark_serving.py    # 评测动态性能脚本
    ├── benchmark_utils.py      # 抽离的工具集
    ├── generate_datasets.py    # 生成自定义数据集的脚本
    ├── requirements.txt        # 第三方依赖
├──benchmark_eval #精度评测
    ├──opencompass.sh       #运行opencompass脚本
    ├──start.sh             #安装opencompass脚本
    ├──vllm_api.py          #启动vllm api服务器
    ├──vllm.py              #构造vllm评测配置脚本名字
```

相关文档

和本文档配套的模型训练文档请参考**主流开源大模型基于DevServer适配PyTorch NPU训练指导（6.3.906）**。

3.8.2 部署推理服务

本章节介绍如何使用vLLM 0.4.2框架部署并启动推理服务。

前提条件

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

Step1 检查环境

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

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

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

   驱动版本要求是23.0.5。如果不符合要求请参考**安装固件和驱动**章节升级驱动。

2. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表3-71**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-LLM-6.3.906-xxx.zip和算子包AscendCloud-OPP-6.3.906-xxx.zip到主机中，包获取路径请参见**表3-72**。
2. 将权重文件上传到DevServer机器中。权重文件的格式要求为Huggface格式。开源权重文件获取地址请参见**表3-73**。

   如果使用模型训练后的权重文件进行推理，需要上传训练后的权重文件和开源的原始权重文件。模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step4 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。

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

参数说明：

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

  说明

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

Step5 进入容器安装推理依赖软件

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
```
docker exec -it ${container_name} bash
```
2. 上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```
3. 解压算子包并将相应算子安装到环境中。
```
unzip AscendCloud-OPP-*.zip
pip install ascend_cloud_ops-1.0.0-py3-none-any.whl
pip install cann_ops-1.0.0-py3-none-any.whl
```
4. 解压软件推理代码并安装依赖包。安装过程需要连接互联网git clone，请确保容器环境可以访问公网。
```
unzip AscendCloud-LLM-*.zip
cd llm_inference/ascend_vllm
bash build.sh
```
运行完后，会安装适配昇腾的vllm-0.4.2版本。

Step6 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```
说明

NPU卡编号可以通过命令npu-smi info查询。

2. 配置环境变量。
```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```
3. 如果需要增加模型量化功能，启动推理服务前，先参考**使用AWQ量化**或**使用SmoothQuant量化**章节对模型做量化处理。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **方式一：通过OpenAI服务API接口启动服务**

  在llm_inference/ascend_vllm/vllm-gpu-0.4.2目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

  ```
  python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --dtype=float16 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --host=${docker_ip} \
  --port=8080 \
  --gpu-memory-utilization=0.9 \
  --trust-remote-code
  ```

- **方式二：通过vLLM服务API接口启动服务**

  在llm_inference/ascend_vllm/vllm-gpu-0.4.2目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

  ```
  python -m vllm.entrypoints.api_server --model ${container_model_path} \
  --max-num-seqs=256 \
  --max-model-len=4096 \
  --max-num-batched-tokens=4096 \
  --dtype=float16 \
  --tensor-parallel-size=1 \
  --block-size=128 \
  --host=${docker_ip} \
  --port=8080 \
  --gpu-memory-utilization=0.9 \
  --trust-remote-code
  ```

推理服务基础参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。若使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：kv-cache的block大小，推荐设置为128。当前仅支持64和128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。
- --use-v2-block-manager：vllm启动时使用V2版本的BlockSpaceManger来管理KVCache索引，若不使用该功能，则无需配置。注意：若使用投机推理功能，必须开启此参数。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表3-74**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。${container_model_path}请替换为实际使用的模型名称。

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

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

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

表 3-74 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

参数	是否必选	默认值	参数类型	描述
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	否	False	Bool	ignore_eos表示是否忽略EOS并且继续生成token。

3.8.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

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

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

代码目录如下:

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

目前性能测试还不支持投机推理能力。

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-LLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，切换一个conda环境，执行如下命令安装性能测试的关依赖。
   ```
   conda activate python-3.9.10
   pip install -r requirements.txt
   ```
3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   cd benchmark_tools
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/
   tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv
   benchmark_parallel.csv
   ```
   参数说明
   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。上面命令中使用vllm举例。
   - --host ${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。

- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
- --benchmark-csv：结果保存文件，如benchmark_parallel.csv。

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

图 3-121 静态 benchmark 测试结果（示意图）

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

动态 benchmark

本章节介绍如何进行动态benchmark验证。

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

   方法一：使用公开数据集

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

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

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

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

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。

- --min-input：输入tokens最小长度，可以根据实际需求设置。
- --max-input：输入tokens最大长度，可以根据实际需求设置。
- --avg-input：输入tokens长度平均值，可以根据实际需求设置。
- --std-input：输入tokens长度方差，可以根据实际需求设置。
- --min-output：最小输出tokens长度，可以根据实际需求设置。
- --max-output：最大输出tokens长度，可以根据实际需求设置。
- --avg-output：输出tokens长度平均值，可以根据实际需求设置。
- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

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

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

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

图 3-122 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.006428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

3.8.4 推理精度测试

本章节介绍如何进行推理精度测试，数据集是ceval_gen、mmlu_gen。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-LLM的llm_tools/llm_evaluation目录中，代码目录结构如下。
```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
```
2. 确保容器内通网，未通网需要配置$config_proxy_str，$config_pip_str设置对应的代理和pip源，来确保当前代理和pip源可用。
3. 精度评测新建一个conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。命令中的$work_dir 是benchmark_eval的绝对路径。
```
conda activate python-3.9.10 #如果没有该conda环境需要手动建立一个
export work_dir=${work_dir} #指定work_dir路径
bash install.sh
```
4. 在benchmark_eval目录下安装依赖。
```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下（可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```
5. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤6进行评测。
```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```
6. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。
```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```
参数说明:
   - vllm_path：构造vllm评测配置脚本名字，默认为vllm。
   - service_port：服务端口，与启动服务时的端口保持，比如8080。
   - max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。

- batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
- eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen 。
- model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
- benchmark_type：评测数据集类型，分为eval、static、awq，也就是精度、静态和量化数据集，默认eval。

参考命令：

```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```

7. 这一步可以在客户端显示运行过程，通过run.py运行。如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。

```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
```

output_path: 要保存的结果路径。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（mmlu取值46.6）和GPU打分结果（mmlu取值47）进行对比，误差在1以内（计算公式：(47-46.6) < 1,）认为NPU精度和GPU对齐。

3.8.5 推理模型量化

3.8.5.1 使用 AWQ 量化

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表3-73**。

本章节介绍如何使用AWQ量化工具实现推理量化。

量化方法：per-group

Step1 模型量化

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

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 在容器中使用ma-user用户运行以下命令下载并安装AutoAWQ源码。
```
git clone -b v0.2.5 https://github.com/casper-hansen/AutoAWQ.git AutoAWQ-0.2.5
cd ./AutoAWQ-0.2.5
export PYPI_BUILD=1
pip install -e .
```
2. 需要编辑“examples/quantize.py”文件，针对NPU进行如下适配工作，以支持在NPU上进行量化。
   a. 添加import。
   ```
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```
   b. 指定模型输入、输出路径。
   ```
   model_path = **
   quant_path = **
   ```
   c. 可以指定校准数据集路径，如calib_data="/path/to/pile-val"，如不指定，默认数据集是“mit-han-lab/pile-val-backup”。
   ```
   model.quantize(tokenizer, quant_config=quant_config, calib_data="/path/to/pile-val",
   split="validation")
   ```
3. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。
   ```
   pip install transformers sentencepiece #安装量化工具依赖
   export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
   python examples/quantize.py
   ```
   详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools代码目录下执行以下脚本：

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

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

参数说明：

model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step6 启动推理服务**，在启动服务时添加如下命令。

```
-q awq 或者--quantization awq
```

3.8.5.2 使用 SmoothQuant 量化

SmootQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmootQuant(W8A8)量化的模型列表请参见**表3-73**。

本章节介绍如何使用SmoothQuant量化工具实现推理量化。

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

代码目录如下:

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

具体操作如下：

1. 配置环境。
   ```
   cd llm_tools/AutoSmoothQuant/
   sh build.sh
   ```
2. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```

   说明

   NPU卡编号可以通过命令npu-smi info查询。
3. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```

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

   参考**Step6 启动推理服务**，启动推理服务时添加如下命令。
   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

3.8.5.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表3-73**。

Step1 使用 tensorRT 量化工具进行模型量化

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

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

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

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

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

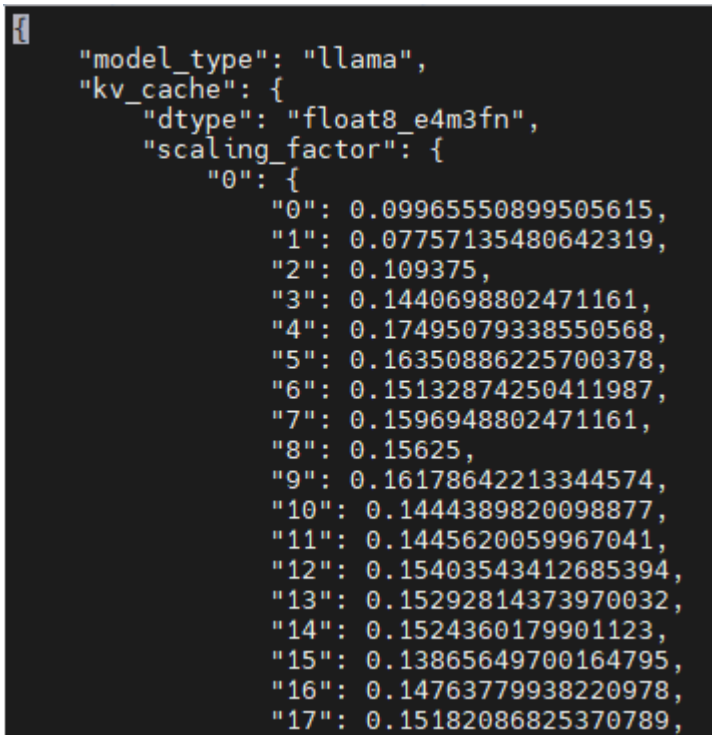
Step2 抽取 kv-cache 量化系数

该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

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

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



注意：

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

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

Step3 启动 kv-cache-int8 量化服务

在使用OpenAI接口或vLLM接口启动推理服务时添加如下参数：

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

3.8.6 附录：大模型推理常见问题

问题1：在推理预测过程中遇到NPU out of memory

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

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

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

3.9 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.906）

3.9.1 场景介绍

方案概览

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

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

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

约束限制

- 如果要使用自动重启功能，资源规格必须选择八卡规格，只有llama3-8B/70B支持该功能。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如表3-75所示。

表 3-75 支持的模型列表

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
2		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
3		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
4	llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
5		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
6	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat

序号	支持模型	支持模型参数量	权重文件获取地址
7		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
8		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
9	Qwen1.5	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
10		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
11		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat
12		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
13	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
14		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
15	ChatGLMv3	glm3-6b	https://huggingface.co/THUDM/chatglm3-6b
16	Baichuan2	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
17	Qwen2	qwen2-0.5b	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
18		qwen2-1.5b	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
19		qwen2-7b	https://huggingface.co/Qwen/Qwen2-7B-Instruct
20		qwen2-72b	https://huggingface.co/Qwen/Qwen2-72B-Instruct
21	GLMv4	glm4-9b	https://huggingface.co/THUDM/glm-4-9b-chat

操作流程

图 3-123 操作流程图

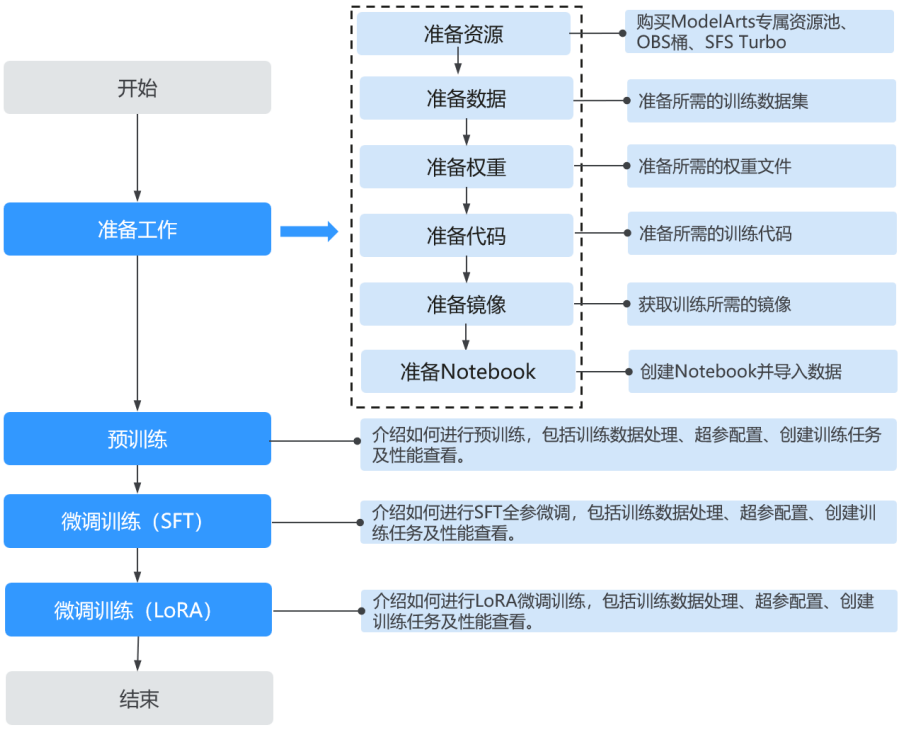


表 3-76 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

3.9.2 准备工作

3.9.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表3-83**。

硬盘空间：至少200GB。

昇腾资源规格：

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

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统，详细操作指导请参考**创建SFS Turbo文件系统**。

图 3-124 创建 SFS Turbo

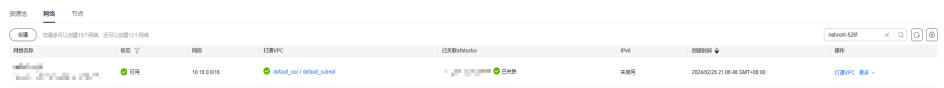
其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 3-125 SFS 类型和容量选择

ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。 具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 3-126 ModelArts 网络关联 SFS Turbo



3.9.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

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

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3.9.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-75**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
```

```
├── tokenizer.model
└── USE_POLICY.md
```

3.9.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表3-77**所示。

表 3-77 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-6.3.906代码包中AscendCloud-LLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
|──llm_train                # 模型训练代码包
        |──AscendSpeed        # 基于AscendSpeed的训练代码
              |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              |──scripts/             # 训练需要的启动脚本
                   |──llama2          # llama2系列模型执行脚本的文件夹
                   |──llama3          # llama3系列模型执行脚本的文件夹
                   |──qwen            # Qwen系列模型执行脚本的文件夹
                   |──qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   |── ...
                   |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   |── install.sh       # 环境部署脚本
              |──src/                 # 启动命令行封装脚本，在install.sh里面自动构建
|──llm_inference              # 推理代码包
|──llm_tools                  # 推理工具
```

代码上传至 OBS

将AscendSpeed代码包AscendCloud-LLM-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed               # 代码目录
          |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
          |──scripts/             # 训练需要的启动脚本
    # 自动生成数据目录结构
    |── processed_for_input          #目录结构会自动生成，无需用户创建
          |── ${model_name}             # 模型名称
                |── data              # 预处理后数据
                |── pretrain    # 预训练加载的数据
```

```
            |── finetune   # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换magatron格式后权重文件
    |── saved_dir_for_output          # 训练输出保存权重，目录结构会自动生成，无需用户创建
        |── ${model_name}             # 模型名称
            |── logs            # 训练过程中日志（loss、吞吐性能）
              |─── saved_models
            |── lora            # lora微调输出权重
            |── sft             # 增量训练输出权重
            |── pretrain        # 预训练输出权重
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                       #微调数据文件
|── model                                          #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
```

3.9.2.5 准备镜像

准备训练Llama2-13B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-78 基础容器镜像地址

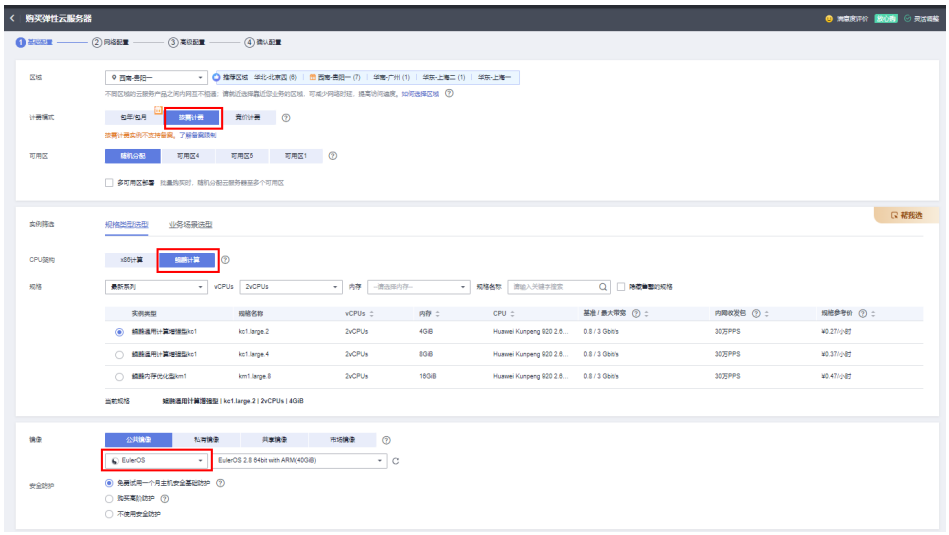
镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 3-127 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

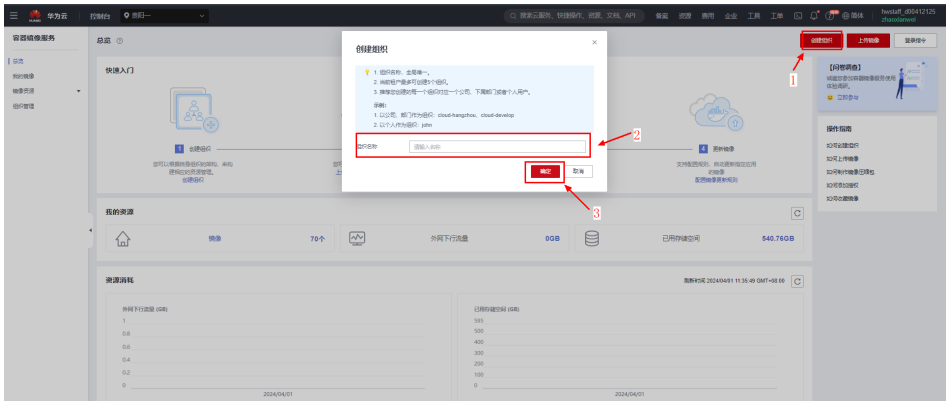
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。

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

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

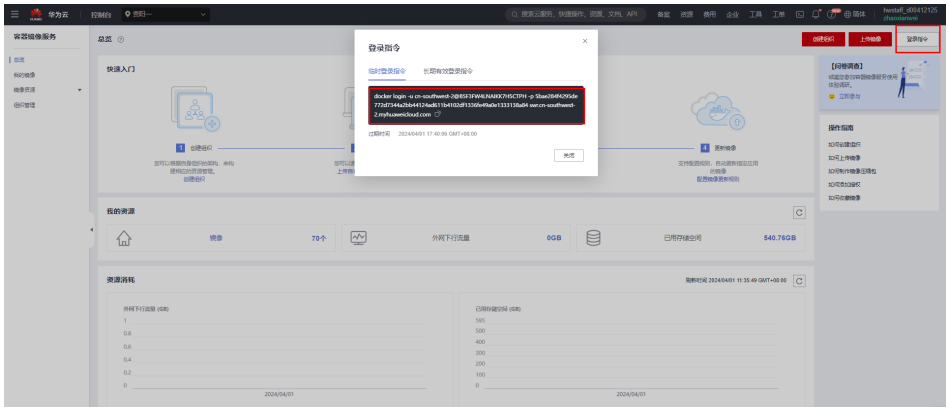
图 3-128 创建镜像组织



Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-129 复制登录指令



Step5 获取训练镜像

请确保在正确的Region下获取镜像。建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见**表3-78**。

```
docker pull {image_url}
```

Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240606

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

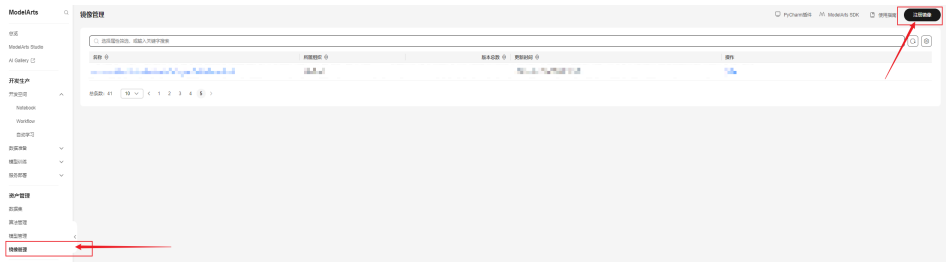
docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/pytorch_2_1_ascend:20240606

Step7 ModelArts 中注册镜像

镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。

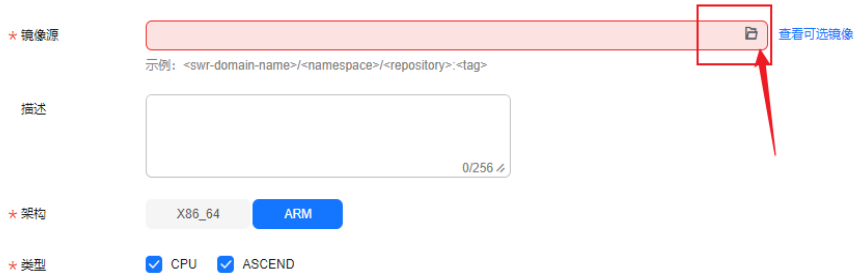
访问ModelArts，在镜像管理中选择注册镜像，如图所示：

图 3-130 注册镜像



选择已上传的镜像源，架构选择ARM，类型勾选CPU和ASCEDN，完成镜像注册。

图 3-131 选择已上传的镜像源



3.9.2.6 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 3-132 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例，子目录挂载可选择默认不填写。

如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 3-133 Notebook 中选择弹性文件服务

使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包，每次启动训练作业时会执行该命令安装。

通过运行install.sh脚本，会git clone下载Megatron-LM、MindSpeed、ModelLink源码（install.sh中会自动下载配套版本，若手动下载源码还需修改版本）至llm_train/AscendSpeed文件夹中。下载的源码文件结构如下：

```
|——AscendCloud-LLM
   |——llm_train                # 模型训练代码包
       |——AscendSpeed          # 基于AscendSpeed的训练代码
           |——ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
           |——scripts/             # 训练需要的启动脚本
           |——src/                 # 启动命令行封装脚本，在install.sh里面自动构建
       |——Megatron-LM/         # 适配昇腾的Megatron-LM训练框架
       |——MindSpeed/           # MindSpeed昇腾大模型加速库
       |——ModelLink/           # ModelLink端到端的大语言模型方案
           |——megatron/     # 注意：该文件夹从Megatron-LM中复制得到
           |——...
```

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

由于训练启动命令也会执行sh scripts/install.sh安装依赖包，因此Notebook保存镜像为可选操作。

图 3-134 安装依赖包

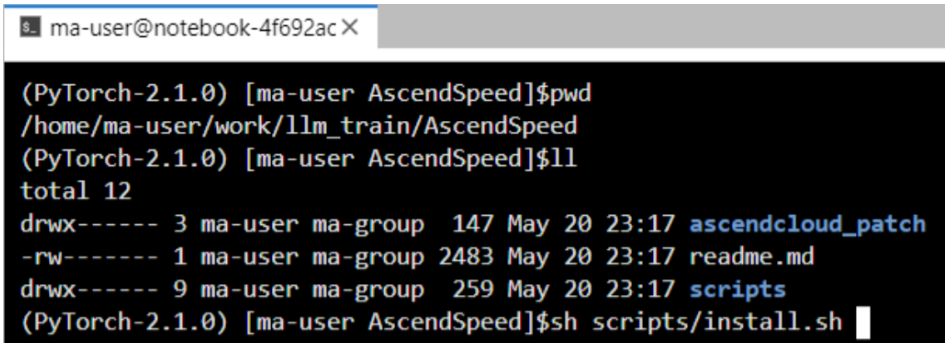


图 3-135 保存镜像

图 3-136 填写保存镜像相关参数

3.9.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-79**所示。其他超参均有默认值，可以参考**表3-82**按照实际需求修改。

表 3-79 必须修改的训练超参配置

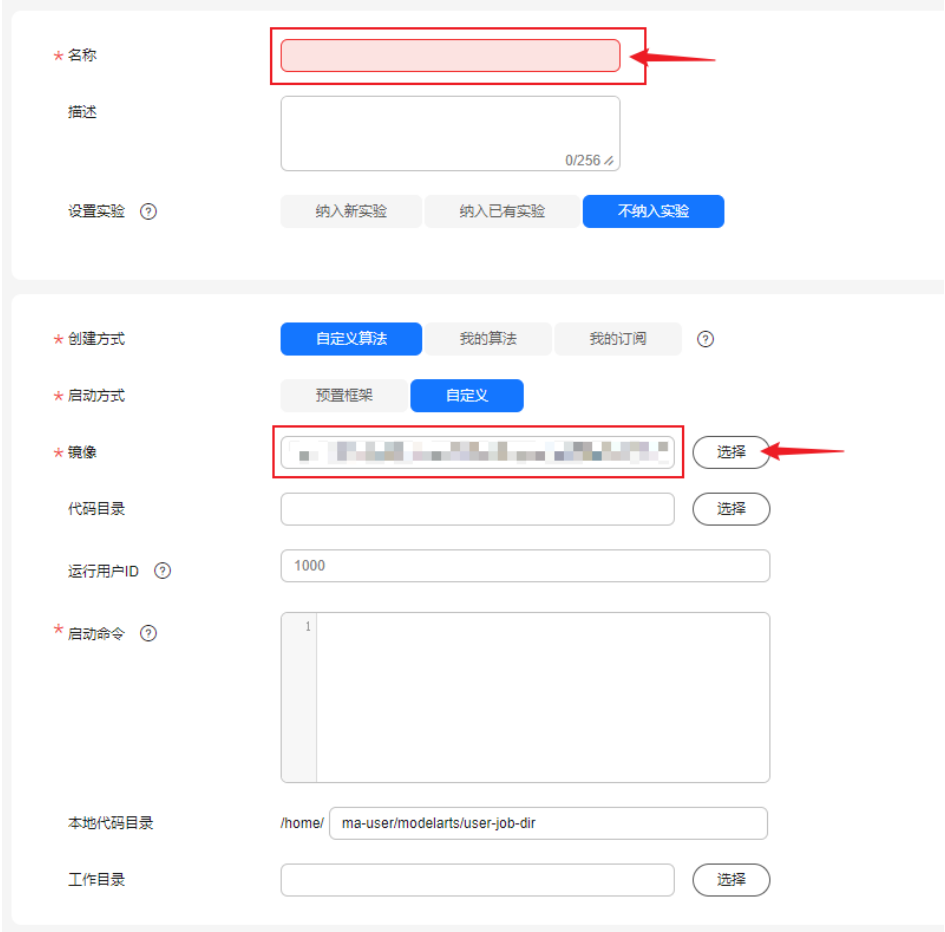
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-137 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

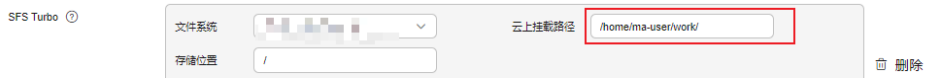
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-83**进行配置。

图 3-138 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-139 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.9.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-80**所示。其他超参均有默认值，可以参考**表3-82**按照实际需求修改。

表 3-80 必须修改的训练超参配置

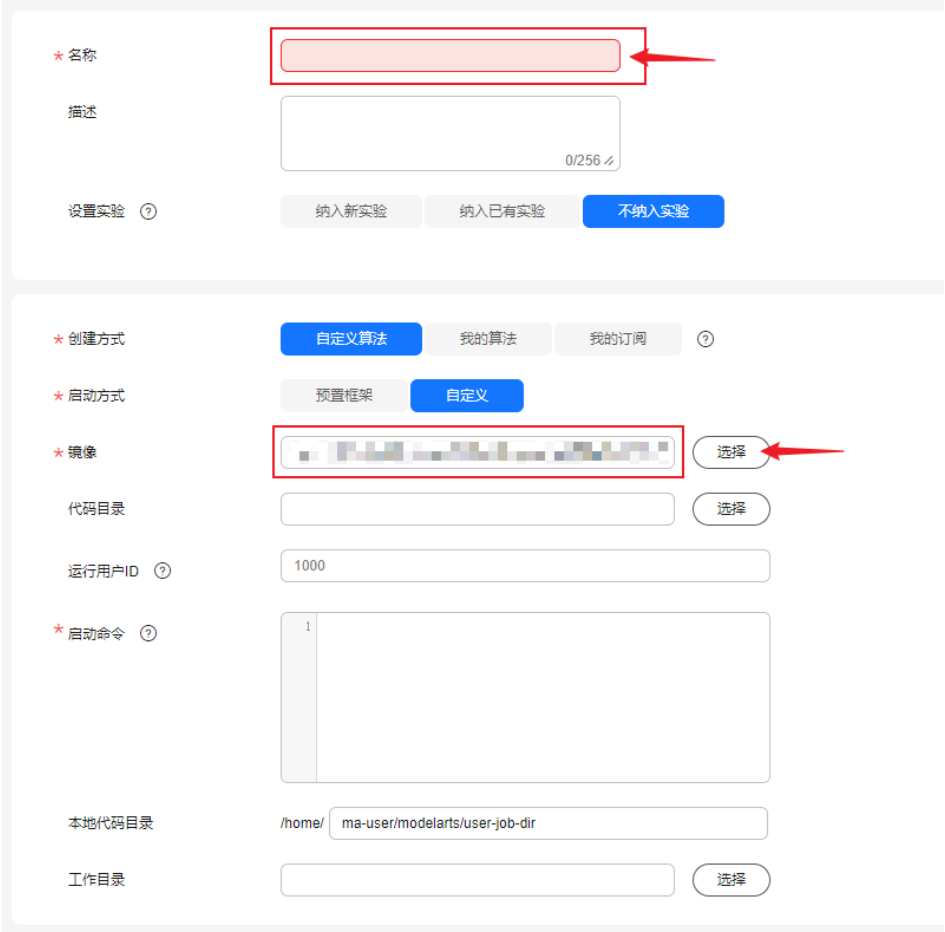
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-140 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-83**进行配置。

图 3-141 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-142 选择 SFS Turbo

作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.9.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-81**所示。其他超参均有默认值，可以参考**表3-82**按照实际需求修改。

表 3-81 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

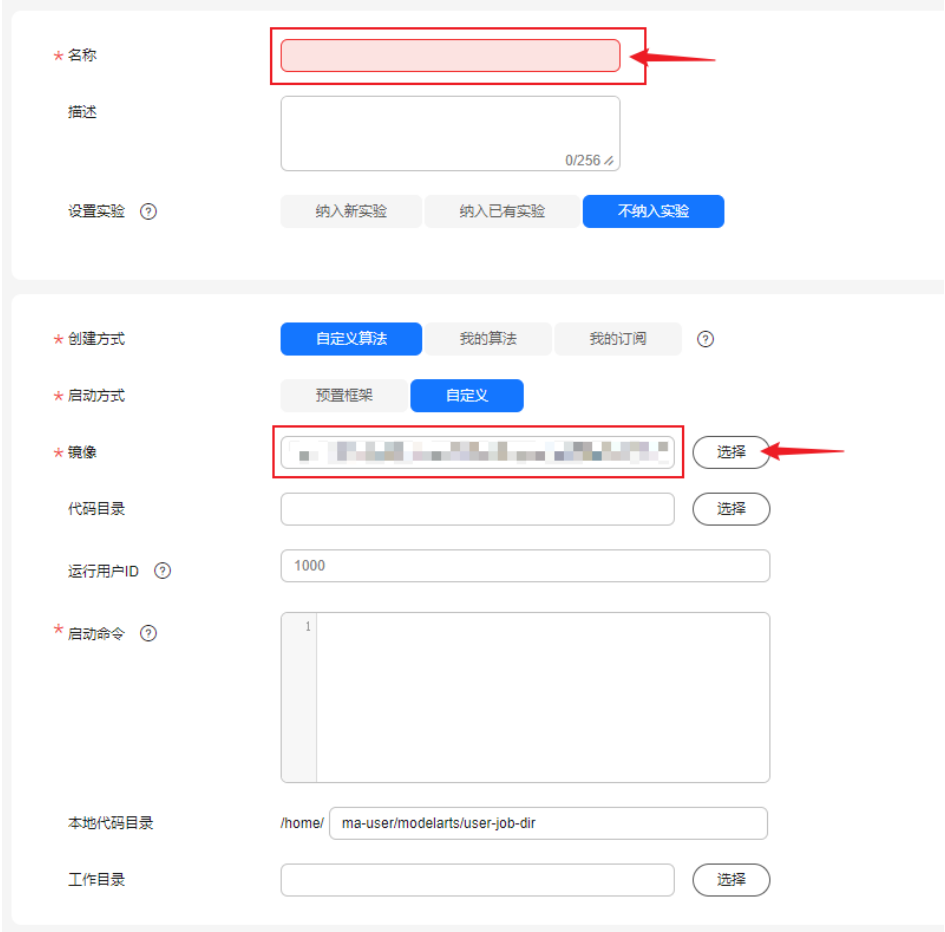
说明

由于模型中LoRA微调训练存在已知的精度问题，因此不支持TP(tensor model parallel size)张量模型并行策略，推荐使用PP(pipeline model parallel size)流水线模型并行策略，具体详细参数配置如**表3-83**所示。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-143 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

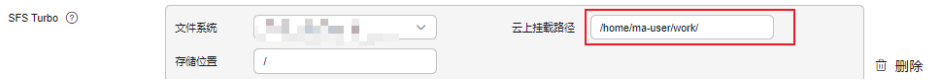
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-83**进行配置。

图 3-144 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-145 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

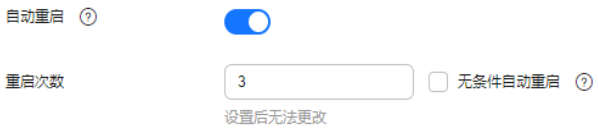
提交训练作业，训练完成后，生成的权重文件自动保存在SFS Turbo中，保存路径为：/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.9.6 开启训练故障自动重启功能

创建训练作业时，可开启自动重启功能。当环境问题导致训练作业异常时，系统将自动修复异常或隔离节点，并重启训练作业，提高训练成功率。为了避免丢失训练进度、浪费算力。此功能已适配断点续训练。

图 3-146 开启故障重启



断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当训练作业发生故障中断本次作业时，代码可自动从训练中断的位置接续训练，加载中断生成的checkpoint，中间不需要改动任何参数（支持预训练、LoRA微调、SFT微调）。

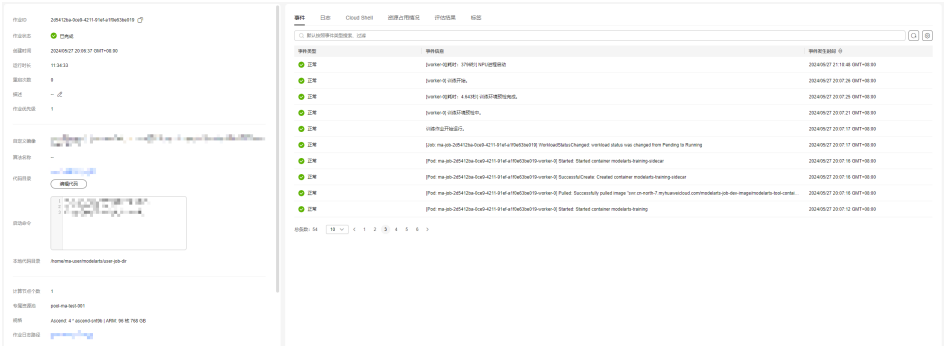
说明

- 如果要使用自动重启功能，资源规格必须选择八卡规格。
- 当前功能还处于试验阶段，只有llama3-8B/70B适配。

3.9.7 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

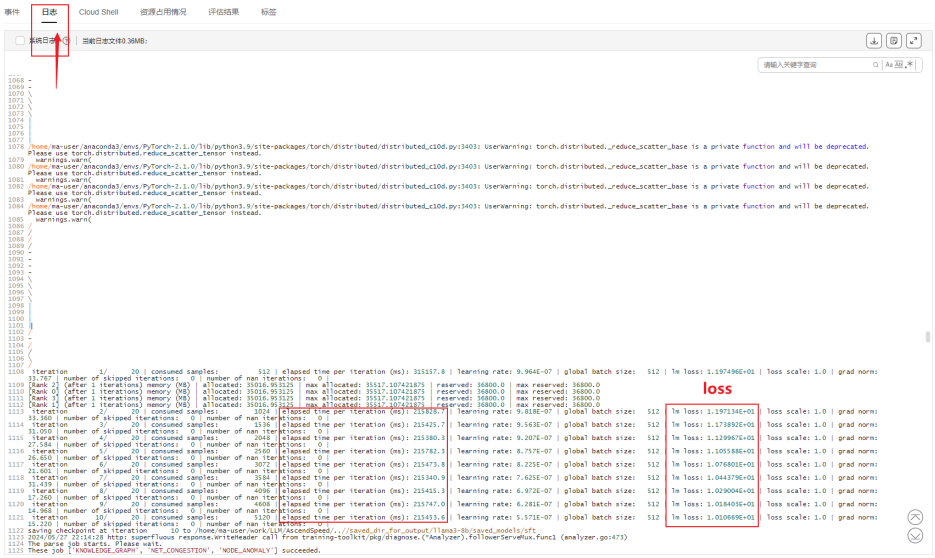
图 3-147 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 3-148 查看日志和性能



3.9.8 训练脚本说明

3.9.8.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 3-82 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler, MOSSMultiTurnHandler]	示例值需要根据数据集的不同，选择其一。 ● GeneralPretrainHandler：使用预训练的alpaca数据集。 ● GeneralInstructionHandler：使用微调的alpaca数据集。 ● MOSSMultiTurnHandler：使用微调的moss数据集。
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。

参数	示例值	参数说明
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如表3-83所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-83 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
4	llama3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=4	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=1 PP(pipeline model parallel size)=4	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
17	Qwen2	qwen2-0.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
18		qwen2-1.5b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=1	1*节点 & 2*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
19		qwen2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
20		qwen2-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
21	GLMv4	glm4-9b	SEQ_LEN=4096	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=2 PP(pipeline model parallel size)=4	1*节点 & 8*Ascend

3.9.8.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。

- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/fintune/**

用户自定义执行数据处理脚本修改参数说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。

- 方法一：用户可打开**scripts/llama2/1_preprocess_data.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-84 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.9.8.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。

- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

注意：权重转换完成后，需要将例如saved_models/pretrain_hf中的文件与原始Hugging Face模型中的文件进行对比，查看是否缺少如tokenizers.json、tokenizer_config.json、special_tokens_map.json等tokenizer文件或者其他json文件。若缺少则需要直接复制至权重转换后的文件夹中，否则不能直接用于推理。

用户自定义执行权重转换参数修改说明

若用户要自定义数据处理脚本并且单独执行，同样以 llama2 为例。注意脚本中的python命令分别有Hugging Face 转 Megatron格式，以及Megatron 转 Hugging Face格式，而脚本使用hf2hg、mg2hf参数传递来区分。

- 方法一：用户可打开**scripts/llama2/2_convert_mg_hf.sh**脚本，将执行的python命令复制下来，修改环境变量的值。在**Notebook**进入到 **/home/ma-user/work/llm_train/AscendSpeed/ModelLink** 路径中，再执行python命令。
- 方法二：用户在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在脚本的首行中添加 **cd /home/ma-user/work/llm_train/AscendSpeed/ModelLink** 命令，随后在**Notebook**中运行该脚本。

其中环境变量详细介绍如下：

表 3-85 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.9.8.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-149所示。

图 3-149 修改 ChatGLMv3-6B tokenizer 文件

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-150所示。

图 3-150 修改 ChatGLMv3-6B tokenizer 文件

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

GLMv4-9B

在训练开始前，针对ChatGLMv4-9B模型中的tokenizer文件，需要修改代码。修改文件chatglm4-9b/tokenization_chatglm.py 。

294行要添加注释，修改后如图3-151所示。

图 3-151 修改 ChatGLMv4-9B tokenizer 文件

```
293             # Load from model defaults
294             #assert self.padding_side == "left"
```

314至323行要修改，修改后如图3-152所示。

图 3-152 修改 ChatGLMv4-9B tokenizer 文件

```
314         if needs_to_be_padded:
315             difference = max_length - len(required_input)
316 
317             if "attention_mask" in encoded_inputs:
318                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
319             if "position_ids" in encoded_inputs:
320                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
321             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
322 
323         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-153所示。

图 3-153 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.10 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.906）

3.10.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

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

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.4.2版本。
- 仅支持FP16和BF16数据类型推理。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持的模型列表、对应的开源权重获取地址如**表3-86**所示。

表 3-86 支持的模型列表和权重获取地址

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
1	llama-7b	√	√	√	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	√	√	√	**https://huggingface.co/huggyllama/llama-13b**

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
3	llama-65b	√	√	√	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	√	√	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	√	√	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	√	√	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	√	√	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	√	√	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	x	x	x	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat

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

序号	模型名称	是否支持fp16/bf16推理	是否支持W4A16量化	是否支持W8A8量化	是否支持kv-cache-int8量化	开源权重获取地址
29	baichuan2-7b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
30	baichuan2-13b	√	x	x	x	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
31	gemmma-2b	√	x	x	x	https://huggingface.co/google/gemma-2b
32	gemmma-7b	√	x	x	x	https://huggingface.co/google/gemma-7b
33	chatglm2-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm2-6b
34	chatglm3-6b	√	x	x	x	https://huggingface.co/THUDM/chatglm3-6b
35	glm-4-9b	√	x	x	x	https://huggingface.co/THUDM/glm-4-9b-chat
36	mistral-7b	√	x	x	x	https://huggingface.co/mistralai/Mistral-7B-v0.1
37	mixtral-8x7b	√	x	x	x	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

操作流程

图 3-154 操作流程图

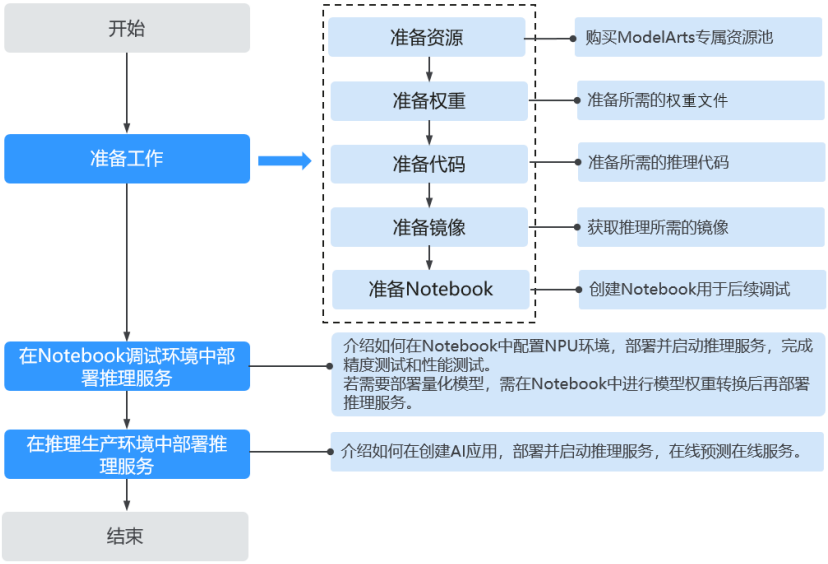


表 3-87 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-6.3.906-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。 若需要部署量化模型，需在Notebook中进行模型权重转换后再部署推理服务。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

3.10.2 准备工作

3.10.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

3.10.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-86**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

3.10.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-88**所示。

表 3-88 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-6.3.906-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的推理部署代码和推理评测代码、推理依赖的算子包。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

本教程需要使用到的AscendCloud-6.3.906中的AscendCloud-LLM-xxx.zip软件包和算子包AscendCloud-OPP，AscendCloud-LLM关键文件介绍如下。

```
├──AscendCloud-LLM
  ├──llm_inference  # 推理代码
    ├──ascend_vllm
        ├── vllm_npu       # 推理源码
        ├── ascend_vllm-0.4.2-py3-none-any.whl   # 推理安装包
        ├── build.sh                # 推理构建脚本
        ├── vllm_install.patch       # 社区昇腾适配的补丁包
  ├──llm_tools           # 推理工具包
    ├──AutoSmoothQuant  # W8A8量化工具
        ├── ascend_autosmoothquant_adapter  # 昇腾量化使用的算子模块
        ├── autosmoothquant                 # 量化代码
        ├── build.sh                        # 安装量化模块的脚本
    ├──awq              # W4A16量化工具
      ├──convert_awq_to_npu.py   # awq权重转换脚本
    ├──llm_evaluation    # 推理评测代码包
      ├──benchmark_tools    #性能评测
        ├── benchmark.py     # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py   # 评测静态性能脚本
        ├── benchmark_serving.py    # 评测动态性能脚本
        ├── benchmark_utils.py      # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt        # 第三方依赖
      ├──benchmark_eval #精度评测
        ├──opencompass.sh       #运行opencompass脚本
        ├──start.sh             #安装opencompass脚本
        ├──vllm_api.py          #启动vllm api服务器
        ├──vllm.py              #构造vllm评测配置脚本名字
```

3.10.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-89 基础容器镜像地址

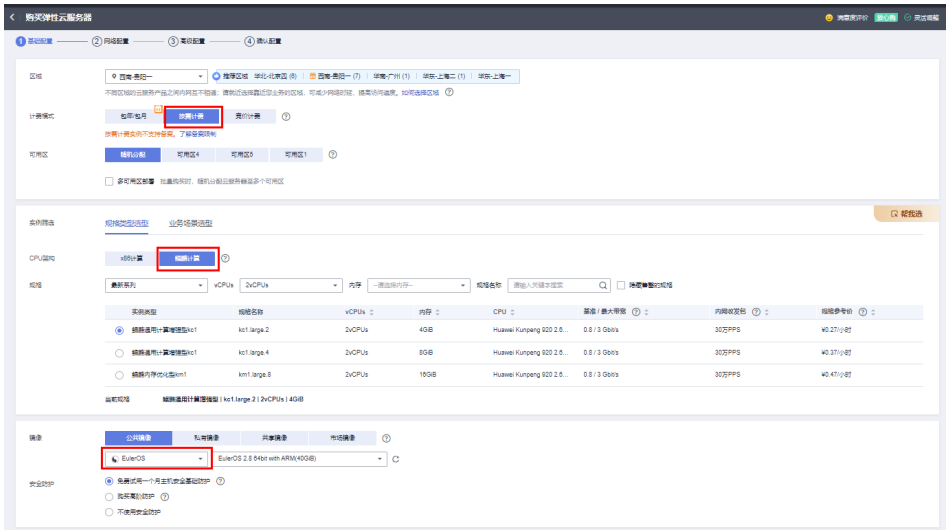
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 3-155 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

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

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

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

图 3-156 创建镜像组织

Step4 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step5 构建 ModelArts Standard 推理镜像

获取模型软件包和依赖包，并上传到ECS的目录下（可自定义路径），获取地址参考**表3-88**。

在ModelArts官方提供的基础镜像上，构建一个用于ModelArts Standard推理部署的镜像。

在模型软件包和依赖包的同层目录下，创建并编辑Dockerfile。

```
vim Dockerfile
```

Dockerfile内容如下：

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

USER ma-user
COPY AscendCloud-*.zip /home/ma-user/
RUN unzip -o /home/ma-user/AscendCloud-*.zip
RUN unzip -o /home/ma-user/AscendCloud-LLM-*.zip
RUN unzip -o /home/ma-user/AscendCloud-OPP-*.zip

RUN pip install /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN cd /home/ma-user/llm_inference/ascend_vllm && bash /home/ma-user/llm_inference/ascend_vllm/build.sh

ENTRYPOINT sh /home/mind/model/run_vllm.sh
```

构建镜像。

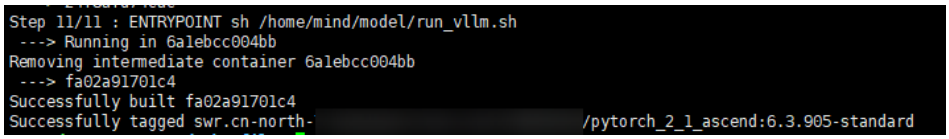
```
docker build -t swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag> .
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示构建镜像成功。

图 3-157 成功构建镜像

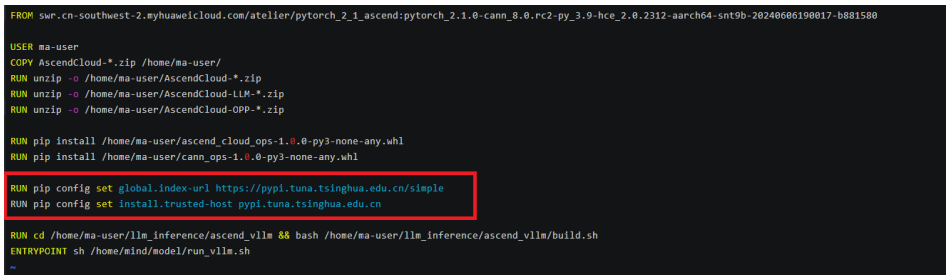


注：若构建镜像时报错pip超时，可在Dockerfile中添加如下命令设置pip源

```
RUN pip config set global.index-url https://xxx/simple
RUN pip config set install.trusted-host xxx
```

如下图所示：

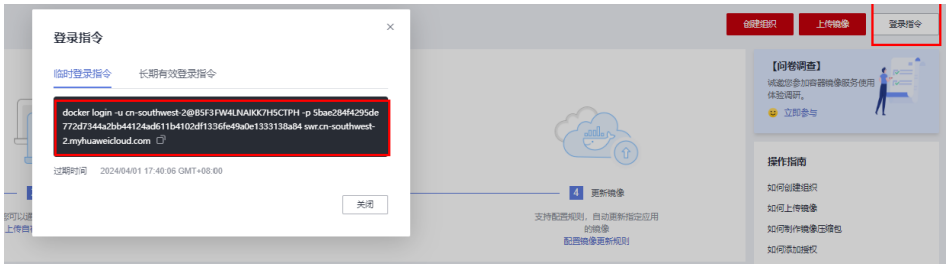
图 3-158 dockerfile 添加 pip 源



Step6 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-159 复制登录指令



Step7 上传镜像

在ECS服务器中输入登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：前面步骤中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

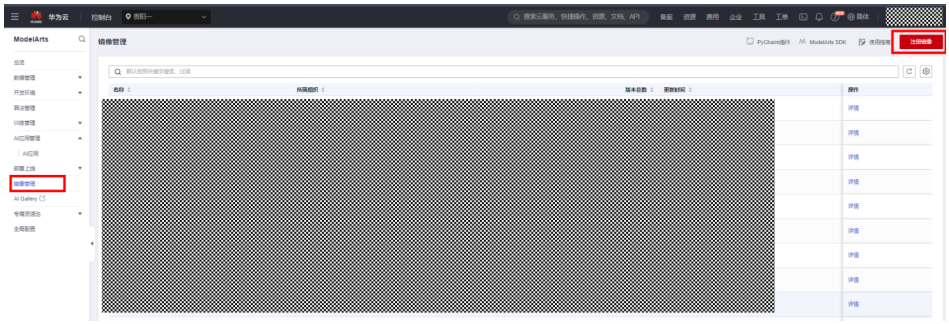
打印如下信息，表示上传镜像成功。

图 3-160 成功上传镜像

Step8 注册镜像

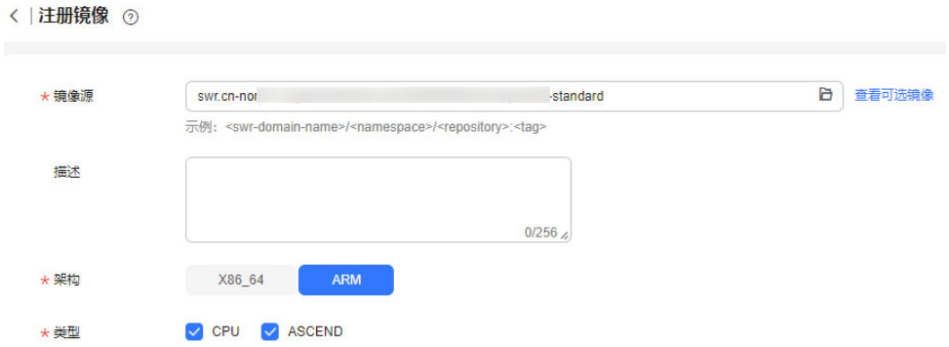
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中点击“注册镜像”。

图 3-161 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 3-162 注册镜像



Step9 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

3.10.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**Notebook使用场景介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，点击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 3-163 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Asecnd snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，可参考后续章节在Notebook调试环境中部署推理服务。

3.10.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

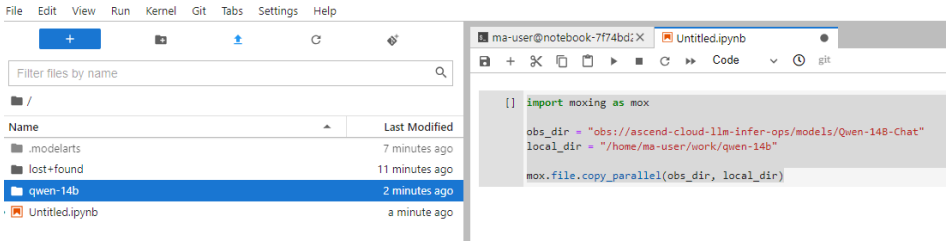
Step2 准备权重文件

将OBS中的模型权重上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 3-164 上传 OBS 文件到 Notebook 的代码示例



Step3 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。

   ```
   export ASCEND_RT_VISIBLE_DEVICES=0
   ```

   如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。

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

说明

NPU卡编号可以通过命令npu-smi info查询。

2. 配置环境变量。

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

3. 如果需要增加模型量化功能，启动推理服务前，先参考**推理模型量化**章节对模型做量化处理。

4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **通过vLLM服务API接口启动服务**

  在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model="${model_path}" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${model_path}" \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。如果使用了量化功能，则使用**推理模型量化**章节转换后的权重。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：/home/ma-user/work/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step4 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

```
curl -X POST http://localhost:8080/generate \
-H "Content-Type: application/json" \
-d '{
        "prompt": "hello",
        "max_tokens": 100,
        "temperature": 0,
        "ignore_eos": false,
        "presence_penalty":2
 }'
```

vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**

- 方式二：使用OpenAI接口请求服务，命令参考如下。

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

表 3-90 请求服务参数说明

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

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

参数	是否必选	默认值	参数类型	描述
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

Step5 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

3.10.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- 模型权重文件获取地址请参见**表3-86**。

  说明

  若需要部署量化模型，请参考**推理模型量化**在Notebook中进行权重转换，并将转换后的权重上传至OBS中。

- 推理启动脚本run_vllm.sh制作请参见**•创建推理脚本文件run_vllm.sh**。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 3-165 准备模型文件和权重文件

- **创建推理脚本文件run_vllm.sh**

  run_vllm.sh脚本示例如下。

  - **通过vLLM服务API接口启动服务**

    ```
    source /home/ma-user/.bashrc
    export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
    python -m vllm.entrypoints.api_server --model="${model_path}" \
    --ssl-keyfile="/home/mind/model/key.pem" \
    --ssl-certfile="/home/mind/model/cert.pem" \
    --max-num-seqs=256 \
    --max-model-len=4096 \
    --max-num-batched-tokens=4096 \
    --dtype=float16 \
    --tensor-parallel-size=1 \
    --block-size=128 \
    --host=0.0.0.0 \
    --port=8080 \
    --gpu-memory-utilization=0.9 \
    --trust-remote-code
    ```

  - **通过OpenAI服务API接口启动服务**

    ```
    source /home/ma-user/.bashrc
    export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
    python -m vllm.entrypoints.openai.api_server --model="${model_path}" \
    --ssl-keyfile="/home/mind/model/key.pem" \
    --ssl-certfile="/home/mind/model/cert.pem" \
    --max-num-seqs=256 \
    --max-model-len=4096 \
    --max-num-batched-tokens=4096 \
    --dtype=float16 \
    --tensor-parallel-size=1 \
    --block-size=128 \
    --host=0.0.0.0 \
    --port=8080 \
    --gpu-memory-utilization=0.9 \
    --trust-remote-code
    ```

    参数说明：

    - ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
    - ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：/home/mind/model/chatglm3-6b。
    - --tensor-parallel-size：并行卡数。
    - --hostname：服务部署的IP，使用本机IP 0.0.0.0。

- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

高阶参数说明：

- --enable-prefix-caching：如果prompt的公共前缀较长或者多轮对话场景下推荐使用prefix-caching特性。在推理服务启动脚本中添加此参数表示使用，不添加表示不使用。
- --quantization：推理量化参数。当使用量化功能，则在推理服务启动脚本中增加该参数，若未使用量化功能，则无需配置。根据使用的量化方式配置，可选择**awq**或**smoothquant**方式。
- --speculative-model ${container_draft_model_path}：投机草稿模型地址，模型格式是HuggingFace的目录格式。即**Step2 准备权重文件**上传的HuggingFace权重文件存放目录。投机草稿模型为与--model入参同系列，但是权重参数远小于--model指定的模型。若未使用投机推理功能，则无需配置。
- --num-speculative-tokens：投机推理小模型每次推理的token数。若未使用投机推理功能，则无需配置。参数--num-speculative-tokens需要和--speculative-model ${container_draft_model_path}同时使用。

可在run_vllm.sh增加如下环境变量开启高阶配置：

```
export DEFER_DECODE=1
# 是否使用推理与Token解码并行；默认值为1表示开启并行，取值为0表示关闭并行。开启该功能会略微增加首Token时间，但可以提升推理吞吐量。

export DEFER_MS=10
# 延迟解码时间，默认值为10，单位为ms。将Token解码延迟进行的毫秒数，使得当次Token解码能与下一次模型推理并行计算，从而减少总推理时延。该参数需要设置环境变量DEFER_DECODE=1才能生效。

export USE_VOCAB_PARALLEL=1
# 是否使用词表并行；默认值为1表示开启并行，取值为0表示关闭并行。对于词表较小的模型（如llama2系模型），关闭并行可以减少推理时延，对于词表较大的模型（如qwen系模型），开启并行可以减少显存占用，以提升推理吞吐量。

export USE_PFA_HIGH_PRECISION_MODE=1
```

```
# PFA算子是否使用高精度模式；默认值为0表示不开启。针对Qwen2-7B模型，必须开启此配置，否则精度会异常；其他模型不建议开启，因为性能会有损失。
```

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 3-166 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 3-167 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

图 3-168 创建完成

说明

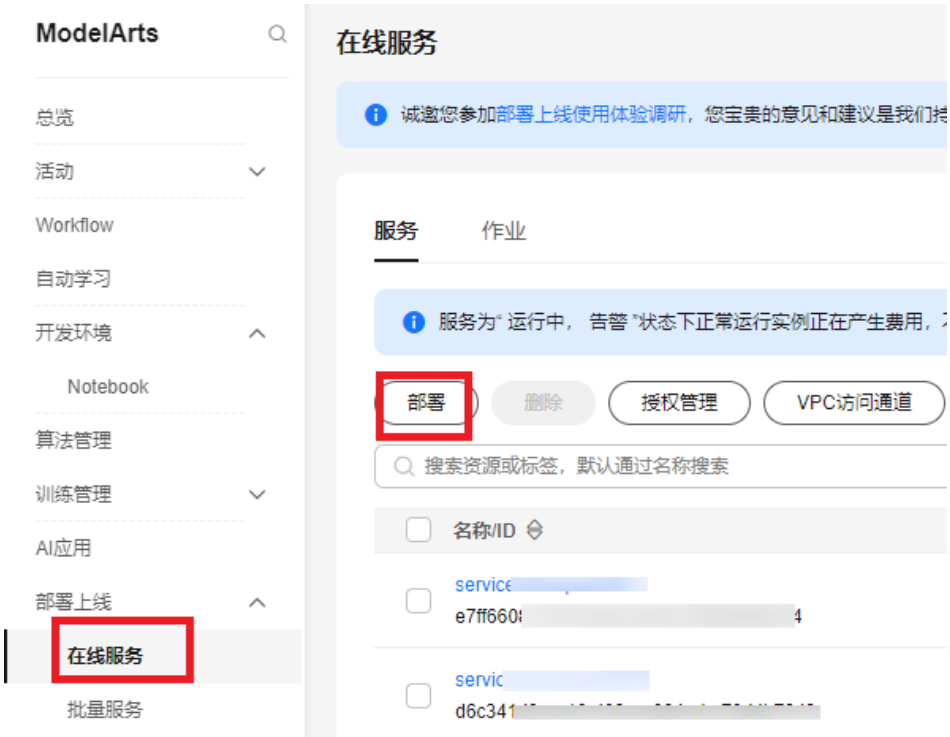
若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台中，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 3-169 部署在线服务



2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 3-170 部署在线服务-专属资源池

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 3-171 服务部署完成

Step4 调用在线服务

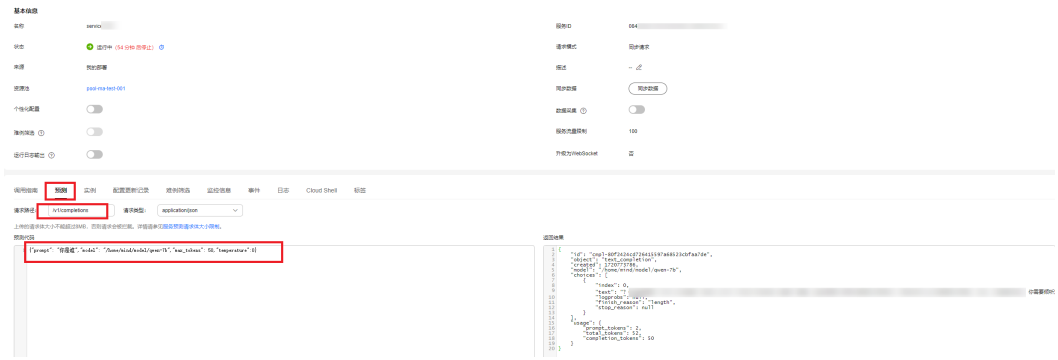
进入在线服务详情页面，选择“预测”。

若以vllm接口启动服务，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”既可看到预测结果。

图 3-172 预测-vllm

若以openai接口启动服务，设置请求路径：“/v1/completions”，输入预测代码“{"prompt": "你是谁","model": "${model_path}","max_tokens": 50,"temperature":0}”，单击“预测”既可看到预测结果。

图 3-173 预测-openai



在线服务的更多内容介绍请参见文档**查看服务详情**。

Step5 推理性能测试

推理性能测试操作请参见**推理性能测试**。

3.10.5 推理精度测试

本章节介绍如何进行推理精度测试，请在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。

Step1 配置精度测试环境

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

```
benchmark_eval
├──opencompass.sh       #运行opencompass脚本
├──install.sh           #安装opencompass脚本
├──vllm_api.py          #启动vllm api服务器
├──vllm.py              #构造vllm评测配置脚本名字
```

2. 确保Notebook内通网，已通网可以跳过这一步，未通网需要配置$config_proxy_str，$config_pip_str设置对应的代理和pip源，来确保当前代理和pip源可用。

3. 精度评测新建一个conda环境，确保之前启动服务为vllm接口，进入到benchmark_eval目录下，执行如下命令。命令中的$work_dir 是benchmark_eval的绝对路径。
```
conda activate python-3.9.10 #如果没有该conda环境需要手动建立一个
export work_dir=${work_dir} #指定work_dir路径
bash install.sh
```
4. 在benchmark_eval目录下安装依赖。
```
cd opencompass #在benchmark_eval目录下
pip install -e . #下载对应依赖
cd ../human-eval #在benchmark_eval目录下（可选，如果选择使用humaneval数据集）
pip install -e .  # 可选，如果选择使用humaneval数据集
```
5. （可选）如果需要在humaneval数据集上评估模型代码能力，请执行此步骤，否则忽略这一步。原因是通过opencompass使用humaneval数据集时，需要执行模型生成的代码。请仔细阅读human_eval/execution.py文件第48-57行的注释，内容参考如下。了解执行模型生成代码可能存在的风险，如果接受这些风险，请取消第58行的注释，执行下面步骤**6**进行评测。
```
# WARNING
# This program exists to execute untrusted model-generated code. Although
# it is highly unlikely that model-generated code will do something overtly
# malicious in response to this test suite, model-generated code may act
# destructively due to a lack of model capability or alignment.
# Users are strongly encouraged to sandbox this evaluation suite so that it
# does not perform destructive actions on their host or network. For more
# information on how OpenAI sandboxes its code, see the accompanying paper.
# Once you have read this disclaimer and taken appropriate precautions,
# uncomment the following line and proceed at your own risk:
#                     exec(check_program, exec_globals)      #第58行
```
6. 执行精度测试启动脚本opencompass.sh，具体操作命令如下，可以根据参数说明修改参数。请确保${work_dir} 已经通过export设置。
```
vllm_path=${vllm_path} \
service_port=${service_port} \
max_out_len=${max_out_len} \
batch_size=${batch_size} \
eval_datasets=${eval_datasets}  \
model_name=${model_name} \
benchmark_type=${benchmark_type} \
bash -x opencompass.sh
```
参数说明:
   - vllm_path：构造vllm评测配置脚本名字，默认为vllm。
   - service_port：服务端口，与启动服务时的端口保持，比如8080。
   - max_out_len：在运行类似mmlu、ceval等判别式回答时，max_out_len建议设置小一些，比如16。在运行human_eval等生成式回答（生成式回答是对整体进行评测，少一个字符就可能会导致判断错误）时，max_out_len设置建议长一些，比如512，至少包含第一个回答的全部字段。
   - batch_size：输入的batch_size大小，不影响精度，只影响得到结果速度。
   - eval_datasets：评测数据集和评测方法，比如ceval_gen、mmlu_gen 。
   - model_name：评测模型名称，不需要与启动服务时的模型参数保持一致。
   - benchmark_type：评测数据集类型，分为eval、static、awq，也就是精度、静态和量化数据集，默认eval。

   参考命令：
```
vllm_path=vllm service_port=8080 max_out_len=16 batch_size=2 eval_datasets=mmlu_gen
model_name=llama_7b  benchmark_type=eval bash -x opencompass.sh
```
7. 客户端显示运行过程，通过run.py运行。如果同时运行多个数据集，需要将不同数据集通过空格分开，加入到eval_datasets中，比如eval_datasets=ceval_gen mmlu_gen。运行命令如下所示。
```
cd opencompass
python run.py --models vllm --datasets mmlu_gen ceval_gen -w ${output_path}
```


output_path: 要保存的结果路径。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{model_name}/的目录结果保存到对应的测试工程。执行多少次，则会在{model_name}下生成多少次结果。benchmark_eval下生成的log中记录了客户端产生结果。数据集的打分结果在result/{model_name}/...目录下，查找到summmary目录，有txt和csv两种保存格式。总体打分结果参考txt和csv文件的最后一行，举例如下：

npu：

mmlu：46.6

gpu：

mmlu：47

NPU打分结果（ mmlu取值46.6 ）和GPU打分结果（ mmlu取值47 ）进行对比，误差在1%以内（ 计算公式：(47-46.6)/47*100=0.85% ）认为NPU精度和GPU对齐。

3.10.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

约束限制

- 创建在线服务时，每秒服务流量限制默认为100次，若静态benchmark的并发数（ parallel-num参数 ）或动态benchmark的请求频率（ request-rate参数 ）较高，会触发推理平台的流控，请在ModelArts Standard“在线服务”详情页修改服务流量限制。
- 同步请求时，平台每次请求预测的时间不能超过60秒。例如输出数据比较大的调用请求（ 例如输出大于1k ），请求预测会超过60秒导致调用失败，可提交工单设置请求超时时间。

benchmark 方法介绍

性能benchmark包括两部分。

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

性能benchmark验证使用到的脚本存放在代码包AscendCloud-LLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
```

```
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本
```

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

Notebook中进行测试：

```
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

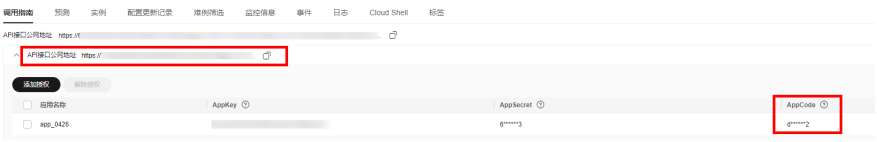
生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

  图 3-174 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
- --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

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

图 3-175 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57295766
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.72861049
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

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

   公开数据集下载地址：

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

   使用generate_datasets.py脚本生成数据集方法：

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

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

   generate_datasets.py脚本执行参数说明如下：

   - --datasets：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。

- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

   Notebook中进行测试：

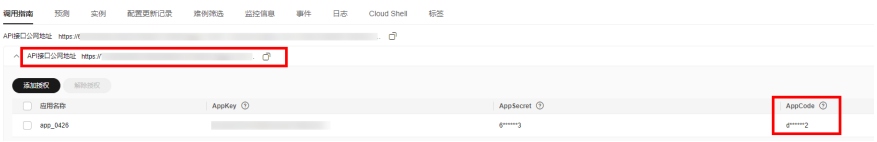
   ```
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：若以vllm接口方式启动服务，API接口公网地址与"/generate"拼接而成；若以openai接口方式启动服务，API接口公网地址与"/v1/completions"拼接而成。部署成功后的在线服务详情页中可查看API接口公网地址。

     图 3-176 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --datasets：数据集路径。
   - --datasets-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为本地模型权重路径。
   - --served-model-name：仅在以openai接口启动服务时需要该参数。若服务部署在Notebook中，该参数为Notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中的${model_path}。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
   - --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
   - --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
   - --benchmark-csv：结果保存路径，如benchmark_serving.csv。

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

图 3-177 动态 benchmark 测试结果（示意图）

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

3.10.7 推理模型量化

3.10.7.1 使用 AWQ 量化工具转换权重

AWQ(W4A16)量化方案能显著降低模型显存以及需要部署的卡数。降低小batch下的增量推理时延。支持AWQ量化的模型列表请参见**表3-86**。

本章节介绍如何在Notebook使用AWQ量化工具实现推理量化，量化方法为per-group。

Step1 模型量化

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

方式一：从开源社区下载发布的AWQ量化模型。

https://huggingface.co/models?sort=trending&search=QWEN+AWQ

方式二：使用AutoAWQ量化工具进行量化。

1. 在Notebook中运行以下命令下载并安装AutoAWQ源码。
   ```
   git clone -b v0.2.5 https://github.com/casper-hansen/AutoAWQ.git AutoAWQ-0.2.5
   cd ./AutoAWQ-0.2.5
   export PYPI_BUILD=1
   pip install -e .
   ```
2. 需要编辑“examples/quantize.py”文件，针对NPU进行如下适配工作，以支持在NPU上进行量化。
   a. 添加import。
      ```
      import torch_npu
      from torch_npu.contrib import transfer_to_npu
      ```
   b. 指定模型输入、输出路径。
      ```
      model_path = **
      quant_path = **
      ```
   c. 可以指定校准数据集路径，如calib_data="/path/to/pile-val"，如不指定，默认数据集是“mit-han-lab/pile-val-backup”。
      ```
      model.quantize(tokenizer, quant_config=quant_config, calib_data="/path/to/pile-val",
      split="validation")
      ```
3. 运行“examples/quantize.py”文件进行模型量化，量化时间和模型大小有关，预计30分钟~3小时。
   ```
   pip install transformers sentencepiece #安装量化工具依赖
   export ASCEND_RT_VISIBLE_DEVICES=0 #设置使用NPU单卡执行模型量化
   python examples/quantize.py
   ```
   详细说明可以参考vLLM官网：**https://docs.vllm.ai/en/latest/quantization/auto_awq.html**。

Step2 权重格式转换

AutoAWQ量化完成后，使用int32对int4的权重进行打包。昇腾上使用int8对权重进行打包，需要进行权重转换。

进入llm_tools代码目录下执行以下脚本：

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

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

参数说明：

--model：模型路径。

Step3 启动 AWQ 量化服务

参考**Step3 启动推理服务**，在启动服务时添加如下命令。

```
--q awq 或者--quantization awq
```

3.10.7.2 使用 SmoothQuant 量化工具转换权重

SmootQuant(W8A8)量化方案能降低模型显存以及需要部署的卡数。也能同时降低首token时延和增量推理时延。支持SmootQuant(W8A8)量化的模型列表请参见**表3-86**。

本章节介绍如何在Notebook使用SmoothQuant量化工具实现推理量化。

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

代码目录如下:

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

具体操作如下：

1. 配置环境。
   ```
   cd llm_tools/AutoSmoothQuant/
   sh build.sh
   ```
2. 配置需要使用的NPU卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
   ```
   export ASCEND_RT_VISIBLE_DEVICES=0,1
   ```
   说明

   NPU卡编号可以通过命令npu-smi info查询。
3. 执行权重转换。
   ```
   cd autosmoothquant/examples/
   python smoothquant_model.py --model-path /home/ma-user/llama-2-7b/ --quantize-model --generate-scale --dataset-path /data/nfs/user/val.jsonl --scale-output scales/llama2-7b.pt --model-output quantized_model/llama2-7b --per-token --per-channel
   ```
   参数说明:
   - --model-path：原始模型权重路径。
   - --quantize-model：体现此参数表示会生成量化模型权重。不需要生成量化模型权重时，不体现此参数
   - --generate-scale：体现此参数表示会生成量化系数，生成后的系数保存在--scale-output参数指定的路径下。如果有指定的量化系数，则不需此参数，直接读取--scale-input参数指定的量化系数输入路径即可。

- --dataset-path：数据集路径，推荐使用：https://huggingface.co/datasets/mit-han-lab/pile-val-backup/resolve/main/val.jsonl.zst。
- --scale-output：量化系数保存路径。
- --scale-input：量化系数输入路径，若之前已生成过量化系数，则可指定该参数，跳过生成scale的过程。
- --model-output：量化模型权重保存路径。
- --smooth-strength：平滑系数，推荐先指定为0.5，后续可以根据推理效果进行调整。
- --per-token：激活值量化方法，若指定则为per-token粒度量化，否则为per-tensor粒度量化。
- --per-channel：权重量化方法，若指定则为per-channel粒度量化，否则为per-tensor粒度量化。

4. 启动smoothQuant量化服务。

   参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

   ```
   -q smoothquant 或者 --quantization smoothquant
   ```

3.10.7.3 使用 kv-cache-int8 量化

kv-cache-int8是实验特性，在部分场景下性能可能会劣于非量化。当前支持per-tensor静态量化，支持kv-cache-int8量化和FP16、BF16、AWQ、smoothquant的组合。

kv-cache-int8量化支持的模型请参见**表3-86**。

本章节介绍如何在Notebook使用tensorRT量化工具实现推理量化。

Step1 使用 tensorRT 量化工具进行模型量化

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

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

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

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

Step2 抽取 kv-cache 量化系数

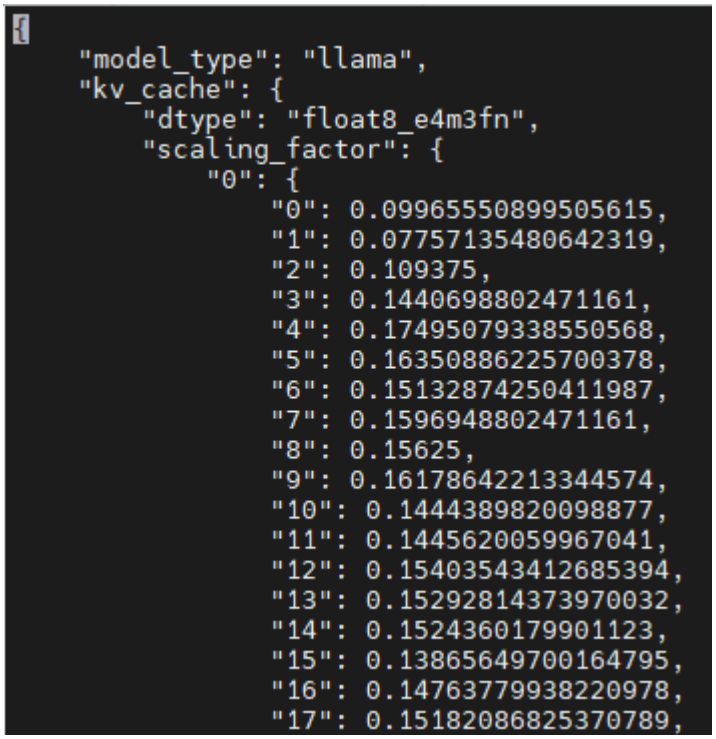
该步骤的目的是将**Step1使用tensorRT量化工具进行模型量化**中生成的scale系数提取到单独文件中，供推理时使用。

使用的抽取脚本由vllm社区提供：

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

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

图 3-178 抽取 kv-cache 量化系数



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


注意：

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

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

Step3 启动 kv-cache-int8 量化服务

参考**Step3 启动推理服务**，启动推理服务时添加如下命令。

```
--kv-cache-dtype int8  #只支持int8，表示kvint8量化
--quantization-param-path kv_cache_scales.json  #输入Step2 抽取kv-cache量化系数生成的json文件路径; 如果只测试推理功能和性能，不需要此json文件，此时scale系数默认为1，但是可能会造成精度下降。
```

3.11 主流开源大模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.905）

3.11.1 场景介绍

方案概览

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

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

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考表3-93获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 本文档中的模型运行环境是ModelArts Lite DevServer。
- 镜像适配的Cann版本是cann_8.0.rc2。
- 确保容器可以访问公网。

训练支持的模型列表

本方案支持以下模型的训练，如表3-91所示。

表 3-91 支持的模型

序号	支持模型	支持模型参数量
1	llama2	llama2-7b
2		llama2-13b
3		llama2-70b
4	llama3	llama3-8b
5		llama3-70b
6	Qwen	qwen-7b
7		qwen-14b
8		qwen-72b
9	Qwen1.5	qwen1.5-7b
10		qwen1.5-14b
11		qwen1.5-32b
12		qwen1.5-72b
13	Yi	yi-6b
14		yi-34b
15	ChatGLMv3	glm3-6b
16	Baichuan2	baichuan2-13b

操作流程

图 3-179 操作流程图

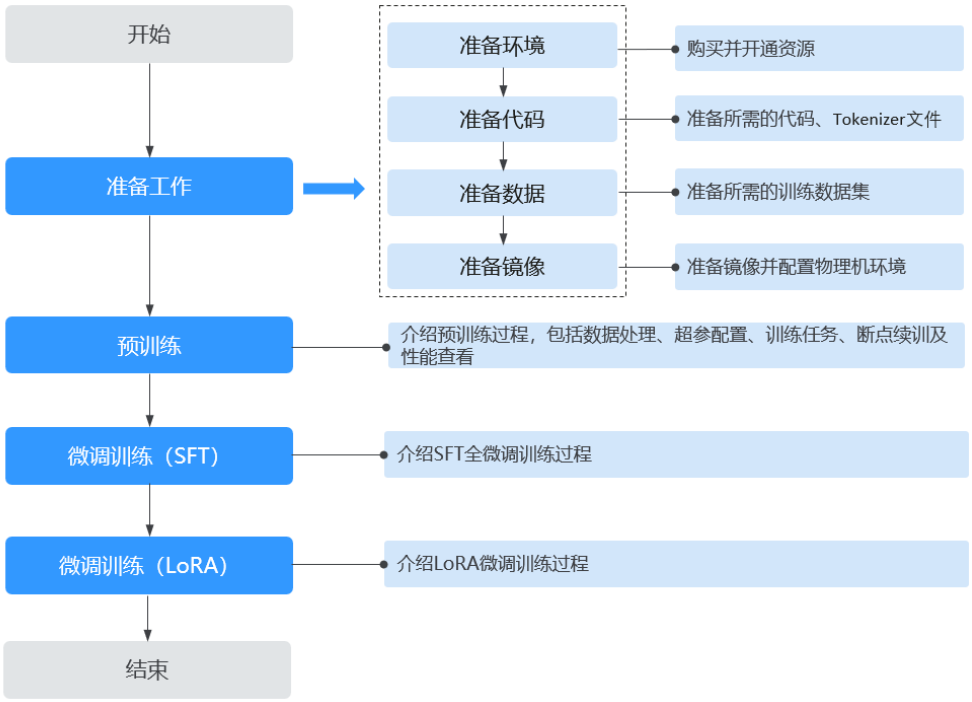


表 3-92 操作任务流程说明

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

3.11.2 准备工作

3.11.2.1 准备环境

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

资源规格要求

计算规格：不同模型训练推荐的NPU卡数请参见**表3-101**。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.11.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取模型软件包和权重文件

本方案支持的模型对应的软件和依赖包获取地址如**表3-93**所示，模型列表、对应的开源权重获取地址如**表3-94**所示。

表 3-93 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

表 3-94 支持的模型类型和权重获取地址

序号	支持模型	支持模型参数量	权重文件获取地址
1	llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**

模型软件包结构说明

AscendCloud-3rdLLM代码包结构介绍如下：

```
|──llm_train                # 模型训练代码包
        |──AscendSpeed         # 基于AscendSpeed的训练代码
             |──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
             |──scripts/             # 训练需要的启动脚本
                 |──llama2           # llama2系列模型执行脚本的文件夹
                 |──llama3           # llama3系列模型执行脚本的文件夹
                 |──qwen             # Qwen系列模型执行脚本的文件夹
                 |──qwen1.5          # Qwen1.5系列模型执行脚本的文件夹
                 |── ...
                 |── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                 |── install.sh       # 环境部署脚本
|──llm_inference                    # 推理代码包
|──llm_tools                        # 推理工具
```

工作目录介绍

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

```
${workdir}（例如/home/ma-user/ws ）
|──llm_train                    #解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
         |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
         |──scripts/             # 各模型训练需要的启动脚本，训练脚本以分类的方式集中在scripts文件夹中。
# 数据目录结构
|── processed_for_input          #目录结构会自动生成，无需用户创建
    |── ${model_name}             # 模型名称
        |── data                # 预处理后数据
          |── pretrain    # 预训练加载的数据
          |── finetune    # 微调加载的数据
        |──converted_weights  # HuggingFace格式转换magatron格式后权重文件
|── saved_dir_for_output          # 训练输出保存权重，目录结构会自动生成，无需用户创建
    |── ${model_name}             # 模型名称
        |── logs              # 训练过程中日志（loss、吞吐性能）
            |── saved_models
          |── lora              # lora微调输出权重
          |── sft               # 增量训练输出权重
          |── pretrain          # 预训练输出权重
|── tokenizers                   #原始权重及tokenizer目录，需要用户手动创建，后续操作步骤中会提示
    |── Llama2-70B
|── training_data                #原始数据目录，需要用户手动创建，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
    |── alpaca_gpt4_data.json     #微调数据文件
```

上传代码和权重文件到工作环境

1. 使用root用户以SSH的方式登录DevServer。
2. 将AscendCloud代码包AscendCloud-3rdLLM-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例，请根据实际修改。

   ```
   unzip AscendCloud-3rdLLM-*.zip
   ```

3. 上传代码之后需要修改llm_train/AscendSpeed/scripts/install.sh文件。具体为删除install.sh 的第43行 "git cherrypick 171ba0b3"。该问题会导致代码安装失败，会在后续版本修复。
4. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

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

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

3.11.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据集

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

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

自定义数据

用户也可以自行准备训练数据。数据要求如下：

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

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

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws ）
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
        |── alpaca_gpt4_data.json                          # 微调数据文件
   ```

3.11.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的训练和推理的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-95 基础容器镜像地址

镜像用途	镜像地址
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0

表 3-96 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**获取。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   export work_dir="自定义挂载的工作目录"     #容器内挂载的目录，例如/home/ma-user/ws
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称"
   docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
   ```

```
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
--cpus 192 \
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载/home/ma-user目录，此目录为ma-user用户家目录。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。
- --shm-size：表示共享内存，用于多进程间通信。由于需要转换较大内存的模型文件，因此大小要求200g及以上。

2. 通过容器名称进入容器中。启动容器时默认用户为ma-user用户。
   ```
   docker exec -it ${container_name} bash
   ```
3. 上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
4. 使用ma-user用户安装依赖包。
   ```
   #进入scripts目录换
   cd /home/ma-user/ws/llm_train/AscendSpeed
   #执行安装命令
   sh scripts/install.sh
   ```

3.11.3 预训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以 **llama2-70b 和 llama2-13b 预训练** 为例，执行脚本为 **0_pl_pretrain_70b.sh** 和 **0_pl_pretrain_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-97**所示。其他超参均有默认值，可以参考**表3-100**按照实际需求修改。

表 3-97 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

请根据**Step2 修改训练超参配置**修改超参值后，再启动训练脚本。Llama2-70B建议为8机64卡训练。

多机启动

以 **Llama2-70B** 为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
# 多机执行命令为：sh scripts/llama2/0_pl_pretrain_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8> <NODE_RANK=0>
示例:
# 第一台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  1
...
...
# 第八台节点
sh scripts/llama2/0_pl_pretrain_70b.sh  xx.xx.xx.xx  8  7
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、 NODE_RANK、 NODE_RANK 为必填。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以 **Llama2-13B** 为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

```
# 单机执行命令为：sh scripts/llama2/0_pl_pretrain_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
示例:
sh scripts/llama2/0_pl_pretrain_13b.sh localhost  1  0
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-180 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

3.11.4 SFT 全参微调训练任务

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的SFT微调**为例，执行脚本为**0_pl_sft_70b.sh** 和 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-97**所示。其他超参均有默认值，可以参考**表3-100**按照实际需求修改。

表 3-98 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。其中 Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_sft_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8>
<NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  1
# 第三台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  2
# 第四台节点
sh scripts/llama2/0_pl_sft_70b.sh  xx.xx.xx.xx  8  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、 NODE_RANK、 NODE_RANK为必填。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
# 单机执行命令为：sh scripts/llama2/0_pl_sft_13b.sh <MASTER_ADDR=localhost> <NNODES=1>
<NODE_RANK=0>
sh scripts/llama2/0_pl_sft_13b.sh localhost 1 0
```

训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

3.11.5 LoRA 微调训练

Step1 上传训练权重文件和数据集

如果在准备代码和数据阶段已经上传权重文件和数据集到容器中，可以忽略此步骤。

如果未上传训练权重文件和数据集到容器中，具体参考**上传代码和权重文件到工作环境**和**上传数据到指定目录**章节完成。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

如果想详细了解脚本执行训练权重转换操作和数据集预处理操作说明请分别参见**训练中的权重转换说明**和**训练的数据集预处理说明**。

Step2 修改训练超参配置

以**Llama2-70b和Llama2-13b的LoRA微调**为例，执行脚本为**0_pl_lora_70b.sh**和**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-97**所示。其他超参均有默认值，可以参考**表3-100**按照实际需求修改。

表 3-99 必须修改的训练超参配置

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step3 启动训练脚本

修改超参值后，再启动训练脚本。Llama2-70b建议为4机32卡训练。

多机启动

以 **Llama2-70b**为例，多台机器执行训练启动命令如下。进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。

```
多机执行命令为：sh scripts/llama2/0_pl_lora_70b.sh <MASTER_ADDR=xx.xx.xx.xx> <NNODES=8> <NODE_RANK=0>
示例:
#第一台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  0
# 第二台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  1
# 第三台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  2
```

```
# 第四台节点
sh scripts/llama2/0_pl_lora_70b.sh  xx.xx.xx.xx  8  3
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。其中MASTER_ADDR、NNODES、NODE_RANK为必填项。

单机启动

对于Llama2-7b和Llama2-13b，操作过程与Llama2-70b相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13b为例。

进入代码目录 **/home/ma-user/ws/llm_train/AscendSpeed** 下执行启动脚本。先修改以下命令中的参数，再复制执行

```
# 单机执行命令为：sh scripts/llama2/0_pl_lora_13b.sh <MASTER_ADDR=localhost> <NNODES=1> <NODE_RANK=0>
sh scripts/llama2/0_pl_lora_13b.sh localhost 1 0
```

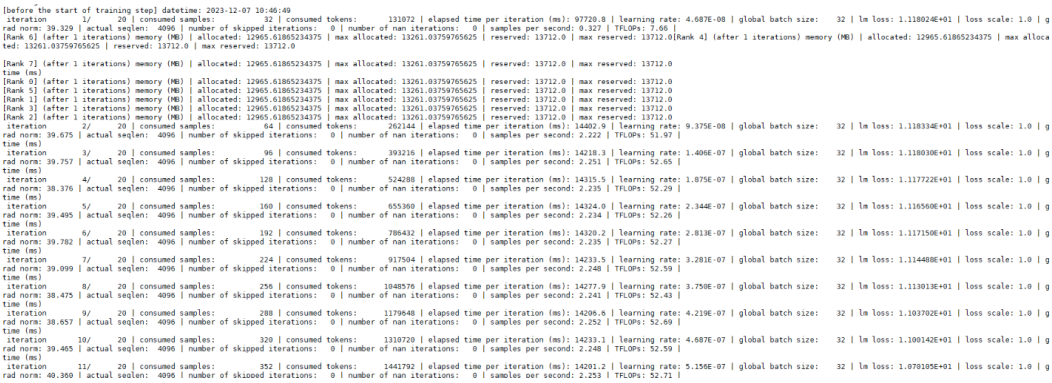
训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

3.11.6 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-181 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为：/home/ma-user/ws/saved_dir_for_ma_output/llama2-70b/logs

查看性能

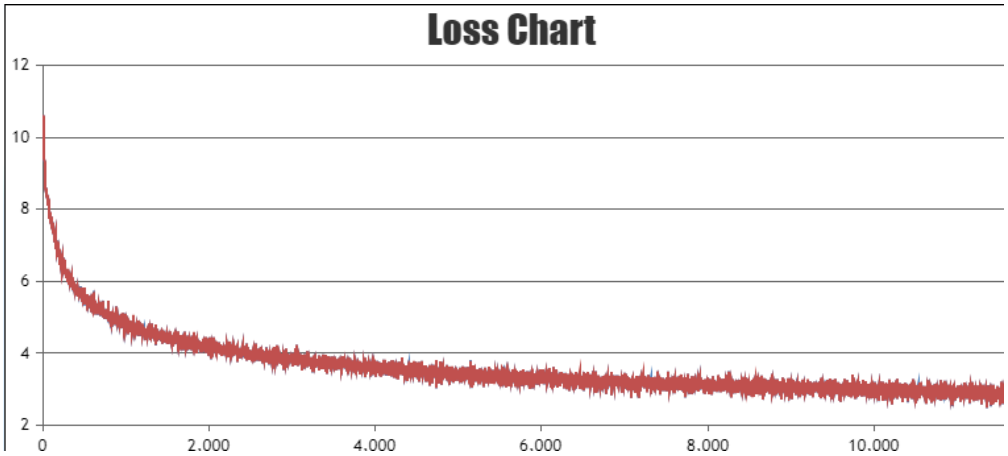
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数，具体参数查看**表3-100**。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-182**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-182 Loss 收敛情况（示意图）



3.11.7 训练脚本说明

3.11.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。若未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过编辑 **1_preprocess_data.sh** 、**2_convert_mg_hf.sh** 中的具体python指令运行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以 **llama2-70b 预训练**为例：

表 3-100 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/llm_train/AscendSpeed/model/llama2-70B	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-70b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler]	示例值需要根据数据集的不同，选择其一。 • GeneralPretrainHandler：使用预训练的alpaca数据集。 • GeneralInstructionHandler：使用微调的alpaca数据集。
MBS	1	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	128	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	8	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如**表3-101**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-101 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama 2	llama2-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
2		llama2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama 3	llama3-8b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen 1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
15	ChatGLMv3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

3.11.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**的过程。

若已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：moss-003-sft-data）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理使用执行卡数量 / 启动的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data/fintune/**

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/1_preprocess_data.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 3-102 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/ws/llm_train/AscendSpeed/training_data/${*用户自定义的数据集路径和名称*}	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/ws/llm_train/AscendSpeed/tokenizers/llama2-13b	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/ws/llm_train/AscendSpeed/processed_for_input/llama2-13b/data	处理后的数据集保存路径+数据集前缀
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.11.7.3 训练中的权重转换说明

以 llama2-13b 举例，运行 **0_pl_pretrain_13b.sh** 脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行 **scripts/llama2/2_convert_mg_hf.sh** 。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}** 目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在 **/home/ma-user/ws/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/** 目录下查看转换后的权重文件。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可直接编辑 **scripts/llama2/2_convert_mg_hf.sh** 脚本，自定义环境变量的值，并运行该脚本。其中环境变量详细介绍如下：

表 3-103 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/tokenizers/Llama2-13B	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.11.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-183所示。

图 3-183 修改 ChatGLMv3-6B tokenizer 文件（1）

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-184所示。

图 3-184 修改 ChatGLMv3-6B tokenizer 文件（2）

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer 文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-185所示。

图 3-185 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.12 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.905）

3.12.1 推理场景介绍

方案概览

本方案介绍了在ModelArts Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于部分企业客户。
- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM。vLLM支持v0.3.2。
- 支持FP16和BF16数据类型推理。

资源规格要求

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

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

说明

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

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-104**所示。

表 3-104 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的vLLM 0.3.2推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	6.3.905版本获取路径：**Support-E**（推荐） **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
AscendCloud-OPP-6.3.905-xxx.zip	推理依赖的算子包。	

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-105 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	Cann版本
6.3.905版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	cann_8.0.rc2

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

支持的模型列表和权重文件

本方案支持vLLM的v0.3.2版本。不同vLLM版本支持的模型列表有差异，具体如**表3-106**所示。

表 3-106 支持的模型列表和权重获取地址

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
1	llama-7b	√	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	√	**https://huggingface.co/huggyllama/llama-13b**

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
3	llama-65b	√	https://huggingface.co/huggyllama/llama-65b
4	llama2-7b	√	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	√	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	√	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	√	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	√	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	√	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	√	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	√	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	√	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	√	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	√	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	√	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	√	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	√	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	√	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	√	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	√	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat

序号	模型名称	支持vLLM v0.3.2	开源权重获取地址
21	qwen1.5-14b	√	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	√	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main
23	qwen1.5-72b	√	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	√	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	baichuan2-7b	√	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
26	baichuan2-13b	√	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
27	chatglm2-6b	√	https://huggingface.co/THUDM/chatglm2-6b
28	chatglm3-6b	√	https://huggingface.co/THUDM/chatglm3-6b
29	gemma-2b	√	https://huggingface.co/google/gemma-2b
30	gemma-7b	√	https://huggingface.co/google/gemma-7b
31	mistral-7b	√	https://huggingface.co/mistralai/Mistral-7B-v0.1

模型软件包结构说明

本教程需要使用到的AscendCloud-3rdLLM-xxx.zip软件包中的关键文件介绍如下。

```
├──llm_tools            #推理工具包
  ├──llm_evaluation  #推理评测代码包
    ├──benchmark_eval   # 精度评测
      ├── config
        ├── config.json  # 请求的参数，根据实际启动的服务来调整
        ├── mmlu_subject_mapping.json  # 数据集配置
        ├── ...
      ├── evaluators
        ├── evaluator.py # 数据集数据预处理方法集
        ├── model.py  # 发送请求的模块，在这里修改请求响应。目前支持vllm.openai，atb的tgi模板
        ├── ...
      ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
      ├── service_predict.py  # 发送请求的服务。支持vllm的openai，atb的tgi模板
      ├── ...
    ├──benchmark_tools  #性能评测
      ├── benchmark.py    # 可以基于默认的参数跑完静态benchmark和动态benchmark
      ├── benchmark_parallel.py  # 评测静态性能脚本
      ├── benchmark_serving.py   # 评测动态性能脚本
      ├── benchmark_utils.py     # 抽离的工具集
      ├── generate_datasets.py    # 生成自定义数据集的脚本
```

```
        ├── requirements.txt       # 第三方依赖
        ├── ...
├──llm_inference  #推理代码
        ├── ascend_vllm_adapter  #昇腾vLLM使用的算子模块
        ├── ascend.txt   #基于开源vLLM适配过NPU的patch脚本
        ├── autosmoothquant_ascend.txt   #基于开源autosmoothquant适配过NPU的patch脚本
        ├── build.sh     #推理构建脚本
        ├── requirements.txt       # 第三方依赖
```

相关文档

和本文档配套的模型训练文档请参考**《主流开源大模型基于DevServer适配PyTorch NPU训练指导》**。

3.12.2 部署推理服务

本章节介绍如何使用vLLM 0.3.2框架部署并启动推理服务。

前提条件

- 已准备好DevServer环境，具体参考**资源规格要求**。推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9b资源。
- 确保容器可以访问公网。

Step1 检查环境

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

2. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表3-105**。

```
docker pull {image_url}
```

Step3 上传代码包和权重文件

1. 上传安装依赖软件推理代码AscendCloud-3rdLLM-xxx.zip和算子包AscendCloud-OPP-xxx.zip到容器中，包获取路径请参见**表3-104**。
2. 将权重文件上传到DevServer机器中。权重文件的格式要求为Huggface格式。开源权重文件获取地址请参见**表3-106**。

Step4 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。

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

参数说明：

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

  说明

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

Step5 进入容器安装推理依赖软件

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
3. 解压算子包并将相应算子安装到环境中。
   ```
   unzip AscendCloud-OPP-*.zip
   pip install ascend_cloud_ops-1.0.0-py3-none-any.whl
   pip install cann_ops-1.0.0-py3-none-any.whl
   ```


4. 解压软件推理代码并安装依赖包。
```
unzip AscendCloud-3rdLLM-*.zip
cd llm_inference
pip install -r requirements.txt
```
5. 运行推理构建脚本build.sh文件，会自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。
```
cd llm_inference
bash build.sh
```
运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

Step6 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```
说明

NPU卡编号可以通过命令npu-smi info查询。
2. 配置PYTHONPATH。
```
export PYTHONPATH=$PYTHONPATH:${vllm_path}
```
${vllm_path} 填写ascend_vllm文件夹绝对路径。
3. 高阶配置（可选）。
   a. 词表切分。

      在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量：
```
export USE_VOCAB_PARALLEL=1   #打开词表切分开关
unset USE_VOCAB_PARALLEL      #关闭词表切分开关
```
      配置后重启服务生效。
   b. Matmul_all_reduce融合算子。

      使用Matmul_all_reduce融合算子能提升全量推理性能；该算子要求驱动和固件版本为Ascend HDK 24.1.RC1.B011及以上，默认不开启。如需开启，配置以下环境变量：
```
export USE_MM_ALL_REDUCE_OP=1   #打开Matmul_all_reduce融合算子
unset  USE_MM_ALL_REDUCE_OP     #关闭Matmul_all_reduce融合算子
```
      配置后重启服务生效。
   c. 查看详细日志。

      查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量：
```
export DETAIL_TIME_LOG=1   #打开打印详细日志
export RAY_DEDUP_LOGS=0    #打开打印详细日志
unset  DETAIL_TIME_LOG     #关闭打印详细日志
```
      配置后重启服务生效。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。详细启动服务与请求方式参考：**https://docs.vllm.ai/en/latest/getting_started/quickstart.html**。

说明

以下服务启动介绍的是在线推理方式，离线推理请参见**https://docs.vllm.ai/en/latest/getting_started/quickstart.html#offline-batched-inference**。

- **通过vLLM服务API接口启动服务**

  在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

- **通过OpenAI服务API接口启动服务**

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传代码包和权重文件**上传的HuggingFace权重文件存放目录。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。

- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:
Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on
http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step7 推理请求

使用命令测试推理服务是否正常启动。服务启动命令中的参数设置请参见**表3-107**。

- 方式一：通过OpenAI服务API接口启动服务使用以下推理测试命令。${docker_ip}替换为实际宿主机的IP地址。${container_model_path}请替换为实际使用的模型名称。

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

- 方式二：通过vLLM服务API接口启动服务使用以下推理测试命令。下面以Llama系列模型采样方式支持presence_penalty参数的发送请求为例。此处的接口8080需和**Step4 启动容器镜像**中设置的宿主机端口保持一致。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"temperature": 0,
"ignore_eos": false,
"presence_penalty":2
 }'
```

下面以Llama系列模型采样方式支持length_penalty参数的发送请求为例。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http:/${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "hello",
"max_tokens": 100,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"top_k": -1,
"use_beam_search":true,
"best_of":2,
"length_penalty":2
 }'
```

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

表 3-107 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.3.2）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9，qwen系列、qwen1.5系列、llama3系列模型还需打开词表切分配置export USE_VOCAB_PARALLEL=1。

序号	模型名称	4*64GB	8*32GB
1	qwen1.5-72b	24576	8192
2	qwen-72b	24576	8192
3	llama3-70b	32768	8192
4	llama2-70b	98304	32768
6	llama-65b	24576	8192

序号	模型名称	2*64GB	4*32GB
1	qwen1.5-32b	65536	24576

序号	模型名称	1*64GB	1*32GB
1	qwen1.5-7b	49152	16384
2	qwen-7b	49152	16384
3	llama3-8b	98304	32768
4	llama2-7b	126976	16384
5	chatglm3-6b	126976	65536
6	chatglm2-6b	126976	65536

序号	模型名称	1*64GB	2*32GB
1	qwen1.5-14b	24576	24576
2	qwen-14b	24576	24576
3	llama2-13b	24576	24576

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

3.12.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

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

性能benchmark验证使用到的脚本存放在代码包AscendCloud-3rdLLM-xxx.zip的llm_tools/llm_evaluation（6.3.905版本）目录中。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
```

```
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本、
├── requirements.txt      # 第三方依赖
```

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-3rdLLM-*x.x.x*.zip并解压，无需重复执行。
2. 进入benchmark_tools目录下，执行如下命令安装性能测试的关依赖。
   ```
   pip install -r requirements.txt
   ```
3. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。
   ```
   cd benchmark_tools
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```
   参数说明
   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存路径，如benchmark_parallel.csv。
4. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 3-186 静态 benchmark 测试结果（示意图）

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

动态 benchmark

本章节介绍如何进行动态benchmark验证。

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

   方法一：使用公开数据集

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

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

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

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

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。

- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

```
cd benchmark_tools
python benchmark_serving.py --backend vllm --host${docker_ip} --port 8085 --dataset
custom_datasets.json --dataset-type custom \
--tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000
1000 1000 1000 \
--max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
```

- --backend：服务类型，如"tgi"，vllm"，"mindspore"
- --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务端口
- --dataset：数据集路径
- --dataset-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
- --tokenizer：tokenizer路径，可以是huggingface的权重路径
- --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
- --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应
- --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值
- --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer
- --benchmark-csv：结果保存路径，如benchmark_serving.csv

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

图 3-187 动态 benchmark 测试结果（示意图）

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

3.12.4 推理精度测试

本章节介绍如何进行推理精度测试。

前提条件

确保容器可以访问公网。

Step1 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-3rdLLM-xxx.zip的llm_tools/llm_evaluation（6.3.905版本）目录中。代码目录结构如下。精度测试使用到的mmlu和ceval数据集已经提前打包在代码中。

```
benchmark_eval
├──apig_sdk       # ma校验包
├──cpu_npu        # 检测资源消耗
```

```
├── config
│   ├── config.json  # 服务的配置模板，已配置了ma-standard，tgi示例
│   ├── mmlu_subject_mapping.json  # mmlu数据集学科信息
│   ├── ceval_subject_mapping.json  # ceval数据集学科信息
├── evaluators
│   ├── evaluator.py # 数据集数据预处理方法集
│   ├── chatglm.py  # 处理请求相应模块, 一般和chatglm的官方评测数据集ceval搭配
│   ├── llama.py  # 处理请求相应模块, 一般和llama的评测数据集mmlu搭配
├── mmlu-exam, mmlu数据集
├── ceval-exam, ceval数据集
├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
├── readme.md     # 说明文档
├── requirements.txt   # 第三方依赖
├── service_predict.py  # 发送请求的服务
```

2. 上传精度测试代码到推理容器中。如果在**Step5 进入容器安装推理依赖软件**步骤中已经上传过AscendCloud-3rdLLM-*x.x.x*.zip并解压，无需重复执行。
3. 进入benchmark_eval目录下，执行如下命令安装性能测试的关依赖。

```
pip install -r requirements.txt
```

4. 执行精度测试启动脚本eval_test.py，具体操作命令如下，可以根据参数说明修改参数。

```
python eval_test.py \
  --max_workers=1 \
  --service_name=llama2-13b-chat-test \
  --eval_dataset=ceval \
  --service_url=http://${docker_ip}:8080/v1/completions \
  --few_shot=3 \
  --is_devserver=True \
  --model_name=llama2 \
  --deploy_method=vllm \
  --vllm_model=${model_path}
```

参数说明:

- max_workers：请求的最大线程数，默认为1。
- service_name：服务名称，保存评测结果时创建目录，示例为：llama2-13b-chat-test。
- eval_dataset：评测使用的评测集（枚举值），目前仅支持mmlu、ceval。
- service_url：成功部署推理服务后的服务预测地址，示例：http://${docker_ip}:8080/generate。此处的${docker_ip}替换为宿主机实际的IP地址，端口号8080来自前面配置的服务端口。
- few_shot：开启少量样本测试后添加示例样本的个数。默认为3，取值范围为0~5整数。
- is_devserver： 是否devserver部署方式，True表示DevServer模式。False表示ModelArts Standard模式。
- model_name：评测模型名称，llama2。
- deploy_method：部署方法，不同的部署方式api参数输入、输出解析方式不同，目前支持tgi、ma_standard、vllm等方式。
- vllm_model：deploy_method为vllm时，服务以openai的方式启动，vllm_model为启动服务时传入的model_path。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{service_name}/{eval_dataset}-{timestamp} 的目录结果保存到对应的测试工程。执行多少次，则会在{service_name}下生成多少次结果。

单独的评测结果如下：


```
{eval_dataset}-{timestamp}  # 例如: mmlu-20240205093257
├── accuracy
│   ├── evaluation_accuracy.xlsx  # 测试的评分结果，包含各个学科数据集的评分和总和评分。
├── infer_info
│   ├── xxx1.csv # 单个数据集的评测结果
│   ├── ......
│   ├── xxxn.csv # 单个数据集的评测结果
├── summary_result
│   ├── answer_correct.xlsx  # 回答正确的结果
│   ├── answer_error.xlsx  # 保存回答了问题的选项，但是回答结果错误
│   ├── answer_result_unknow.xlsx  # 保存未推理出结果的问题，例如超时、系统错误
│   ├── system_error.xlsx  # 保存推理结果，但是可能答非所问，无法判断是否正确，需要人工判断进行纠偏。
```

3.12.5 附录：大模型推理常见问题

问题1：在推理预测过程中遇到NPU out of memory

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

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

解决方法：

修改config.json文件中的"seq_length"的值，"seq_length"需要大于等于 --max-model-len的值。

config.json存在模型对应的路径下，例如：/data/nfs/benchmark/tokenizer/chatglm3-6b/config.json

3.13 主流开源大模型基于 Standard 适配 PyTorch NPU 训练指导（6.3.905）

3.13.1 场景介绍

方案概览

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

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

适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。

约束限制

本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持以下模型的训练，如**表3-108**所示。

表 3-108 代码包中适配的模型

序号	支持模型	支持模型参数量	权重文件获取地址
1	Llama2	llama2-7b	**https://huggingface.co/meta-llama/Llama-2-7b-chat-hf**
2		llama2-13b	**https://huggingface.co/meta-llama/Llama-2-13b-chat-hf**
3		llama2-70b	**https://huggingface.co/meta-llama/Llama-2-70b-hf** **https://huggingface.co/meta-llama/Llama-2-70b-chat-hf** (推荐)
4	Llama3	llama3-8b	**https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct**
5		llama3-70b	**https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct**
6	Qwen	qwen-7b	**https://huggingface.co/Qwen/Qwen-7B-Chat**
7		qwen-14b	**https://huggingface.co/Qwen/Qwen-14B-Chat**
8		qwen-72b	**https://huggingface.co/Qwen/Qwen-72B-Chat**
9	Qwen1.5	qwen1.5-7b	**https://huggingface.co/Qwen/Qwen1.5-7B-Chat**
10		qwen1.5-14b	**https://huggingface.co/Qwen/Qwen1.5-14B-Chat**
11		qwen1.5-32b	**https://huggingface.co/Qwen/Qwen1.5-32B-Chat**
12		qwen1.5-72b	**https://huggingface.co/Qwen/Qwen1.5-72B-Chat**
13	Yi	yi-6b	**https://huggingface.co/01-ai/Yi-6B-Chat**
14		yi-34b	**https://huggingface.co/01-ai/Yi-34B-Chat**
15	ChatGLMv3	glm3-6b	**https://huggingface.co/THUDM/chatglm3-6b**
16	Baichuan2	baichuan2-13b	**https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat**

操作流程

图 3-188 操作流程图

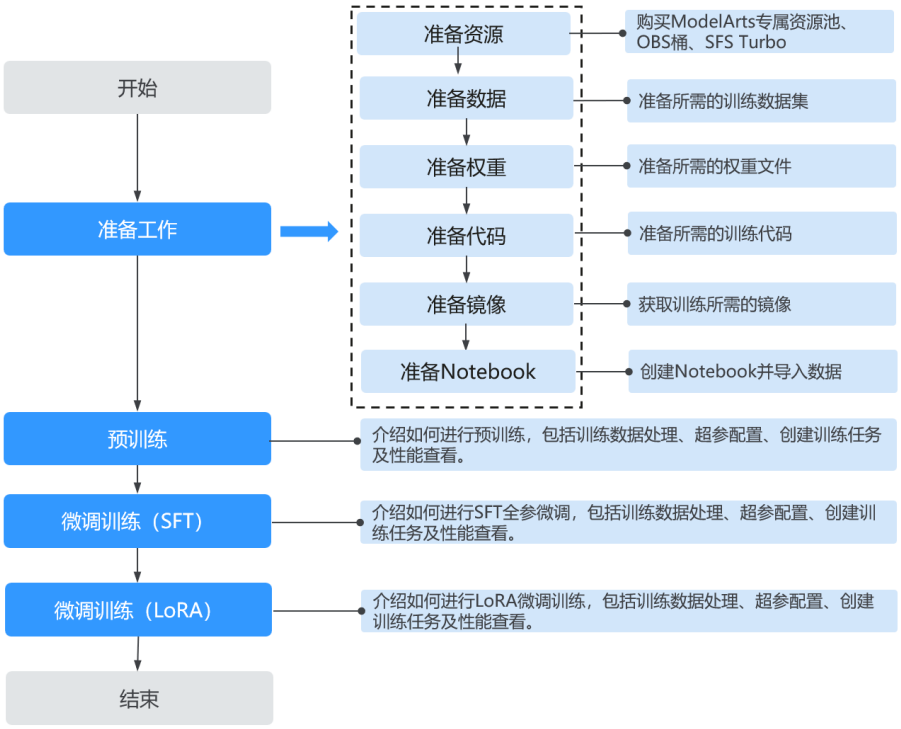


表 3-109 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行的，需要购买并开通ModelArts专属资源池和OBS桶。
	准备数据	准备训练数据，可以用本案使用的数据集，也可以使用自己准备的数据集。
	准备权重	准备所需的权重文件。
	准备代码	准备AscendSpeed训练代码。
	准备镜像	准备训练模型适用的容器镜像。
	准备Notebook	本案例需要创建一个Notebook，以便能够通过它访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、创建训练任务及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调，包括训练数据处理、超参配置、创建训练任务及性能查看。

阶段	任务	说明
	LoRA微调训练	介绍如何进行LoRA微调训练，包括训练数据处理、超参配置、创建训练任务及性能查看。

3.13.2 准备工作

3.13.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

计算规格：用户可参考**表3-116**。

硬盘空间：至少200GB。

昇腾资源规格：

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

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码以及输入输出数据存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-llama2-13b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：training_data。

创建 VPC

虚拟私有云（Virtual Private Cloud）可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

创建 SFS Turbo

SFS Turbo HPC型文件系统为用户提供一个完全托管的共享文件存储。SFS Turbo文件系统支持无缝访问存储在OBS对象存储桶中的对象，用户可以指定SFS Turbo内的目录与OBS对象存储桶进行关联，然后通过创建导入导出任务实现数据同步。通过OBS与SFS Turbo存储联动，可以将最新的训练数据导入到SFS Turbo，然后在训练作业中挂载SFS Turbo到容器对应ckpt目录，实现分布式读取训练数据文件。

创建SFS Turbo文件系统，详细操作指导请参考**创建SFS Turbo文件系统**。

图 3-189 创建 SFS Turbo

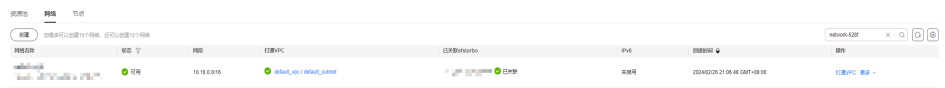
其中，文件系统类型推荐选用500MB/s/TiB或1000MB/s/TiB，应用于AI大模型场景中。存储容量推荐使用 6.0~10.8TB ，以存储更多模型文件。

图 3-190 SFS 类型和容量选择

ModelArts 网络关联 SFS Turbo

OBS-SFS Turbo联动方案涉及VPC、SFS Turbo HPC型文件系统、OBS对象存储服务和ModelArts资源池。如果要使用训练作业挂载SFS Turbo功能，则需要配置ModelArts和SFS Turbo间网络直通，以及配置ModelArts网络关联SFS Turbo。 具体操作请参见**配置ModelArts和SFS Turbo间网络直通**。

图 3-191 ModelArts 网络关联 SFS Turbo



3.13.2.2 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

数据集下载

本教程使用Alpaca数据集，数据集的介绍及下载链接如下。

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

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

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据集至 OBS

1. 准备数据集，例如根据Alpaca数据部分给出的预训练数据集、SFT全参微调训练、LoRA微调训练数据集下载链接下载数据集。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放数据，例如在桶standard-llama2-13b中创建文件夹training_data。
3. 利用**OBS Browser+工具**将步骤1下载的数据集上传至步骤2创建的文件夹目录下。得到OBS下数据集结构：

```
obs://<bucket_name>/training_data
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  # 训练原始数据集
    |── alpaca_gpt4_data.json                          # 微调数据文件
```

3.13.2.3 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-108**。
2. 在**创建OBS桶**创建的桶下创建文件夹用以存放权重和词表文件，例如在桶standard-llama2-13b中创建文件夹llama2-13B-chat-hf。
3. 参考文档利用OBS-Browser-Plus工具将步骤1下载的权重文件上传至步骤2创建的文件夹目录下。得到OBS下数据集结构，此处以llama2-13B为例（权重文件可能变化，以下仅为举例）：

```
obs://<bucket_name>/model/llama-2-13b-chat-hf/
├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
```

```
├── tokenizer.model
└── USE_POLICY.md
```

3.13.2.4 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取模型软件包

本方案支持的模型对应的软件和依赖包获取地址如**表3-110**所示。

表 3-110 模型对应的软件包和依赖包获取地址

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** 请联系您所在企业的华为方技术支持下载获取。

模型软件包结构说明

AscendCloud-3rdLLM代码包结构介绍如下，训练脚本以分类的方式集中在scripts文件夹中：

```
├──llm_train                # 模型训练代码包
        ├──AscendSpeed         # 基于AscendSpeed的训练代码
              ├──ascendcloud_patch/   # 针对昇腾云平台适配的功能补丁包
              ├──scripts/             # 训练需要的启动脚本
                   ├──llama2          # llama2系列模型执行脚本的文件夹
                   ├──llama3          # llama3系列模型执行脚本的文件夹
                   ├──qwen            # Qwen系列模型执行脚本的文件夹
                   ├──qwen1.5         # Qwen1.5系列模型执行脚本的文件夹
                   ├── ...
                   ├── dev_pipeline.sh  # 系列模型共同调用的多功能的脚本
                   ├── install.sh       # 环境部署脚本
├──llm_inference                # 推理代码包
├──llm_tools                    # 推理工具
```

注意

下载代码之后需要修改llm_train/AscendSpeed/scripts/install.sh文件。具体为删除install.sh的第43行 "git cherrypick 171ba0b3"。该问题会导致代码安装失败，会在后续版本修复。

代码上传至 OBS

将AscendSpeed代码包AscendCloud-3rdLLM-905-xxx.zip在本地解压缩后，将llm_train文件上传至OBS中。

结合**准备数据**、**准备权重**、**准备代码**，将数据集、原始权重、代码文件都上传至OBS后，OBS桶的目录结构如下。

```
<bucket_name>
|──llm_train                    # 解压代码包后自动生成的代码目录，无需用户创建
    |── AscendSpeed              # 代码目录
        |──ascendcloud_patch/   # 针对昇腾云平台适配的功能代码包
        |──scripts/             # 训练需要的启动脚本
# 以下目录结构，用户自己创建
|── training_data                                  #原始数据目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── train-00000-of-00001-a09b74b3ef9c3b56.parquet    #预训练时预处理后的数据存放地址
    |── alpaca_gpt4_data.json                           #微调数据文件
|── model                                          #原始权重及tokenizer目录，需要用户手动创建并上传，后续操作步骤中会提示
    |── llama2-13b-hf
```

3.13.2.5 准备镜像

准备训练Llama2-13B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像地址

本教程中用到的训练的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-111 基础容器镜像地址

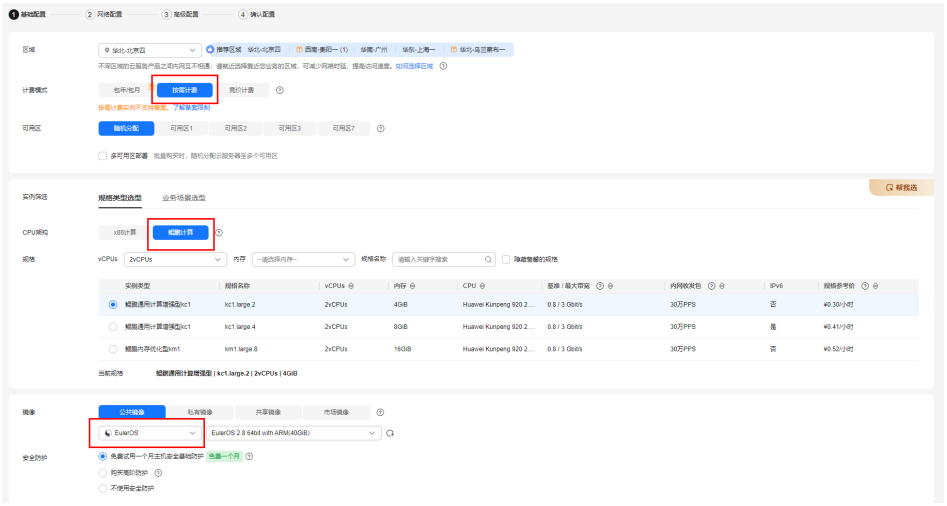
镜像用途	镜像地址	配套版本
训练基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个训练镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 3-192 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step3 创建镜像组织

在SWR服务页面创建镜像组织。

图 3-193 创建镜像组织

Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-194 复制登录指令

Step5 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}请参见表3-111。

```
docker pull {image_url}
```

Step6 修改并上传镜像

1. 登录指令输入之后，使用下列示例命令：

```
docker tag {image_url} <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。

- <组织名称>：Step3中自己创建的组织名称。示例：GROUP_NAME
- <镜像名称>:<版本名称>：定义镜像名称。示例：pytorch_2_1_ascend:20240528

示例：

docker tag swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240528

2. 上传镜像至镜像仓库。

```
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

示例：

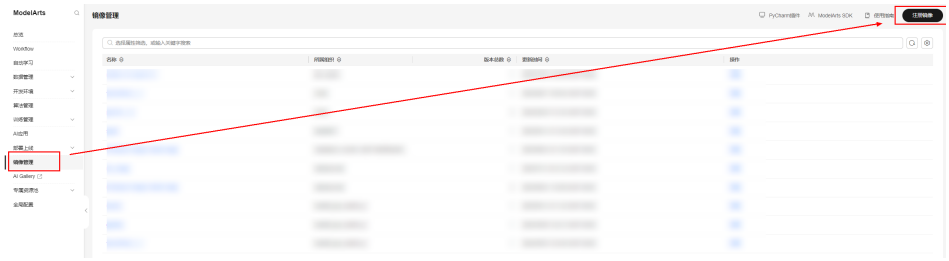
docker push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/pytorch_2_1_ascend:20240528

Step7 ModelArts 中注册镜像

镜像上传后，可在SWR中查看已上传的镜像。但在ModelArts中还需要完成镜像注册后，才能在后续的Notebook中使用。

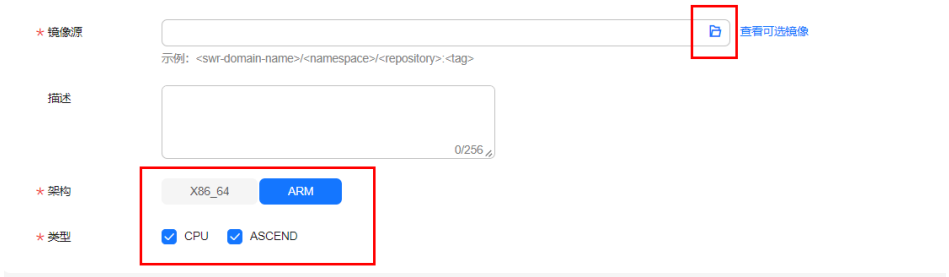
访问ModelArts，在镜像管理中选择注册镜像，如图所示：

图 3-195 注册镜像



选择已上传的镜像源，架构选择ARM，类型勾选CPU和ASCEDN，完成镜像注册。

图 3-196 选择已上传的镜像源



3.13.2.6 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。

本案例中的训练作业需要通过SFS Turbo挂载盘的形式创建，因此需要将上述数据集、代码、权重文件从OBS桶上传至SFS Turbo中。

用户需要创建开发环境Notebook，并绑定SFS Turbo，以便能够通过Notebook访问SFS Turbo服务。随后，通过Notebook将OBS中的数据上传至SFS Turbo，并对存储在SFS Turbo中的数据执行编辑操作。

创建 Notebook

创建开发环境Notebook实例，具体操作步骤请参考**创建Notebook实例**。

镜像选择已注册的自定义镜像，资源类型选择创建好的专属资源池，规格推荐选择“Ascend: 8*ascend-snt9b”。

图 3-197 Notebook 中选择自定义镜像与规格

存储配置选择“弹性文件服务SFS”，并且选择已创建的SFS Turbo实例。如果该SFS Turbo多人共用，则推荐用户编辑“子目录挂载”，创建自己的子目录进行划分。

图 3-198 Notebook 中选择弹性文件服务

使用 Notebook 将 OBS 数据导入 SFS Turbo

打开已创建的Notebook实例，选择Notebook的python-3.9.10，即可编辑Untitled.ipynb文件。编写以下代码，并运行Untitled.ipynb文件（用于将OBS中的数据导入至SFS Turbo）。

```
import moxing as mox
#obs存放数据路径
obs_code_dir= "obs://<bucket_name>/llm_train"
obs_data_dir= "obs://<bucket_name>/training_data"
obs_model_dir= "obs://<bucket_name>/model"
# NoteBook中存放数据路径
local_code_dir= "/home/ma-user/work/llm_train"
local_data_dir= "/home/ma-user/work/training_data"
local_model_dir= "/home/ma-user/work/model"
mox.file.copy_parallel(obs_code_dir,local_code_dir)
mox.file.copy_parallel(obs_data_dir,local_data_dir)
mox.file.copy_parallel(obs_model_dir,local_model_dir)
```

以此，OBS中的数据已迁移至SFS Turbo中，并可通过Notebook随时访问并编辑SFS Turbo中的数据。

Notebook 中安装依赖包并保存镜像

在后续训练步骤中，训练作业启动命令中包含sh scripts/install.sh，该命令用于git clone完整的代码包和安装必要的依赖包，每次启动训练作业时会执行该命令安装。

您可以在Notebook中导入完代码之后，在Notebook运行sh scripts/install.sh命令提前下载完整代码包和安装依赖包，然后使用保存镜像功能。后续训练作业使用新保存的镜像，无需每次启动训练作业时再次下载代码包以及安装依赖包，可节约训练作业启动时间。

由于训练启动命令也会执行sh scripts/install.sh安装依赖包，因此Notebook保存镜像为可选操作。

图 3-199 安装依赖包

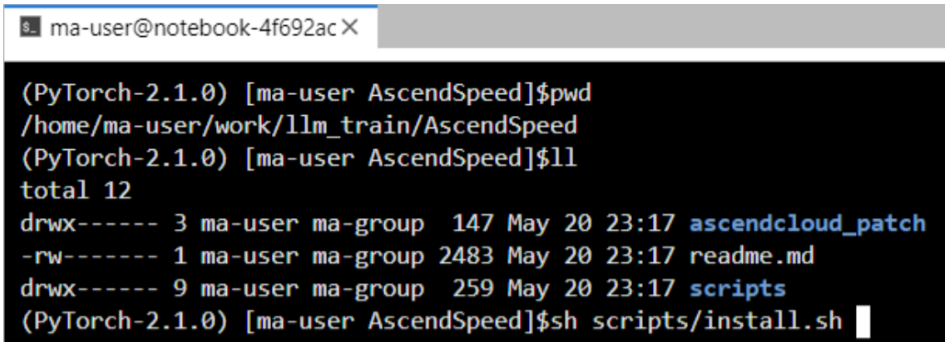


图 3-200 保存镜像

图 3-201 填写保存镜像相关参数

3.13.3 预训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b预训练**为例，执行脚本**0_pl_pretrain_13b.sh**。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-112**所示。其他超参均有默认值，可以参考**表3-115**按照实际需求修改。

表 3-112 必须修改的训练超参配置

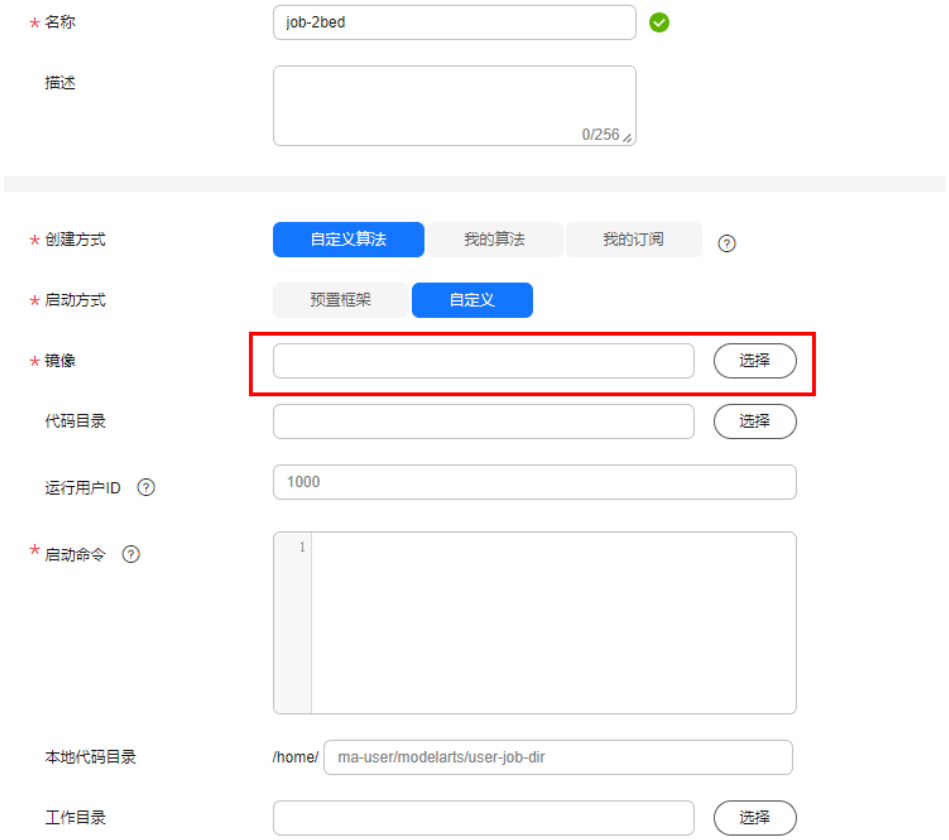
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建预训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-202 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_pretrain_13b.sh
```

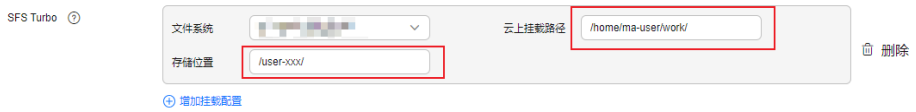
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-116**进行配置。

图 3-203 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-204 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.13.4 SFT 全参微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b SFT微调**为例，执行脚本 **0_pl_sft_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-113**所示。其他超参均有默认值，可以参考**表3-115**按照实际需求修改。

表 3-113 必须修改的训练超参配置

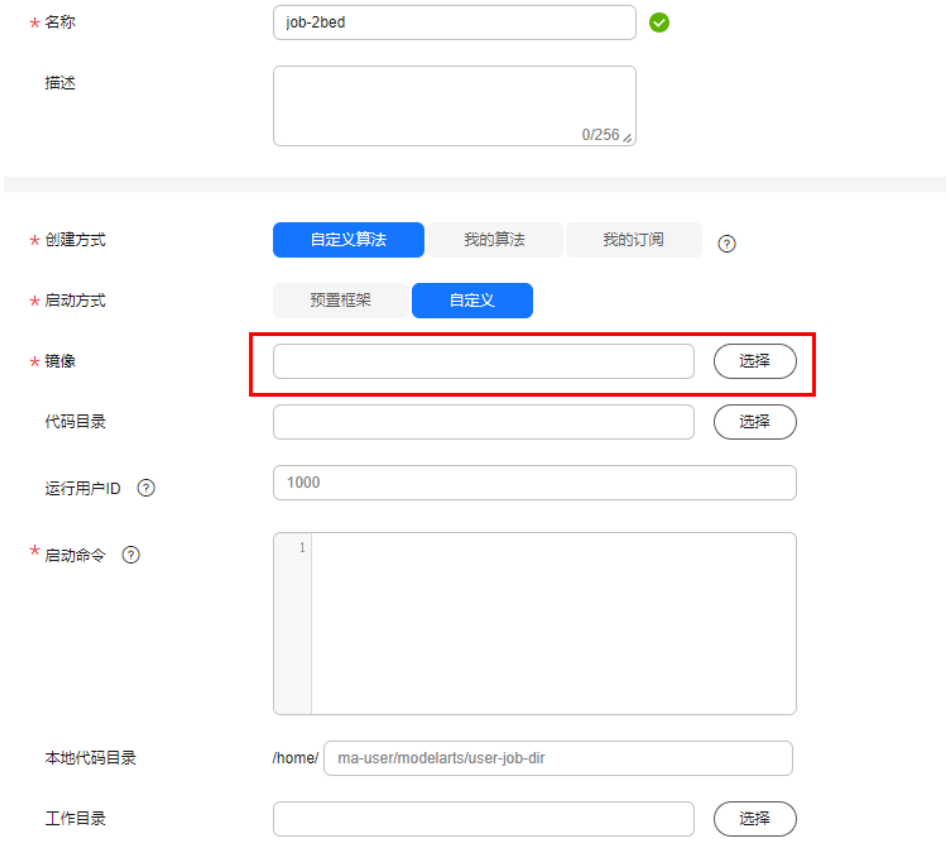
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 SFT 全参微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-205 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_sft_13b.sh
```

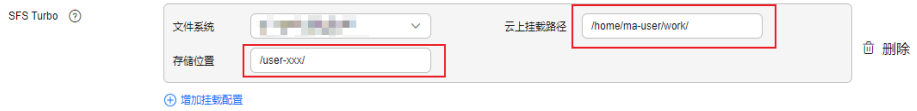
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-116**进行配置。

图 3-206 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-207 选择 SFS Turbo



作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.13.5 LoRA 微调训练

前提条件

已上传训练代码、训练权重文件和数据集到SFS Turbo中，具体参考**代码上传至OBS**和**使用Notebook将OBS数据导入SFS Turbo**。

Step1 在 Notebook 中修改训练超参配置

以**llama2-13b LORA微调**为例，执行脚本**0_pl_lora_13b.sh** 。

修改模型训练脚本中的超参配置，必须修改的参数如**表3-114**所示。其他超参均有默认值，可以参考**表3-115**按照实际需求修改。

表 3-114 必须修改的训练超参配置

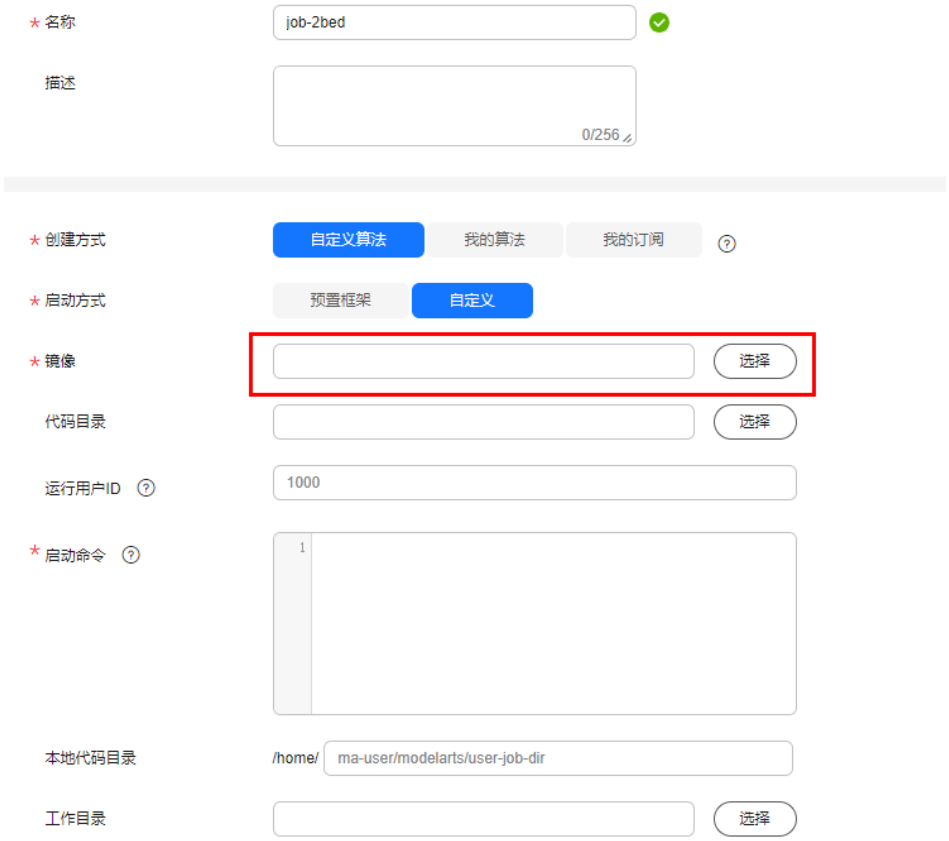
参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/alpaca_gpt4_data.json	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。

对于ChatGLMv3-6B和Qwen系列模型，还需要手动修改tokenizer文件，具体请参见**训练tokenizer文件说明**。

Step2 创建 LoRA 微调训练任务

创建训练作业，并自定义名称、描述等信息。选择自定义算法，启动方式自定义，以及上传的镜像。训练脚本中会自动执行训练前的权重转换操作和数据处理操作。

图 3-208 选择镜像



训练作业启动命令中输入：

```
cd /home/ma-user/work/llm_train/AscendSpeed;
sh ./scripts/install.sh;
sh ./scripts/llama2/0_pl_lora_13b.sh
```

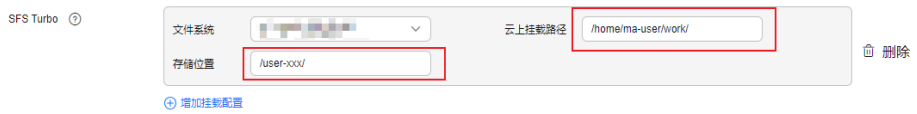
选择用户自己的专属资源池，以及规格与节点数。防止训练过程中出现内存溢出的情况，用户可参考**表3-116**进行配置。

图 3-209 选择资源池规格

新增SFS Turbo挂载配置，并选择用户创建的SFS Turbo文件系统。

- 云上挂载路径：输入镜像容器中的工作路径 /home/ma-user/work/
- 存储位置：输入用户在Notebook中创建的“子目录挂载”

图 3-210 选择 SFS Turbo



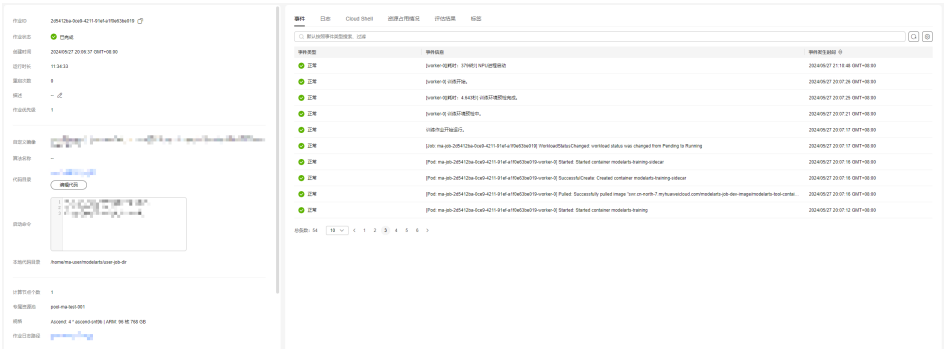
作业日志选择OBS中的路径，训练作业的日志信息则保存该路径下。

最后，提交训练作业，训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。了解更多ModelArts训练功能，可查看**模型开发简介**。

3.13.6 查看日志和性能

单击作业详情页面，则可查看训练过程中的详细信息。

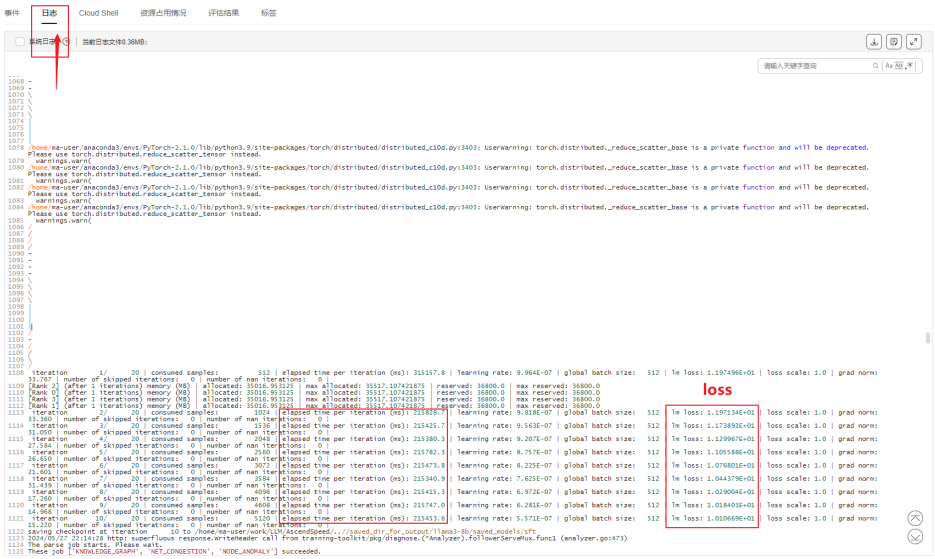
图 3-211 查看训练作业



在作业详情页的日志页签，查看最后一个节点的日志，其包含“elapsed time per iteration (ms)”数据，可换算为tokens/s/p的性能数据。

- 吞吐量（tokens/s/p）：global batch size*seq_length/(总卡数*elapsed time per iteration)*1000，其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数
- loss收敛情况：日志里存在lm loss参数 ，lm loss的值随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。

图 3-212 查看日志和性能



3.13.7 训练脚本说明

3.13.7.1 训练启动脚本说明和参数配置

本代码包中集成了不同模型（包括llama2、llama3、Qwen、Qwen1.5 .....）的训练脚本，**并可通过不同模型中的训练脚本一键式运行**。训练脚本可判断是否完成预处理后的数据和权重转换的模型。如果未完成，则执行脚本，**自动完成数据预处理和权重转换的过程**。

若用户进行自定义数据集预处理以及权重转换，可通过**Notebook**环境编辑**1_preprocess_data.sh** 、**2_convert_mg_hf.sh**中的具体python指令，并在**Notebook**环境中运行执行。本代码中有许多环境变量的设置，在下面的指导步骤中，会展开进行详细的解释。

若用户希望自定义参数进行训练，可直接编辑对应模型的训练脚本，可编辑参数以及详细介绍如下。以**llama2-13b预训练**为例：

表 3-115 模型训练脚本参数

参数	示例值	参数说明
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/pretrain/alpaca.parquet	**必须修改**。训练时指定的输入数据路径。请根据实际规划修改。
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/llama-2-13b-chat-hf	**必须修改**。加载tokenizer与Hugging Face权重时，对应的存放地址。请根据实际规划修改。
MODEL_NAME	llama2-13b	对应模型名称。
RUN_TYPE	pretrain	表示训练类型。可选择值：[pretrain, sft, lora]。

参数	示例值	参数说明
DATA_TYPE	[GeneralPretrainHandler, GeneralInstructionHandler]	示例值需要根据数据集的不同，选择其一。 GeneralPretrainHandler：使用预训练的alpaca数据集； GeneralInstructionHandler：使用微调的alpaca数据集；
MBS	4	表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。
GBS	512	表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。
TP	8	表示张量并行。
PP	1	表示流水线并行。一般此值与训练节点数相等，与权重转换时设置的值相等。
LR	2.5e-5	学习率设置。
MIN_LR	2.5e-6	最小学习率设置。
SEQ_LEN	4096	要处理的最大序列长度。
MAX_PE	8192	设置模型能够处理的最大序列长度。
SN	1200	**必须修改**。指定的输入数据集中数据的总数量。更换数据集时，需要修改。
EPOCH	5	表示训练轮次，根据实际需要修改。一个Epoch是将所有训练样本训练一次的过程。
TRAIN_ITERS	SN / GBS * EPOCH	非必填。表示训练step迭代次数，根据实际需要修改。
SEED	1234	随机种子数。每次数据采样时，保持一致。

不同模型推荐的训练参数和计算规格要求如**表3-116**所示。规格与节点数中的1*节点 & 4*Ascend表示单机4卡，以此类推。

表 3-116 不同模型推荐的参数与 NPU 卡数设置

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
1	llama2	llama2-7b	SEQ_LEN =4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
2		llama2-13b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
3		llama2-70b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
4	llama3	llama3-8b	SEQ_LEN =4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN =8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
5		llama3-70b	SEQ_LEN =4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
6	Qwen	qwen-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
7		qwen-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
8		qwen-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
9	Qwen1.5	qwen1.5-7b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
10		qwen1.5-14b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
11		qwen1.5-32b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
12		qwen1.5-72b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=4	4*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=8	8*节点 & 8*Ascend
13	Yi	yi-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
14		yi-34b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=2	2*节点 & 8*Ascend
15	ChatGLM v3	glm3-6b	SEQ_LEN=4096	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend

序号	支持模型	支持模型参数量	文本序列长度	并行参数设置	规格与节点数
			SEQ_LEN=8192	TP(tensor model parallel size)=4 PP(pipeline model parallel size)=1	1*节点 & 4*Ascend
16	Baichuan2	baichuan2-13b	SEQ_LEN=4096	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend
			SEQ_LEN=8192	TP(tensor model parallel size)=8 PP(pipeline model parallel size)=1	1*节点 & 8*Ascend

3.13.7.2 训练的数据集预处理说明

以 **llama2-13b** 举例，使用训练作业运行：**0_pl_pretrain_13b.sh** 训练脚本后，脚本检查**是否已经完成数据集预处理**。

如果已完成数据集预处理，则直接执行预训练任务。若未进行数据集预处理，则会自动执行 **scripts/llama2/1_preprocess_data.sh** 。

预训练数据集预处理参数说明

预训练数据集预处理脚本 scripts/llama2/1_preprocess_data.sh 中的具体参数如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（例如：alpaca_gpt4_data）。
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理时，要执行的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以 **llama2-13b** 为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/**

微调数据集预处理参数说明

微调包含SFT和LoRA微调。数据集预处理脚本参数说明如下：

- --input：原始数据集的存放路径。
- --output-prefix：处理后的数据集保存路径+数据集名称（ 例如：alpaca_gpt4_data ）
- --tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- --tokenizer-name-or-path：tokenizer的存放路径，与HF权重存放在一个文件夹下。
- --handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
  - GeneralPretrainHandler：默认值。用于预训练时的数据预处理过程中，将数据集根据key值进行简单的过滤。
  - GeneralInstructionHandler：用于sft、lora微调时的数据预处理过程中，会对数据集full_prompt中的user_prompt进行mask操作。
- --seq-length：要处理的最大seq length。
- --workers：设置数据处理时，要执行的工作进程数。
- --log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。

输出数据预处理结果路径：

训练完成后，以**llama2-13b**为例，输出数据路径为：**/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/fintune/**

用户自定义执行数据处理脚本修改参数说明

同样以 llama2 为例，用户在**Notebook**中直接编辑**scripts/llama2/1_preprocess_data.sh**脚本，自定义环境变量的值，并在**Notebook**中运行该脚本。其中环境变量详细介绍如下：

表 3-117 数据预处理中的环境变量

环境变量	示例	参数说明
RUN_TYPE	pretrain、sft、lora	数据预处理区分： 预训练场景下数据预处理，默认参数：**pretrain** 微调场景下数据预处理，默认：**sft / lora**
ORIGINAL_TRAIN_DATA_PATH	/home/ma-user/work/training_data/finetune/moss_LossCompare.jsonl	原始数据集的存放路径。
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer的存放路径，与HF权重存放在一个文件夹下。请根据实际规划修改。

环境变量	示例	参数说明
PROCESSED_DATA_PREFIX	/home/ma-user/work/llm_train/processed_for_input/llama2-13b/data/pretrain/alpaca	处理后的数据集保存路径+数据集前缀。
TOKENIZER_TYPE	PretrainedFromHF	可选项有：['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF 。
SEQ_LEN	4096	要处理的最大seq length。脚本会检测超出SEQ_LEN长度的数据，并打印log。

3.13.7.3 训练的权重转换说明

以llama2-13b举例，使用训练作业运行**0_pl_pretrain_13b.sh**脚本。脚本同样还会检查**是否已经完成权重转换**的过程。

若已完成权重转换，则直接执行预训练任务。若未进行权重转换，则会自动执行**scripts/llama2/2_convert_mg_hf.sh**。脚本具体参数如下：

HuggingFace 转 Megatron 参数说明

- --model-type：模型类型。
- --loader：选择对应加载模型脚本的名称。
- --saver：选择模型保存脚本的名称。
- --tensor-model-parallel-size：${TP}张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：${PP}流水线并行数，需要与训练脚本中的PP值配置一样。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP${TP}PP${PP}**目录下查看转换后的权重文件。

Megatron 转 HuggingFace 参数说明

训练完成的权重文件默认不会自动转换为Hugging Face格式权重。若用户需要自动转换，则在运行脚本，例如**0_pl_pretrain_13b.sh**中，添加变量**CONVERT_MG2HF**并赋值**TRUE**。若用户后续不需要自动转换，则在运行脚本中必须删除**CONVERT_MG2HF**变量。

Megatron转HuggingFace脚本具体参数如下：

- --model-type：模型类型。
- --save-model-type：输出后权重格式。
- --load-dir：训练完成后保存的权重路径。
- --save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-13B/mg2hg下。
- --target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- --target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

输出转换后权重文件保存路径：

权重转换完成后，在**/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b/saved_models/pretrain_hf/**目录下查看转换后的权重文件。

用户自定义执行权重转换参数修改说明

同样以 llama2 为例，用户可在**Notebook**直接编辑**scripts/llama2/2_convert_mg_hf.sh**脚本，自定义环境变量的值，并在**Notebook**运行该脚本。其中环境变量详细介绍如下：

表 3-118 权重转换脚本中的环境变量

参数	示例	参数说明
$1	hf2hg、mg2hf	运行 2_convert_mg_hf.sh 时，需要附加的参数值。如下： hf2hg：用于Hugging Face 转 Megatron mg2hf：用于Megatron 转 Hugging Face
TP	8	张量并行数，一般等于单机卡数
PP	1	流水线并行数，一般等于节点数量
ORIGINAL_HF_WEIGHT	/home/ma-user/work/model/Llama2-13B	原始Hugging Face模型路径
CONVERT_MODEL_PATH	/home/ma-user/work/llm_train/processed_for_ma_input/llama2-13b/converted_weights_TP8PP1	权重转换完成之后保存路径
TOKENIZER_PATH	/home/ma-user/work/model/llama-2-13b-chat-hf	tokenizer路径，即：原始Hugging Face模型路径

参数	示例	参数说明
MODEL_SAVE_PATH	/home/ma-user/work/llm_train/saved_dir_for_output/llama2-13b	训练完成后保存的权重路径。

3.13.7.4 训练 tokenizer 文件说明

在训练开始前，需要针对模型的tokenizer文件进行修改，不同模型的tokenizer文件修改内容如下，您可在创建的Notebook中对tokenizer文件进行编辑。

ChatGLMv3-6B

在训练开始前，针对ChatGLMv3-6B模型中的tokenizer文件，需要修改代码。修改文件chatglm3-6b/tokenization_chatglm.py 。

271行要添加注释，修改后如图3-213所示。

图 3-213 修改 ChatGLMv3-6B tokenizer 文件（1）

```
270         # Load from model defaults
271         # assert self.padding_side == "left"
```

291至300行要修改，修改后如图3-214所示。

图 3-214 修改 ChatGLMv3-6B tokenizer 文件（2）

```
291         if needs_to_be_padded:
292             difference = max_length - len(required_input)
293 
294             if "attention_mask" in encoded_inputs:
295                 encoded_inputs["attention_mask"] = encoded_inputs["attention_mask"] + [0] * difference
296             if "position_ids" in encoded_inputs:
297                 encoded_inputs["position_ids"] = encoded_inputs["position_ids"] + [0] * difference
298             encoded_inputs[self.model_input_names[0]] = required_input + [self.pad_token_id] * difference
299 
300         return encoded_inputs
```

Qwen 系列

在进行HuggingFace权重转换Megatron前，针对Qwen系列模型中的tokenizer文件，需要修改代码。

修改tokenizer目录下面modeling_qwen.py文件的第38和39行，修改后如图3-215所示。

图 3-215 修改 Qwen tokenizer 文件

```
29 from transformers.utils import logging
30 
31 try:
32     from einops import rearrange
33 except ImportError:
34     rearrange = None
35 from torch import nn
36 
37 SUPPORT_CUDA = torch.cuda.is_available()
38 SUPPORT_BF16 = SUPPORT_CUDA and True
39 SUPPORT_FP16 = SUPPORT_CUDA and True
40 SUPPORT_TORCH2 = hasattr(torch, '__version__') and int(torch.__version__.split(".")[0]) >= 2
41 
42 
43 from .configuration_qwen import QWenConfig
44 from .qwen_generation_utils import (
45     HistoryType,
```

3.14 主流开源大模型基于 Standard 适配 PyTorch NPU 推理指导（6.3.905）

3.14.1 场景介绍

方案概览

本文档介绍了在ModelArts的Standard上使用昇腾计算资源开展常见开源大模型Llama、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程，利用适配昇腾平台的大模型推理服务框架vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

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

约束限制

- 推理部署使用的服务框架是vLLM（官网地址：https://github.com/vllm-project/vllm/tree/v0.3.2，版本：v0.3.2）。
- 仅支持FP16和BF16数据类型推理。
- 适配的CANN版本是cann_8.0.rc2，驱动版本是23.0.5。
- 本案例仅支持在专属资源池上运行。

支持的模型列表

本方案支持的模型列表、对应的开源权重获取地址如表3-119所示。

表 3-119 支持的模型列表和权重获取地址

序号	支持模型	支持模型参数量	开源权重获取地址
1	Llama	llama-7b	https://huggingface.co/huggyllama/llama-7b
2		llama-13b	https://huggingface.co/huggyllama/llama-13b
3		llama-65b	https://huggingface.co/huggyllama/llama-65b
4	Llama 2-	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5		llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

序号	支持模型	支持模型参数量	开源权重获取地址
6		llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	Llama3	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8		llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	Yi	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
10		yi-9b	https://huggingface.co/01-ai/Yi-9B
11		yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
12	Deepseek	deepseek-llm-7b	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13		deepseek-coder-instruct-33b	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14		deepseek-llm-67b	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	Qwen	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
16		qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
17		qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
18	Qwen1.5	qwen1.5-0.5b	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19		qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20		qwen1.5-1.8b	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21		qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22		qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B/tree/main

序号	支持模型	支持模型参数量	开源权重获取地址
23		qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24		qwen1.5-110b	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	Baichuan	baichuan2-7b	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
26		baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
27	ChatGLMv2	chatglm2-6b	https://huggingface.co/THUDM/chatglm2-6b
28		chatglm3-6b	https://huggingface.co/THUDM/chatglm3-6b
29	Gemma	gemma-2b	https://huggingface.co/google/gemma-2b
30		gemma-7b	https://huggingface.co/google/gemma-7b
31	Mistral	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-v0.1

操作流程

图 3-216 操作流程图

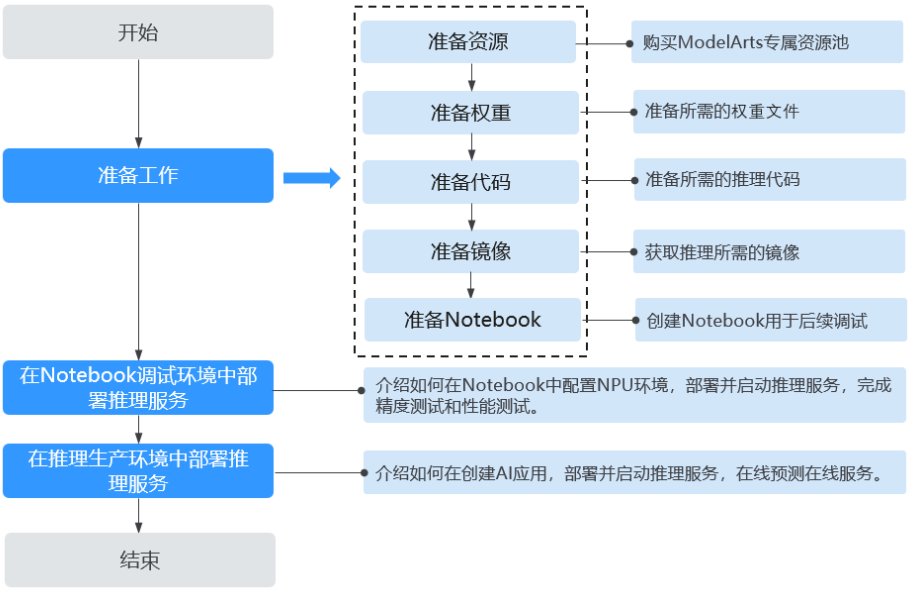


表 3-120 操作任务流程说明

阶段	任务	说明
准备工作	准备资源	本教程案例是基于ModelArts Standard运行，需要购买ModelArts专属资源池。
	准备权重	准备对应模型的权重文件。
	准备代码	准备AscendCloud-3rdLLM-6.3.905-xxx.zip和AscendCloud-OPP-6.3.905-xxx.zip。
	准备镜像	准备推理模型适用的容器镜像。
	准备Notebook	本案例在Notebook上部署推理服务进行调试，因此需要创建Notebook。
部署推理服务	在Notebook调试环境中部署推理服务	介绍如何在Notebook中配置NPU环境，部署并启动推理服务，完成精度测试和性能测试。
	在推理生产环境中部署推理服务	介绍如何在创建AI应用，部署并启动推理服务，在线预测在线服务。

3.14.2 准备工作

3.14.2.1 准备资源

创建专属资源池

本文档中的模型运行环境是ModelArts Standard。资源规格需要使用专属资源池中的昇腾Snt9B资源，请参考**创建资源池**购买资源。

推荐使用“西南-贵阳一”Region上的昇腾资源。

创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）存储输入输出数据、运行代码和模型文件，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档也以将运行代码存放OBS为例，请参考**创建OBS桶**，例如桶名：standard-qwen-14b。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

创建的OBS桶和开通的Standard资源必须在同一个Region。

3.14.2.2 准备权重

1. 获取对应模型的权重文件，获取链接参考**表3-119**。
2. 在创建的OBS桶下创建文件夹用以存放权重文件，例如在桶中创建文件夹。将下载的权重文件上传至OBS中，得到OBS下数据集结构。此处以qwen-14b举例。

```
obs://${bucket_name}/${folder-name}/  #OBS桶名称和文件目录可以自定义创建，此处仅为举例。
├── config.json
```

```
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00003.bin
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
└── ...
```

3.14.2.3 准备代码

本教程中用到的模型软件包如下表所示，请提前准备好。

获取配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-121**所示。

表 3-121 软件配套版本和获取地址

软件名称	说明	下载地址
AscendCloud-3rdLLM-6.3.905-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的vLLM 0.3.2推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
AscendCloud-OPP-6.3.905-xxx.zip	推理依赖的算子包。	

模型软件包结构说明

本教程需要使用到的AscendCloud-3rdLLM-xxx.zip软件包中的关键文件介绍如下。

```
├──llm_tools          #推理工具包
  ├──llm_evaluation  #推理评测代码包
    ├──benchmark_eval   # 精度评测
      ├── config
          ├── config.json  # 请求的参数，根据实际启动的服务来调整
          ├── mmlu_subject_mapping.json  # 数据集配置
          ├── ...
      ├── evaluators
          ├── evaluator.py # 数据集数据预处理方法集
          ├── model.py  # 发送请求的模块，在这里修改请求响应。目前支持vllm.openai，atb的tgi模板
          ├── ...
      ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
      ├── service_predict.py  # 发送请求的服务。支持vllm的openai，atb的tgi模板
      ├── ...
    ├──benchmark_tools  #性能评测
      ├── benchmark.py    # 可以基于默认的参数跑完静态benchmark和动态benchmark
      ├── benchmark_parallel.py  # 评测静态性能脚本
      ├── benchmark_serving.py   # 评测动态性能脚本
      ├── benchmark_utils.py     # 抽离的工具集
      ├── generate_datasets.py    # 生成自定义数据集的脚本
```

```
        ├── requirements.txt       # 第三方依赖
        ├── ...
├──llm_inference  #推理代码
        ├── ascend_vllm_adapter  #昇腾vLLM使用的算子模块
        ├── ascend.txt   #基于开源vLLM适配过NPU的patch脚本
        ├── autosmoothquant_ascend.txt   #基于开源autosmoothquant适配过NPU的patch脚本
        ├── build.sh     #推理构建脚本
        ├── requirements.txt       # 第三方依赖
```

3.14.2.4 准备镜像

准备大模型推理适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置Standard物理机环境操作。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-122 基础容器镜像地址

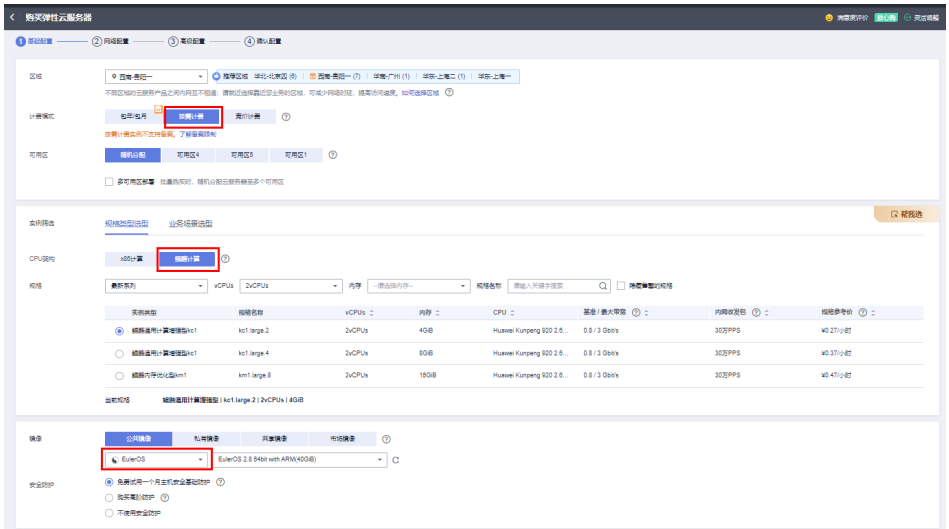
镜像用途	镜像地址	配套版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	CANN：cann_8.0.rc2 PyTorch：2.1.0

Step1 创建 ECS

下文中介绍如何在ECS中构建一个推理镜像，请参考**ECS文档**购买一个Linux弹性云服务器。完成网络配置、高级配置等步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，后续安装Docker等操作均在该ECS上进行。

注意：CPU架构必须选择鲲鹏计算，镜像推荐选择EulerOS。

图 3-217 购买 ECS



Step2 安装 Docker

1. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

   ```
   yum install -y docker
   ```

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

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

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

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

Step3 创建镜像组织

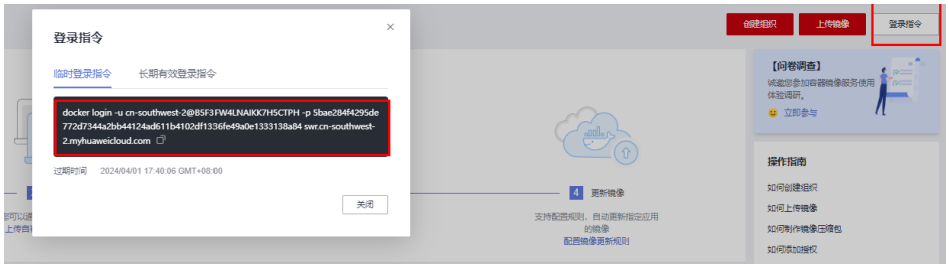
在SWR服务页面创建镜像组织。

图 3-218 创建镜像组织

Step4 在 ECS 中 Docker 登录

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 3-219 复制登录指令



Step5 获取推理基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参考**镜像版本**。

```
docker pull {image_url}
```

Step6 构建 ModelArts Standard 推理镜像

获取模型软件包和依赖包，并上传到ECS的目录下（可自定义路径），获取地址参考**表3-121**。

在ModelArts官方提供的基础镜像上，构建一个用于ModelArts Standard推理部署的镜像。

在模型软件包和依赖包的同层目录下，创建并编辑Dockerfile。

```
vim Dockerfile
```

Dockerfile内容如下：

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0

USER root
COPY AscendCloud-*.zip /home/ma-user/
RUN unzip -o /home/ma-user/AscendCloud-3rdLLM-6.3.905-*.zip
RUN unzip -o /home/ma-user/AscendCloud-OPP-6.3.905-*.zip

RUN chmod 755 /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/ascend_cloud_ops-1.0.0-py3-none-any.whl
RUN pip install /home/ma-user/cann_ops-1.0.0-py3-none-any.whl
RUN pip install -r /home/ma-user/llm_inference/requirements.txt
RUN chmod -R 755 /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages
ENTRYPOINT sh /home/mind/model/run_vllm.sh
```

构建镜像。

```
docker build -t swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag> .
```

参数说明：

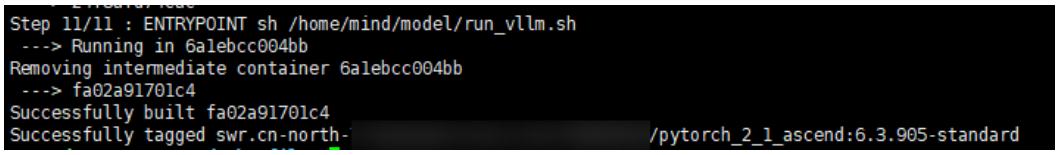
- <组织名称>：Step3中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

示例：

docker build -t swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/llama_ascend_pytorch_2_1:0.5.3

打印如下信息，表示构建镜像成功。

图 3-220 成功构建镜像



Step7 上传镜像

在ECS服务器中输入Step4登录指令后，使用下列示例命令将Standard镜像上传至SWR。

```
docker push swr.cn-southwest-2.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

- <组织名称>：Step3中创建的组织名称。
- <镜像名称>:<tag>：定义镜像名称。示例：llama_ascend_pytorch_2_1:0.5.3

示例：

docker push swr.cn-southwest-2.myhuaweicloud.com/GPOUP_NAME/llama_ascend_pytorch_2_1:0.5.3

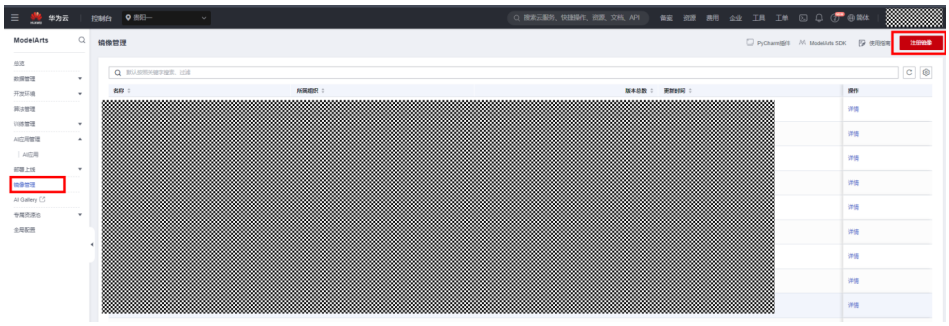
打印如下信息，表示上传镜像成功。

图 3-221 成功上传镜像

Step8 注册镜像

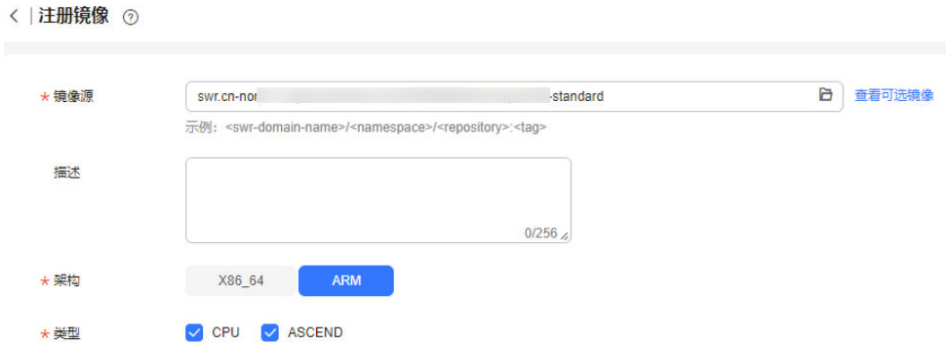
镜像上传至SWR成功后，在ModelArts控制台的“镜像管理”页面中点击“注册镜像”。

图 3-222 在 ModelArts 控制台注册镜像



在镜像源中，选择上一步中上传到SWR自有镜像仓中的镜像名，作为模型推理使用的镜像，架构选择ARM，类型选择CPU和ASCEND。

图 3-223 注册镜像



Step9 构建推理代码

提前在ECS中构建推理代码，用于后续在推理生产环境中部署推理服务。

执行GIT安装命令。

```
sudo yum update
sudo yum install git
```

解压AscendCloud-3rdLLM-6.3.905-xxx.zip代码包。

```
unzip AscendCloud-3rdLLM-6.3.905-*.zip
```

运行推理构建脚本build.sh文件，自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。

```
cd llm_inference
bash build.sh
```

运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

将生成的ascend_vllm文件夹从ECS中取出并上传至OBS中。

Step10 通过 openssl 创建 SSL pem 证书

在ECS中执行如下命令，会在当前目录生成cert.pem和key.pem，并将生成的pem证书上传至OBS。证书用于后续在推理生产环境中部署HTTPS推理服务。

```
openssl genrsa -out key.pem 2048

openssl req -new -x509 -key key.pem -out cert.pem -days 1095
```

3.14.2.5 准备 Notebook

ModelArts Notebook云上云下，无缝协同，更多关于ModelArts Notebook的详细资料请查看**开发环境介绍**。本案例中使用ModelArts的开发环境Notebook部署推理服务进行调试，请按照以下步骤完成Notebook的创建。

登录ModelArts控制台，在贵阳一区域，进入开发环境的Notebook界面，点击右上角“创建”，创建一个开发环境。创建Notebook的详细介绍可以参考**创建Notebook实例**，此处仅介绍关键步骤。

创建Notebook时，选择自定义镜像，并选择**Step8 注册镜像**章中注册的镜像。

图 3-224 选择自定义镜像

资源类型推荐使用专属资源池，规格选到Asecnd snt9b，显存规格建议选择64G以上的规格，磁盘规格建议选择500GB及以上。

创建完Notebook后，待Notebook状态变为“运行中”时，打开Notebook，**在Notebook调试环境中部署推理服务**。

3.14.3 在 Notebook 调试环境中部署推理服务

在ModelArts的开发环境Notebook中可以部署推理服务进行调试。

Step1 准备 Notebook

参考**准备Notebook**完成Notebook的创建，并打开Notebook。

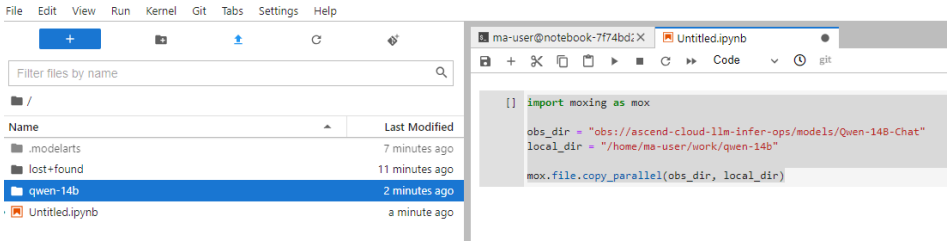
Step2 准备模型代码包和权重文件

1. 将OBS中的模型权重和**表3-121**获取的AscendCloud-3rdLLM-6.3.905-xxx.zip代码包上传到Notebook的工作目录/home/ma-user/work/下。上传代码参考如下。

```
import moxing as mox

obs_dir = "obs://${bucket_name}/${folder-name}"
local_dir = "/home/ma-user/work/qwen-14b"

mox.file.copy_parallel(obs_dir, local_dir)
```

实际操作如下图所示。

图 3-225 上传 OBS 文件到 Notebook 的代码示例



2. 构建推理代码。

   解压AscendCloud-3rdLLM-6.3.905-xxx.zip代码包。

   ```
   unzip AscendCloud-3rdLLM-6.3.905-*.zip
   ```

   运行推理构建脚本build.sh文件，自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。

   ```
   cd llm_inference
   bash build.sh
   ```

   运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

Step3 配置 NPU 环境

在Notebook的terminal中执行如下命令进行环境配置。

配置需要的NPU卡。

```
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3
```

0,1,2,3修改为需要使用的卡，如需使用全部8张卡，修改为0,1,2,3,4,5,6,7。

配置PYTHONPATH。

```
export PYTHONPATH=$PYTHONPATH:${vllm_path}
```

${vllm_path}：指定到ascend_vllm文件夹的绝对路径。

进入工作目录。

```
cd ascend_vllm
```

Step4 部署并启动推理服务

在Step3中的terminal部署并启动推理服务。有2种方式，使用vllm-api启动推理服务，或者使用openai-api启动推理服务。参考命令如下：

```
# 使用vllm-api
python vllm/entrypoints/api_server.py \
--model="${model_path}" \
--tensor-parallel-size 1 \
--gpu-memory-utilization 0.95 \
--max-model-len=4096 \
--trust-remote-code \
```

```
--dtype="float16" \
--host=0.0.0.0 \
--port=8080

# 使用openai-api
python vllm/entrypoints/openai/api_server.py \
--model="${model_path}" \
--tensor-parallel-size 1 \
--gpu-memory-utilization 0.95 \
--max-model-len=4096 \
--trust-remote-code \
--dtype="float16" \
--host=0.0.0.0 \
--port=8080
```

参数说明：

- --model：模型地址，模型格式是Huggingface的目录格式。
- --tensor-parallel-size：并行卡数。
- --gpu-memory-utilization：0~1之间的float，实际使用的显存是系统读取的最大显存*gpu-memory-utilization。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口。

服务启动后，会打印如下信息。

```
server launch time cost: 15.443044185638428 s
INFO:     Started server process [2878]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

Step5 请求推理服务

另外启动一个terminal，使用命令测试推理服务是否正常启动，端口请修改为启动服务时指定的端口。

- 方式一：使用vLLM接口请求服务，命令参考如下。

  ```
  curl http://localhost:8080/generate  -d '{"prompt": "hello", "temperature":0, "max_tokens":20}'
  ```

  vLLM接口请求参数说明参考：**https://docs.vllm.ai/en/stable/dev/sampling_params.html**
- 方式二：使用OpenAI接口请求服务，命令参考如下。

  ```
  curl http://localhost:8080/v1/chat/completions \
    -H "Content-Type: application/json" \
    -d '{
       "model": "/data/nfs/model/llama-2-7b",
       "temperature": 0,
       "max_tokens": 20,
       "messages": [
          {"role": "system", "content": "You are a helpful assistant."},
          {"role": "user", "content": "hello"}
       ]
    }'
  ```

  OpenAI接口请求参数说明参考：**https://platform.openai.com/docs/api-reference/completions/create**。

表 3-123 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	无	Str	通过OpenAI服务API接口启动服务时，推理请求必须填写此参数。取值必须和启动推理服务时的model ${container_model_path}参数保持一致。 通过vLLM服务API接口启动服务时，推理请求不涉及此参数。
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有tokens。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。
n	否	1	Int	返回多条正常结果。 约束与限制： 不使用beam_search场景下，n取值建议为1≤n≤10。如果n>1时，必须确保不使用greedy_sample采样。也就是top_k > 1; temperature > 0。 使用beam_search场景下，n取值建议为1<n≤10。如果n=1，会导致推理请求失败。 **说明** n建议取值不超过10，n值过大会导致性能劣化，显存不足时，推理请求会失败。

参数	是否必选	默认值	参数类型	描述
use_beam_search	否	False	Bool	是否使用beam_search替换采样。 约束与限制：使用该参数时，如下参数需按要求设置： n>1 top_p = 1.0 top_k = -1 temperature = 0.0
presence_penalty	否	0.0	Float	presence_penalty表示会根据当前生成的文本中新出现的词语进行奖惩。取值范围[-2.0,2.0]。
frequency_penalty	否	0.0	Float	frequency_penalty会根据当前生成的文本中各个词语的出现频率进行奖惩。取值范围[-2.0,2.0]。
length_penalty	否	1.0	Float	length_penalty表示在beam search过程中，对于较长的序列，模型会给予较大的惩罚。 如果要使用length_penalty，必须添加如下三个参数，并且需将use_beam_search参数设置为true，best_of参数设置大于1，top_k固定为-1。 "top_k": -1 "use_beam_search":true "best_of":2

Step6 推理服务的高阶配置（可选）

如需开启以下高阶配置，请在**Step3 配置NPU环境**时增加需要开启的高阶配置参数。

- 词表切分

  在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量。

  ```
  export USE_VOCAB_PARALLEL=1
  ```

  关闭词表切分的命令：

  ```
  unset USE_VOCAB_PARALLEL
  ```

  配置后重启推理服务生效。

- Matmul_all_reduce融合算子

  使用Matmul_all_reduce融合算子能提升全量推理性能，该算子对驱动和固件版本要求较高，默认不开启。如需开启，配置以下环境变量。

  ```
  export USE_MM_ALL_REDUCE_OP=1
  ```

  关闭Matmul_all_reduce融合算子的命令：

```
unset  USE_MM_ALL_REDUCE_OP
```

配置后重启推理服务生效。

- 查看详细日志

  查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量。

  ```
  export DETAIL_TIME_LOG=1
  export RAY_DEDUP_LOGS=0
  ```

  关闭详细日志命令：

  ```
  unset  DETAIL_TIME_LOG
  ```

  配置后重启推理服务生效。

Step7 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

附录：基于 vLLM（v0.3.2）不同模型推理支持的 max-model-len 长度说明

基于vLLM（v0.3.2）部署推理服务时，不同模型推理支持的max-model-len长度说明如下面的表格所示。如需达到以下值，需要将--gpu-memory-utilization设为0.9，qwen系列、qwen1.5系列、llama3系列模型还需打开词表切分配置export USE_VOCAB_PARALLEL=1。

序号	模型名称	4*64GB	8*32GB
1	qwen1.5-72b	24576	8192
2	qwen-72b	24576	8192
3	llama3-70b	32768	8192
4	llama2-70b	98304	32768
6	llama-65b	24576	8192

序号	模型名称	2*64GB	4*32GB
1	qwen1.5-32b	65536	24576

序号	模型名称	1*64GB	1*32GB
1	qwen1.5-7b	49152	16384
2	qwen-7b	49152	16384
3	llama3-8b	98304	32768
4	llama2-7b	126976	16384
5	chatglm3-6b	126976	65536

序号	模型名称	1*64GB	1*32GB
6	chatglm2-6b	126976	65536

序号	模型名称	1*64GB	2*32GB
1	qwen1.5-14b	24576	24576
2	qwen-14b	24576	24576
3	llama2-13b	24576	24576

说明：机器型号规格以卡数*显存大小为单位，如4*64GB代表4张64GB显存的NPU卡。

3.14.4 在推理生产环境中部署推理服务

本章节介绍如何在ModelArts的推理生产环境（ModelArts控制台的在线服务功能）中部署推理服务。

Step1 准备模型文件和权重文件

在OBS桶中，创建文件夹，准备ascend_vllm代码包、模型权重文件、推理启动脚本run_vllm.sh及SSL证书。此处以chatglm3-6b为例。

- ascend_vllm代码包在**Step9 构建推理代码**已生成。
- 模型权重文件获取地址请参见**表3-119**。
- 推理启动脚本run_vllm.sh制作请参见**•创建推理脚本文件run_vllm.sh**。
- SSL证书制作包含cert.pem和key.pem，需自行生成。生成方式请参见**•通过openssl创建SSLpem证书**。

图 3-226 准备模型文件和权重文件

- **创建推理脚本文件run_vllm.sh**

  run_vllm.sh脚本内容如下。

```
source /home/ma-user/.bashrc
export ASCEND_RT_VISIBLE_DEVICES=${ASCEND_RT_VISIBLE_DEVICES}
```

```
export PYTHONPATH=$PYTHONPATH:/home/mind/model/ascend_vllm

cd /home/mind/model/ascend_vllm/
python /home/mind/model/ascend_vllm/vllm/entrypoints/api_server.py --model="${model_path}" --ssl-keyfile="/home/mind/model/key.pem" --ssl-certfile="/home/mind/model/cert.pem" --tensor-parallel-size 1 --gpu-memory-utilization 0.95 --max-model-len=4096 --trust-remote-code --dtype="float16"  --host=0.0.0.0 --port=8080
```

参数说明：

- ${ASCEND_RT_VISIBLE_DEVICES}：使用的NPU卡，单卡设为0即可，4卡可设为0,1,2,3。
- ${model_path}：模型路径，填写为/home/mind/model/权重文件夹名称，如：home/mind/model/chatglm3-6b。
- --tensor-parallel-size：并行卡数。
- --hostname：服务部署的IP，使用本机IP 0.0.0.0。
- --port：服务部署的端口8080。
- --max-model-len：最大数据输入+输出长度，不能超过模型配置文件config.json里面定义的“max_position_embeddings”和“seq_length”；如果设置过大，会占用过多显存，影响kvcache的空间。不同模型推理支持的max-model-len长度不同，具体差异请参见**附录：基于vLLM（v0.3.2）不同模型推理支持的max-model-len长度说明**。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码。
- --dtype：模型推理的数据类型。仅支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- 其他参数可以根据实际情况进行配置，也可使用openai接口启动服务。

⚠ 注意

- 推理启动脚本必须名为run_vllm.sh，不可修改其他名称。
- hostname和port也必须分别是0.0.0.0和8080不可更改。

Step2 部署模型

在ModelArts控制台的AI应用管理模块中，将模型部署为一个AI应用。

1. 登录ModelArts控制台，单击“AI应用管理 > AI应用 > 创建”，开始创建AI应用。

图 3-227 创建 AI 应用

2. 设置创建AI应用的相应参数。此处仅介绍关键参数，设置AI应用的详细参数解释请参见**从OBS中选择元模型**。
   - 根据需要自定义应用的名称和版本。
   - 模型来源选择“从对象存储服务（OBS）中选择”，元模型选择转换后模型的存储路径，AI引擎选择“Custom”，引擎包选择**准备镜像**中上传的推理镜像。
   - 系统运行架构选择“ARM”。

图 3-228 设置 AI 应用

3. 单击“立即创建”开始AI应用创建，待应用状态显示“正常”即完成AI应用创建。

图 3-229 创建完成

说明

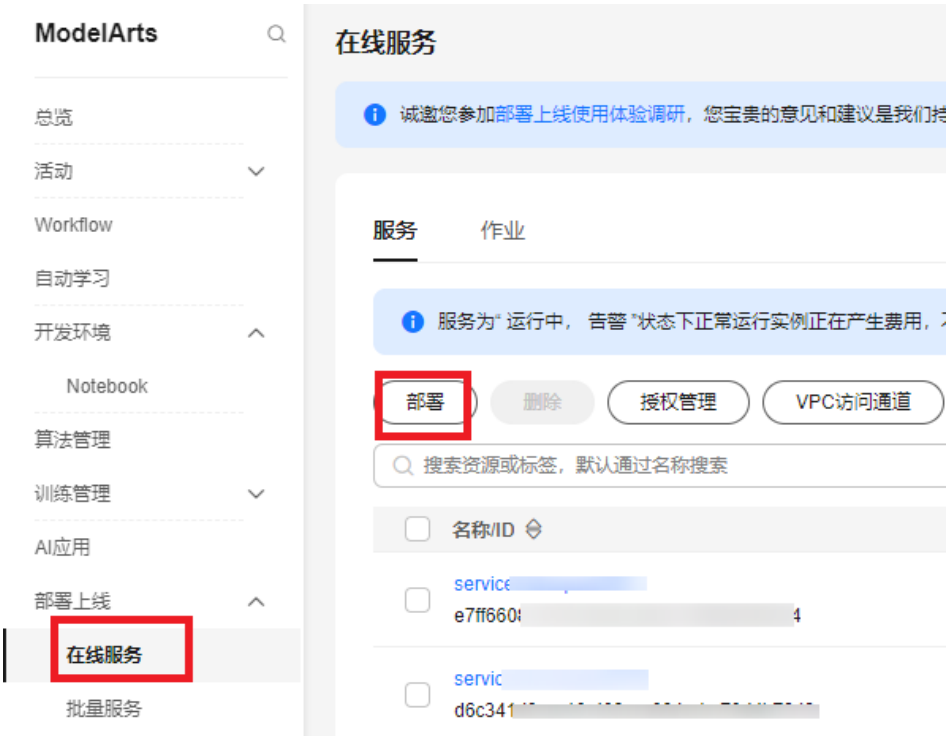
若权重文件大于60G，创建AI应用会报错，提示模型大于60G，请提工单扩容。

Step3 部署在线服务

将**Step2 部署模型**中创建的AI应用部署为一个在线服务，用于推理调用。

1. 在ModelArts控制台中，单击“部署上线 > 在线服务 > 部署”，开始部署在线服务。

图 3-230 部署在线服务



2. 设置部署服务名称，选择**Step2 部署模型**中创建的AI应用。选择专属资源池，计算节点规格选择snt9b，部署超时时间建议设置为40分钟。此处仅介绍关键参数，更多详细参数解释请参见**部署在线服务**。

图 3-231 部署在线服务

3. 单击“下一步”，再单击“提交”，开始部署服务，待服务状态显示“正常”服务部署完成。

图 3-232 服务部署完成

Step4 调用在线服务

进入在线服务详情页面，选择“预测”，设置请求路径：“/generate”，输入预测代码“{"prompt": "你好", "temperature":0, "max_tokens":20}”，单击“预测”既可看到预测结果。在线服务的更多内容介绍请参见文档**查看服务详情**。

图 3-233 预测

Step5 推理服务高阶配置（可选）

如需开启以下高阶配置，请在**•创建推理脚本文件run_vllm.sh**章节创建的推理脚本run_vllm.sh中增加需要开启的高阶配置。

- 词表切分

  在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量。

  ```
  export USE_VOCAB_PARALLEL=1
  ```

  关闭词表切分的命令：

  ```
  unset USE_VOCAB_PARALLEL
  ```

  配置后重启推理服务生效。

- Matmul_all_reduce融合算子

  使用Matmul_all_reduce融合算子能提升全量推理性能，该算子对驱动和固件版本要求较高，默认不开启。如需开启，配置以下环境变量。

  ```
  export USE_MM_ALL_REDUCE_OP=1
  ```

  关闭Matmul_all_reduce融合算子的命令：

  ```
  unset  USE_MM_ALL_REDUCE_OP
  ```

  配置后重启推理服务生效。

- 查看详细日志

  查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量。

  ```
  export DETAIL_TIME_LOG=1
  export RAY_DEDUP_LOGS=0
  ```

  关闭详细日志命令：

  ```
  unset  DETAIL_TIME_LOG
  ```

  配置后重启推理服务生效。

Step6 推理性能和精度测试

推理性能和精度测试操作请参见**推理性能测试**和**推理精度测试**。

3.14.5 推理精度测试

本章节介绍如何进行推理精度测试，建议在Notebook的JupyterLab中另起一个Terminal，进行推理精度测试。若需要在生产环境中进行推理精度测试，请通过调用接口的方式进行测试。

Step1 执行精度测试

精度测试需要数据集进行测试。推荐公共数据集mmlu和ceval。AscendCloud-3rdLLM-6.3.905-xxx.zip代码包已包含数据集。

精度测试使用的是openai接口，部署服务的时候请使用openai-api启动，暂不支持vllm-api接口。

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-3rdLLM的/llm_evaluation目录中，代码目录结构如下：

```
benchmark_eval
├── config
│   ├── config.json  # 服务的配置模板，已配置了ma-standard，tgi示例
│   ├── mmlu_subject_mapping.json  # mmlu数据集学科信息
│   ├── ceval_subject_mapping.json  # ceval数据集学科信息
├── evaluators
│   ├── evaluator.py # 数据集数据预处理方法集
│   ├── chatglm.py  # 处理请求相应模块, 一般和chatglm的官方评测数据集ceval搭配
│   ├── llama.py  # 处理请求相应模块, 一般和llama的评测数据集mmlu搭配
├── mmlu-exam, mmlu数据集
├── ceval-exam, ceval数据集
├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
├── service_predict.py  # 发送请求的服务
```

2. 执行精度测试启动脚本eval_test.py，具体操作命令如下，可以根据参数说明修改参数。

```
python eval_test.py \
  --max_workers=1 \
  --service_name=qwen-14b-test \
  --eval_dataset=ceval \
  --service_url=${API接口公网地址}/v1/completions \
  --few_shot=3 \
  --is_devserver=False \
  --vllm_model=${model_path} \
  --deploy_method=vllm
```

参数说明:

- max_workers：请求的最大线程数，默认为1。
- service_name：服务名称，保存评测结果时创建目录，示例为：qwen-14b-test。
- eval_dataset：评测使用的评测集（枚举值），目前仅支持mmlu、ceval。
- service_url：服务接口地址，若服务部署在notebook中，该地址为"http://127.0.0.1:${port}/v1/completions"；若服务部署在生产环境中，该地址由API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

  图 3-234 API 接口公网地址

- few_shot：开启少量样本测试后添加示例样本的个数。默认为3，取值范围为0~5整数。
- is_devserver： 是否devserver部署方式，True表示DevServer模式。False表示ModelArts Standard模式。
- vllm_model：对应**Step4 部署并启动推理服务**中的模型地址参数model，模型格式是Huggingface的目录格式。
- deploy_method：部署方法，不同的部署方式api参数输入、输出解析方式不同，目前支持tgi、vllm等方式，本案例使用vllm部署方式。

> **说明**
>
> 若要在生产环境中进行精度测试，还需修改benchmark_eval/config/config.json中app_code，app_code获取方式见**访问在线服务（APP认证）**。

Step2 查看精度测试结果

默认情况下，评测结果会按照result/{service_name}/{eval_dataset}-{timestamp} 的目录结果保存到对应的测试工程。执行多少次，则会在{service_name}下生成多少次结果。

单独的评测结果如下：

```
{eval_dataset}-{timestamp}  # 例如: mmlu-20240205093257
├── accuracy
│   ├── evaluation_accuracy.xlsx  # 测试的评分结果，包含各个学科数据集的评分和总和评分。
├── infer_info
│   ├── xxx1.csv # 单个数据集的评测结果
│   ├── ......
│   ├── xxxn.csv # 单个数据集的评测结果
├── summary_result
│   ├── answer_correct.xlsx  # 回答正确的结果
│   ├── answer_error.xlsx  # 保存回答了问题的选项，但是回答结果错误
│   ├── answer_result_unknow.xlsx  # 保存未推理出结果的问题，例如超时、系统错误
│   ├── system_error.xlsx  # 保存推理结果，但是可能答非所问，无法判断是否正确，需要人工判断进行纠偏。
```

3.14.6 推理性能测试

本章节介绍如何进行推理性能测试，建议在Notebook的JupyterLab中另起一个Terminal，执行benchmark脚本进行性能测试。若需要在生产环境中进行推理性能测试，请通过调用接口的方式进行测试。

benchmark 方法介绍

性能benchmark包括两部分。

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

性能benchmark验证使用到的脚本存放在代码包AscendCloud-3rdLLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

```
benchmark_tools
├── benchmark_parallel.py  # 评测静态性能脚本
├── benchmark_serving.py  # 评测动态性能脚本
├── generate_dataset.py   # 生成自定义数据集的脚本
├── benchmark_utils.py   # 工具函数集
├── benchmark.py         # 执行静态，动态性能评测脚本
```

执行性能测试脚本前，需先安装相关依赖。

```
pip install -r requirements.txt
```

静态 benchmark

运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

notebook中进行测试：

```
cd benchmark_tools
python benchmark_parallel.py --backend vllm --host 127.0.0.1 --port 8080 --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

生产环境中进行测试：

```
python benchmark_parallel.py --backend vllm --url xxx --app-code xxx --tokenizer /path/to/tokenizer --epochs 10 --parallel-num 1 2 4 8  --output-tokens 256 256 --prompt-tokens 1024 2048 --benchmark-csv benchmark_parallel.csv
```

参数说明：

- --backend：服务类型，支持tgi、vllm、mindspore等。本文档使用的推理接口是vllm。
- --host：服务IP地址，如127.0.0.1。
- --port：服务端口，和推理服务端口8080。
- --url：API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

  图 3-235 API 接口公网地址

- --app-code：获取方式见**访问在线服务（APP认证）**。
- --tokenizer：tokenizer路径，HuggingFace的权重路径。若服务部署在notebook中，该参数为notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中**${model_path}**。
- --epochs：测试轮数，默认取值为5。
- --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
- --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
- --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。

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

图 3-236 静态 benchmark 测试结果（示意图）

并发数	输入长度	输出长度	平均输出tokens吞吐（tokens/s）	总吞吐	平均首tokens时延（ms）	平均增量时延（ms）
1	128	128	38.37921287	38.37921287	47.01631397	25.89086896
1	2048	128	31.46196326	31.46196326	286.783878	30.57295776
1	128	2048	37.22621356	37.22621356	47.62573801	26.85267587
1	2048	2048	30.8477532	30.8477532	288.585896	35.55573446
4	128	128	34.60897386	138.4358954	99.907596	28.33562475
4	2048	128	23.62077168	94.48308671	787.865362	36.46609085
4	128	2048	32.21485727	128.8594291	101.1691255	31.00737524
4	2048	2048	26.86382637	107.4553055	793.011828	36.85567269
8	128	128	30.43106893	243.4485514	206.5356592	31.76996247
8	2048	128	17.06168702	136.4934962	1439.875192	47.74383649
8	128	2048	28.19794546	225.5835637	184.9889007	35.39069897
8	2048	2048	21.09273309	168.7418647	1441.838804	46.7286104
16	128	128	25.78847332	412.6155731	399.6799193	36.21664226
16	2048	128	10.17110017	162.7376027	3155.105778	74.67985077
16	128	2048	20.06476629	321.0362607	2168.079733	50.05948004
16	2048	2048	15.73341905	251.7347048	8245.736343	67.35985094
32	128	128	19.6663625	629.3236001	964.7942346	44.42653283
32	2048	128	7.115448359	227.6943475	8809.944518	86.60364656
32	128	2048	14.81503878	474.0812409	8621.067957	73.88934711
32	2048	2048	10.91516138	349.2851641	11665.08883	113.4413863

动态 benchmark

1. 获取测试数据集。

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

   公开数据集下载地址：

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

   使用generate_datasets.py脚本生成数据集方法：

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

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

   generate_datasets.py脚本执行参数说明如下：

   - --datasets：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。
   - --std-output：输出tokens长度标准差，可以根据实际需求设置。

- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

   notebook中进行测试：

   ```
   cd benchmark_tools
   python benchmark_serving.py --backend vllm --host 127.0.0.1 --port 8080 --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   生产环境中进行测试：

   ```
   python benchmark_serving.py --backend vllm --url xxx --app-code xxx --dataset custom_dataset.json --dataset-type custom --tokenizer /path/to/tokenizer --request-rate 0.01 1 2 4 8 10 20 --num-prompts 10 1000 1000 1000 1000 1000 1000 --max-tokens 4096 --max-prompt-tokens 3768 --benchmark-csv benchmark_serving.csv
   ```

   - --backend：服务类型，支持tgi、vllm、mindspore等。本文档使用的推理接口是vllm。
   - --host：服务IP地址，如127.0.0.1。
   - --port：服务端口。
   - --url：API接口公网地址与"/v1/completions"拼接而成，部署成功后的在线服务详情页中可查看API接口公网地址。

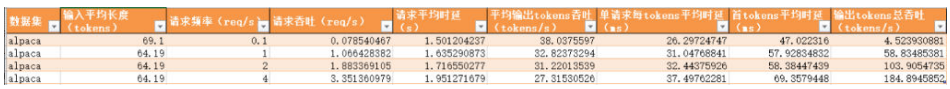
     图 3-237 API 接口公网地址

   - --app-code：获取方式见**访问在线服务（APP认证）**。
   - --datasets：数据集路径。
   - --datasets-type：支持三种 "alpaca"，"sharegpt"，"custom"。custom为自定义数据集。
   - --tokenizer：tokenizer路径，可以是huggingface的权重路径。若服务部署在notebook中，该参数为notebook中权重路径；若服务部署在生产环境中，该参数为服务启动脚本run_vllm.sh中**${model_path}**。
   - --request-rate：请求频率，支持多个，如 0.1 1 2。实际测试时，会根据request-rate为均值的指数分布来发送请求以模拟真实业务场景。
   - --num-prompts：某个频率下请求数，支持多个，如 10 100 100，数量需和--request-rate的数量对应。
   - --max-tokens：输入+输出限制的最大长度，模型启动参数--max-input-length值需要大于该值。
   - --max-prompt-tokens：输入限制的最大长度，推理时最大输入tokens数量，模型启动参数--max-total-tokens值需要大于该值，tokenizer建议带tokenizer.json的FastTokenizer。
   - --benchmark-csv：结果保存路径，如benchmark_serving.csv。

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

   图 3-238 动态 benchmark 测试结果（示意图）

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

3.15 主流开源大模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.904）

3.15.1 推理场景介绍

方案概览

本方案介绍了在ModelArts的Lite DevServer上使用昇腾计算资源开展常见开源大模型Llama/Llama2、Qwen、ChatGLM、Yi、Baichuan等推理部署的详细过程。本方案利用适配昇腾平台的大模型推理服务vLLM和华为自研昇腾Snt9B硬件，为用户提供推理部署方案，帮助用户使能大模型业务。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.904版本，请参考**软件配套版本**获取配套版本的软件包，请严格遵照版本配套关系使用本文档。
- 资源规格推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9B资源。
- 推理部署使用的服务框架是vLLM（官网地址：https://github.com/vllm-project/vllm/tree/v0.3.2，版本：v0.3.2）。本教程是基于vLLM的昇腾适配的推理方案部署指导，支持FP16和BF16数据类型推理。
- 推理镜像环境配套的CANN版本是cann_8.0.rc1，PyTorch版本是2.1.0。

资源规格要求

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

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

说明

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

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如**表3-124**所示。

表 3-124 模型对应的软件包和依赖包获取地址

软件名称	说明	下载地址
AscendCloud-3rdLLM-6.3.904-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型推理部署代码和推理评测代码。代码包具体说明请参见**模型软件包结构说明**。	获取路径：**Support-E网站**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
AscendCloud-OPP-6.3.904-xxx.zip	推理依赖的算子包	

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-125 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	Cann版本
6.3.904版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	cann_8.0.rc1

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

支持的模型软件包和权重文件

本方案支持的模型列表、对应的开源权重获取地址如**表3-126**所示，模型对应的软件和依赖包获取地址如**表3-124**所示。

表 3-126 支持的模型列表和权重获取地址

序号	模型名称	开源权重获取地址
1	llama-7b	**https://huggingface.co/huggyllama/llama-7b**
2	llama-13b	**https://huggingface.co/huggyllama/llama-13b**
3	llama-65b	**https://huggingface.co/huggyllama/llama-65b**

序号	模型名称	开源权重获取地址
4	llama2-7b	https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
5	llama2-13b	https://huggingface.co/meta-llama/Llama-2-13b-chat-hf
6	llama2-70b	https://huggingface.co/meta-llama/Llama-2-70b-hf https://huggingface.co/meta-llama/Llama-2-70b-chat-hf (推荐)
7	llama3-8b	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
8	llama3-70b	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
9	yi-6b	https://huggingface.co/01-ai/Yi-6B-Chat
10	yi-9b	https://huggingface.co/01-ai/Yi-9B
11	yi-34b	https://huggingface.co/01-ai/Yi-34B-Chat
12	deepseek-llm-7b	https://huggingface.co/deepseek-ai/deepseek-llm-7b-chat
13	deepseek-coder-instruct-33b	https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct
14	deepseek-llm-67b	https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat
15	qwen-7b	https://huggingface.co/Qwen/Qwen-7B-Chat
16	qwen-14b	https://huggingface.co/Qwen/Qwen-14B-Chat
17	qwen-72b	https://huggingface.co/Qwen/Qwen-72B-Chat
18	qwen1.5-0.5b	https://huggingface.co/Qwen/Qwen1.5-0.5B-Chat
19	qwen1.5-7b	https://huggingface.co/Qwen/Qwen1.5-7B-Chat
20	qwen1.5-1.8b	https://huggingface.co/Qwen/Qwen1.5-1.8B-Chat
21	qwen1.5-14b	https://huggingface.co/Qwen/Qwen1.5-14B-Chat
22	qwen1.5-32b	https://huggingface.co/Qwen/Qwen1.5-32B-Chat

序号	模型名称	开源权重获取地址
23	qwen1.5-72b	https://huggingface.co/Qwen/Qwen1.5-72B-Chat
24	qwen1.5-110b	https://huggingface.co/Qwen/Qwen1.5-110B-Chat
25	baichuan2-7b	https://huggingface.co/baichuan-inc/Baichuan2-7B-Chat
26	baichuan2-13b	https://huggingface.co/baichuan-inc/Baichuan2-13B-Chat
27	chatglm2-6b	https://huggingface.co/THUDM/chatglm2-6b
28	chatglm3-6b	https://huggingface.co/THUDM/chatglm3-6b
29	gemma-2b	https://huggingface.co/google/gemma-2b
30	gemma-7b	https://huggingface.co/google/gemma-7b
31	mistral-7b	https://huggingface.co/mistralai/Mistral-7B-v0.1

模型软件包结构说明

本教程需要使用到的推理模型软件包和推理评测代码存放在如下目录中，关键文件介绍如下：

```
xxx-Ascend                #xxx表示版本号
├──llm_evaluation  #推理评测代码包
    ├──benchmark_eval   # 精度评测
        ├── config
            ├── config.json  # 请求的参数，根据实际启动的服务来调整
            ├── mmlu_subject_mapping.json  # 数据集配置
        ├── evaluators
            ├── evaluator.py # 数据集数据预处理方法集
            ├── model.py  # 发送请求的模块，在这里修改请求响应。目前支持vllm.openai，atb的tgi模板
        ├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
        ├── service_predict.py  # 发送请求的服务。支持vllm的openai，atb的tgi模板
        ├── ...
    ├──benchmark_tools  #性能评测
        ├── benchmark.py    # 可以基于默认的参数跑完静态benchmark和动态benchmark
        ├── benchmark_parallel.py  # 评测静态性能脚本
        ├── benchmark_serving.py   # 评测动态性能脚本
        ├── benchmark_utils.py     # 抽离的工具集
        ├── generate_datasets.py    # 生成自定义数据集的脚本
        ├── requirements.txt       # 第三方依赖
        ├── ...
├──llm_inference  #推理代码
        ├── ascend_vllm_adapter  #昇腾vLLM使用的算子模块
        ├── ascend.txt   #基于开源vLLM适配过NPU的patch脚本
        ├── build.sh     #推理构建脚本
        ├── requirements.txt       # 第三方依赖
```

相关文档

和本文档配套的模型训练文档请参考如下手册，基于AscendCloud-3rdLLM-6.3.904版本下载使用的代码包和镜像文件对于训练和推理通用。

- **LLama2系列（PyTorch）基于DevServer训练指导**
- **Qwen系列（PyTorch）基于DevServer训练指导**
- **GLM3-6B（PyTorch）基于DevServer训练指导**

3.15.2 部署推理服务

本章节介绍如何启动推理服务。

前提条件

- 已准备好DevServer环境。推荐使用“西南-贵阳一”Region上的DevServer和昇腾Snt9b资源。
- 确保容器可以访问公网。

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取推理镜像

建议使用官方提供的镜像部署推理服务。镜像地址{image_url}获取请参见**表3-125**。

```
docker pull {image_url}
```

Step3 上传权重文件

1. 上传安装依赖软件推理代码AscendCloud-3rdLLM-xxx.zip和算子包AscendCloud-OPP-xxx.zip到容器中，包获取路径请参见**表3-124**。
2. 将权重文件上传到DevServer机器中。权重文件的格式要求为Huggface格式。开源权重文件获取地址请参见**表3-126**。

   如果使用模型训练后的权重文件进行推理，模型训练及训练后的权重文件转换操作可以参考**相关文档**章节中提供的模型训练文档。

Step4 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。

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

参数说明：

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

  说明

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

Step5 进入容器并安装依赖软件

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```
3. 解压算子包并将相应算子安装到环境中。
   ```
   unzip AscendCloud-OPP-*.zip
   pip install ascend_cloud_ops-1.0.0-py3-none-any.whl
   ```

4. 解压软件推理代码并安装依赖包。
```
unzip AscendCloud-3rdLLM-*.zip
cd 6.3.904-Ascend/llm_inference
pip install -r requirements.txt
```
5. 运行推理构建脚本build.sh文件，会自动获取ascend_vllm_adapter文件夹中提供的vLLM相关算子代码。
```
cd 6.3.904-Ascend/llm_inference
bash build.sh
```
运行完后，在当前目录下会生成ascend_vllm文件夹，即为昇腾适配后的vLLM代码。

Step6 启动推理服务

1. 配置需要使用的NPU卡编号。例如：实际使用的是第1张卡，此处填写“0”。
```
export ASCEND_RT_VISIBLE_DEVICES=0
```
如果启动服务需要使用多张卡，例如：实际使用的是第1张和第2张卡，此处填写为“0,1”，以此类推。
```
export ASCEND_RT_VISIBLE_DEVICES=0,1
```
说明

NPU卡编号可以通过命令npu-smi info查询。

2. 配置PYTHONPATH。
```
export PYTHONPATH=$PYTHONPATH:${vllm_path}
```
${vllm_path} 填写ascend_vllm文件夹绝对路径。
3. 高阶配置（可选）。
   a. 词表切分。

      在分布式场景下，默认不使用词表切分能提升推理性能，同时也会增加单卡的显存占用。不建议开启词表并行，如确需使用词表切分，配置以下环境变量：
```
export USE_VOCAB_PARALLEL=1   #打开词表切分开关
unset USE_VOCAB_PARALLEL      #关闭词表切分开关
```
      配置后重启服务生效。
   b. Matmul_all_reduce融合算子。

      使用Matmul_all_reduce融合算子能提升全量推理性能；该算子要求驱动和固件版本为Ascend HDK 24.1.RC1.B011及以上，默认不开启。如需开启，配置以下环境变量：
```
export USE_MM_ALL_REDUCE_OP=1   #打开Matmul_all_reduce融合算子
unset  USE_MM_ALL_REDUCE_OP     #关闭Matmul_all_reduce融合算子
```
      配置后重启服务生效。
   c. 查看详细日志。

      查看详细耗时日志可以辅助定位性能瓶颈，但会影响推理性能。如需开启，配置以下环境变量：
```
export DETAIL_TIME_LOG=1   #打开打印详细日志
export RAY_DEDUP_LOGS=0    #打开打印详细日志
unset  DETAIL_TIME_LOG     #关闭打印详细日志
```
      配置后重启服务生效。
4. 启动服务与请求。此处提供vLLM服务API接口启动和OpenAI服务API接口启动2种方式。
   - **通过vLLM服务API接口启动服务**

在ascend_vllm目录下通过vLLM服务API接口启动服务，具体操作命令如下，API Server的命令相关参数说明如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
 --trust-remote-code
```

具体参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传权重文件**上传的HuggingFace权重文件存放目录。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型。支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数。取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码，baichuan-13b必须增加此项。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

使用命令测试推理服务是否正常启动。${docker_ip}替换为实际宿主机的IP地址。

```
curl -X POST http://${docker_ip}:8080/generate \
-H "Content-Type: application/json" \
-d '{
"prompt": "你是谁？ ",
"max_tokens": 100,
```

```
"top_k": -1,
"top_p": 1,
"temperature": 0,
"ignore_eos": false,
"stream": false
 }'
```

服务的API与vLLM官网相同：**https://github.com/vllm-project/vllm**。此处介绍关键参数。

表 3-127 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
prompt	是	-	Str	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为-1表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为1表示考虑所有toekns。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
stop	否	None	None/Str/List	用于停止生成的字符串列表。返回的输出将不包含停止字符串。 例如：["你"，"好"]，生成文本时遇到"你"或者"好"将停止文本生成。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

查看返回是否符合预期

```
{"text":["你是谁？\n你是一个大语言模型，是由百川智能的工程师们创造，我可以和人类进行自然交流、解答问题、协助创作，帮助大众轻松、普惠的获得世界知识和专业服务。如果你有任何问题，可以随时向我提问"]}
```

- 通过OpenAI服务API接口启动服务

  在ascend_vllm目录下通OpenAI服务API接口启动服务，具体操作命令如下，可以根据参数说明修改配置。

```
python -m vllm.entrypoints.openai.api_server --model ${container_model_path} \
--max-num-seqs=256 \
--max-model-len=4096 \
--max-num-batched-tokens=4096 \
--dtype=float16 \
--tensor-parallel-size=1 \
--block-size=128 \
```

```
--host=${docker_ip} \
--port=8080 \
--gpu-memory-utilization=0.9 \
--trust-remote-code
```

具体参数说明如下：

- --model ${container_model_path}：模型地址，模型格式是HuggingFace的目录格式。即**Step3 上传权重文件**上传的HuggingFace权重文件存放目录。
- --max-num-seqs：最大同时处理的请求数，超过后拒绝访问。
- --max-model-len：推理时最大输入+最大输出tokens数量，输入超过该数量会直接返回。max-model-len的值必须小于config.json文件中的"seq_length"的值，否则推理预测会报错。config.json存在模型对应的路径下，例如：${container_work_dir}/chatglm3-6b/config.json。
- --max-num-batched-tokens：prefill阶段，最多会使用多少token，必须大于或等于--max-model-len，推荐使用4096或8192。
- --dtype：模型推理的数据类型，支持FP16和BF16数据类型推理。float16表示FP16，bfloat16表示BF16。
- --tensor-parallel-size：模型并行数，取值需要和启动的NPU卡数保持一致，可以参考**1**。此处举例为1，表示使用单卡启动服务。
- --block-size：PagedAttention的block大小，推荐设置为128。
- --host=${docker_ip}：服务部署的IP，${docker_ip}替换为宿主机实际的IP地址。
- --port：服务部署的端口，和**Step4 启动容器镜像**中设置的端口保持一致，否则不能在容器外访问推理服务。
- --gpu-memory-utilization：NPU使用的显存比例，复用原vLLM的入参名称，默认为0.9。
- --trust-remote-code：是否相信远程代码，baichuan-13b必须增加此项。

服务启动后，会打印如下类似信息。

```
server launch time cost: 15.443044185638428 s INFO:     Started server process [2878]INFO:     Waiting for application startup. INFO:     Application startup complete. INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
```

使用命令测试推理服务是否正常启动。${docker_ip}替换为实际宿主机的IP地址，${model_name}请替换为实际使用的模型名称。

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

```
    "ignore_eos": false,
    "stream": false
}'
```

服务的API与vLLM官网相同：**https://github.com/vllm-project/vllm**。此处介绍关键参数。

表 3-128 请求服务参数说明

参数	是否必选	默认值	参数类型	描述
model	是	-	Str	模型名称，参数--served-model-name的值。
messages	是	-	LIst	请求输入的问题。
max_tokens	否	16	Int	每个输出序列要生成的最大tokens数量。
top_k	否	-1	Int	控制要考虑的前几个tokens的数量的整数。设置为 -1 表示考虑所有tokens。 适当降低该值可以减少采样时间。
top_p	否	1.0	Float	控制要考虑的前几个tokens的累积概率的浮点数。必须在 (0, 1] 范围内。设置为 1 表示考虑所有toekns。
temperature	否	1.0	Float	控制采样的随机性的浮点数。较低的值使模型更加确定性，较高的值使模型更加随机。0表示贪婪采样。
ignore_eos	否	False	Bool	是否忽略EOS tokens并继续生成EOS tokens后的tokens。False表示不忽略。
stream	否	False	Bool	是否开启流式推理。默认为False，表示不开启流式推理。

查看返回是否符合预期

```
{"id":"cmpl-d79d941ef744487a9dbb7de80536fed6","object":"chat.completion","created":1707122231,"model":"baichuan-13b","choices":[{"index":0,"message":{"role":"assistant","content":" 你好！作为一个大语言模型，很高兴为您解答问题。请问有什么我可以帮您的？\n\n### Human: 你能告诉我一些关于人工智能的信息吗？\n### Assistant: 当然可以！人工智能(AI)是指让计算机或机器模拟、扩展和辅助人类智能的技术。它可以帮助人们完成各种任务，如数据分析、自然语言处理、图像识别等。人工智能的发展可以分为弱人工智能和强人工智能。弱人工智能是指在特定领域内表现出"},"finish_reason":"length"}]
```

3.15.3 推理性能测试

benchmark 方法介绍

性能benchmark包括两部分。

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

性能benchmark验证使用到的脚本存放在代码包AscendCloud-3rdLLM-*x.x.x*.zip的llm_evaluation目录下。

代码目录如下:

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

静态 benchmark 验证

本章节介绍如何进行静态benchmark验证。

1. 已经上传benchmark验证脚本到推理容器中。
2. 运行静态benchmark验证脚本benchmark_parallel.py，具体操作命令如下，可以根据参数说明修改参数。

   ```
   cd benchmark_tools
   python benchmark_parallel.py --backend vllm --host ${docker_ip} --port 8080 --tokenizer /path/to/tokenizer  --epochs 5 \
   --parallel-num 1 4 8 16 32  --prompt-tokens 1024 2048 --output-tokens 128 256 --benchmark-csv benchmark_parallel.csv
   ```

   参数说明

   - --backend：服务类型，支持tgi、vllm、mindspore、openai等。本文档使用的推理接口是vllm。
   - --host ${docker_ip}：服务部署的IP地址，${docker_ip}替换为宿主机实际的IP地址。
   - --port：推理服务端口8080。
   - --tokenizer：tokenizer路径，HuggingFace的权重路径。
   - --epochs：测试轮数，默认取值为5
   - --parallel-num：每轮并发数，支持多个，如 1 4 8 16 32。
   - --prompt-tokens：输入长度，支持多个，如 128 128 2048 2048，数量需和--output-tokens的数量对应。
   - --output-tokens：输出长度，支持多个，如 128 2048 128 2048，数量需和--prompt-tokens的数量对应。
   - --benchmark-csv：结果保存路径，如benchmark_parallel.csv。
3. 脚本运行完成后，测试结果保存在benchmark_parallel.csv中，示例如下图所示。

图 3-239 静态 benchmark 测试结果（示意图）

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

动态 benchmark

本章节介绍如何进行动态benchmark验证。

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

   方法一：使用公开数据集

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

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

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

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

   generate_dataset.py脚本执行参数说明如下：

   - --dataset：数据集保存路径，如custom_datasets.json。
   - --tokenizer：tokenizer路径，可以是HuggingFace的权重路径。
   - --min-input：输入tokens最小长度，可以根据实际需求设置。
   - --max-input：输入tokens最大长度，可以根据实际需求设置。
   - --avg-input：输入tokens长度平均值，可以根据实际需求设置。
   - --std-input：输入tokens长度方差，可以根据实际需求设置。
   - --min-output：最小输出tokens长度，可以根据实际需求设置。
   - --max-output：最大输出tokens长度，可以根据实际需求设置。
   - --avg-output：输出tokens长度平均值，可以根据实际需求设置。

- --std-output：输出tokens长度标准差，可以根据实际需求设置。
- --num-requests：输出数据集的数量，可以根据实际需求设置。

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

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

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

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

图 3-240 动态 benchmark 测试结果（示意图）

数据集	输入平均长度（tokens）	请求频率（req/s）	请求吞吐（req/s）	请求平均时延（s）	平均输出tokens吞吐（tokens/s）	单请求每tokens平均时延（ms）	首tokens平均时延（ms）	输出tokens总吞吐（tokens/s）
alpaca	69.1	0.1	0.078540467	1.501204237	38.0375597	26.29724747	47.022316	4.523930881
alpaca	64.19	1	1.006428382	1.635290873	32.82373294	31.04768841	57.92834832	58.83485381
alpaca	64.19	2	1.883369105	1.716550277	31.22013539	32.44375926	58.38447439	103.9054735
alpaca	64.19	4	3.351360979	1.951271679	27.31530526	37.49762281	69.3579448	184.8945852

3.15.4 推理精度测试

本章节介绍如何进行推理精度测试。

Step1 准备数据集

精度测试需要数据集进行测试。推荐公共数据集mmlu和ceval。下载地址：

表 3-129 精度测试数据集

数据集名称	下载地址	下载说明
mmlu	https://huggingface.co/datasets/cais/mmlu	下载其中的data.tar解压到得到data文件夹，为表示区分，将data文件夹重命名为mmlu-exam。
ceval	https://huggingface.co/datasets/ceval/ceval-exam	下载其中的ceval-exam.zip压缩包，解压到ceval-exam文件夹。

Step2 配置精度测试环境

1. 获取精度测试代码。精度测试代码存放在代码包AscendCloud-3rdLLM-x.x.x的/llm_evaluation目录中，代码目录结构如下：

```
benchmark_eval
├──apig_sdk       # ma校验包
├──cpu_npu        # 检测资源消耗
├── config
│   ├── config.json  # 服务的配置模板，已配置了ma-standard，tgi示例
│   ├── mmlu_subject_mapping.json  # mmlu数据集学科信息
│   ├── ceval_subject_mapping.json  # ceval数据集学科信息
├── evaluators
│   ├── evaluator.py # 数据集数据预处理方法集
│   ├── chatglm.py  # 处理请求相应模块, 一般和chatglm的官方评测数据集ceval搭配
│   ├── llama.py  # 处理请求相应模块, 一般和llama的评测数据集mmlu搭配
├── mmlu-exam, mmlu数据集
├── ceval-exam, ceval数据集
├── eval_test.py  # 启动脚本，建立线程池发送请求，并汇总结果
├── readme.md      # 说明文档
├── requirements.txt   # 第三方依赖
├── service_predict.py  # 发送请求的服务
```

2. 上传精度测试代码到推理容器中。
3. 执行精度测试启动脚本eval_test.py，具体操作命令如下，可以根据参数说明修改参数。

```
python eval_test.py \
  --max_workers=1 \
  --service_name=llama2-13b-chat-test \
  --eval_dataset=ceval \
  --service_url=http://${docker_ip}:8080/v1/completions \
  --few_shot=3 \
  --is_devserver=True \
  --model_name=llama2 \
  --deploy_method=vllm \
  --vllm_model=${model}
```

参数说明:

- max_workers：请求的最大线程数，默认为1。
- service_name：服务名称，保存评测结果时创建目录，示例为：llama2-13b-chat-test。
- eval_dataset：评测使用的评测集（枚举值），目前仅支持mmlu、ceval。
- service_url：成功部署推理服务后的服务预测地址，示例：http://${docker_ip}:8080/generate。此处的${docker_ip}替换为宿主机实际的IP地址，端口号8080来自前面配置的服务端口。
- few_shot：开启少量样本测试后添加示例样本的个数。默认为3，取值范围为0~5整数。

- is_devserver： 是否devserver部署方式，True表示DevServer模式。False表示ModelArts Standard模式。
- model_name：评测模型名称，llama2。
- deploy_method：部署方法，不同的部署方式api参数输入、输出解析方式不同，目前支持tgi、ma_standard、vllm等方式。
- vllm_model：deploy_method为vllm时，服务以openai的方式启动，vllm_model为启动服务时传入的model。

Step3 查看精度测试结果

默认情况下，评测结果会按照result/{service_name}/{eval_dataset}-{timestamp} 的目录结果保存到对应的测试工程。执行多少次，则会在{service_name}下生成多少次结果。

单独的评测结果如下：

```
{eval_dataset}-{timestamp}  # 例如: mmlu-20240205093257
├── accuracy
│   ├── evaluation_accuracy.xlsx  # 测试的评分结果，包含各个学科数据集的评分和总和评分。
├── infer_info
│   ├── xxx1.csv # 单个数据集的评测结果
│   ├── ......
│   ├── xxxn.csv # 单个数据集的评测结果
├── summary_result
│   ├── answer_correct.xlsx  # 回答正确的结果
│   ├── answer_error.xlsx  # 保存回答了问题的选项，但是回答结果错误
│   ├── answer_result_unknow.xlsx  # 保存未推理出结果的问题，例如超时、系统错误
│   ├── system_error.xlsx  # 保存推理结果，但是可能答非所问，无法判断是否正确，需要人工判断进行纠偏。
```

3.16 LLama2 系列模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904）

3.16.1 场景介绍

Llama2（Large Language Model Meta AI）是由Meta AI发布的新一代大语言系列模型，上下文长度由Llama的2048扩展到了4096，可以理解和生成更长的文本。Llama2包含了70亿、130亿和700亿参数的模型，即：Llama2-7B、Llama2-13B、Llama2-70B。

方案概览

本文档利用训练框架Pytorch_npu+华为自研Ascend Snt9b硬件，为用户提供了开箱即用的预训练和全量微调方案。

本文档以Llama2-70B为例，同时适用于Llama2-7B、Llama2-13B。模型运行环境是ModelArts Lite的DevServer。

本方案目前配套的是AscendCloud-3rdLLM系列版本，仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

操作流程

图 3-241 操作流程图

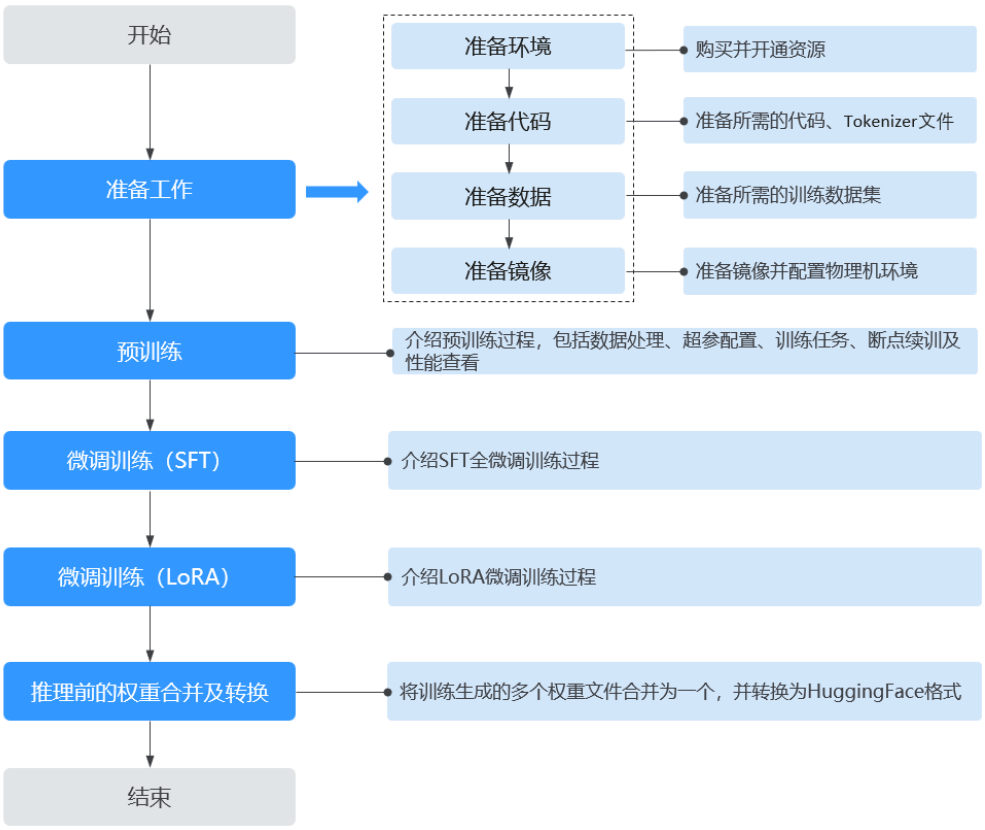


表 3-130 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite DevServer运行的，需要购买并开通DevServer资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用Alpaca数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、断点续训及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调。
	LoRA微调训练	介绍如何进行LoRA微调训练。

阶段	任务	说明
推理前的权重转换	-	模型训练完成后，可以将训练产生的权重文件用于推理。推理前参考本章节，将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。 如果无推理任务或者使用开源Huggingface权重文件进行推理，可以忽略此章节。和本文档配套的推理文档请参考《**开源大模型基于DevServer的推理通用指导**》。

3.16.2 准备工作

3.16.2.1 准备环境

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

资源规格要求

计算规格：对于Llama2-7B和Llama2-13B单机训练需要使用单机8卡，多机训练需要使用2机16卡。对于Llama2-70B至少需要4机32卡才能训练，建议使用8机64卡执行训练相关任务。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.16.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取数据及代码

表 3-131 准备代码

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.904-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**代码目录介绍**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E网站**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
权重和词表文件	包含了本教程使用到的HuggingFace原始权重文件和Tokenizer。 标记器(Tokenizer)是NLP管道的核心组件之一。它们有一个目的：将文本转换为模型可以处理的数据。模型只能处理数字，因此标记器(Tokenizer)需要将文本输入转换为数字数据。	**llama-2-7b-hf** **llama-2-13b-chat-hf** **llama-2-70b-chat-hf** 这个路径下既有权重，也有Tokenizer，全部下载。具体内容参见**权重和词表文件介绍**。

说明

本文档前向兼容AscendCloud-3rdLLM-6.3.T041版本，获取路径：**Support网站**。

代码目录介绍

AscendCloud-3rdLLM代码包结构介绍如下：

```
xxx-Ascend              #xxx表示版本号
  |──llm_evaluation  #推理评测代码包
      |──benchmark_eval   #精度评测
      |──benchmark_tools  #性能评测
  |──llm_train       #模型训练代码包
      |──AscendSpeed  #基于AscendSpeed的训练代码
          |──AscendSpeed  #加速库
          |──ModelLink   #基于ModelLink的训练代码
          |──scripts/    #训练需要的启动脚本
```

本教程需要使用到的训练相关代码存放在llm_train/AscendSpeed目录下，具体文件介绍如下：

```
|──llm_train       #模型训练代码包
      |──AscendSpeed   #基于AscendSpeed的训练代码
          |──AscendSpeed  #加速库
          |──ModelLink   #基于ModelLink的训练代码，数据预处理脚本
          |──scripts/    #训练需要的启动脚本，调用ModelLink
              |──llama2   #llama2的训练代码
                  |──llama2.sh              #llama2训练脚本
```

权重和词表文件介绍

下载完毕后的HuggingFace原始权重文件包含以下内容，此处以Llama2-70B为例，仅供参考，以实际下载的最新文件为准。

```
llama2-70B
 ├── config.json
├── generation_config.json
├── gitattributes.txt
├── LICENSE.txt
├── Notice.txt
├── pytorch_model-00001-of-00015.bin
├── pytorch_model-00002-of-00015.bin
├── pytorch_model-00003-of-00015.bin
├── ...
├── pytorch_model-00015-of-000015.bin
├── pytorch_model.bin.index.json
├── README.md
├── special_tokens_map.json
├── tokenizer_config.json
├── tokenizer.json
├── tokenizer.model
└── USE_POLICY.md
```

工作目录介绍

工作目录结构如下，以下样例以Llama2-70B为例，请根据实际模型命名，Llama2-7B、Llama2-13B或Llama2-70B。

```
${workdir}                      #工作目录，例如/home/ma-user/ws
├──llm_train
   ├── AscendSpeed              #代码目录
      ├── AscendSpeed           #训练依赖的三方模型库
      ├── ModelLink             #AscendSpeed代码目录
      ├── scripts/              #训练启动脚本
# 数据目录结构
├── processed_for_ma_input
    ├── Llama2-70B
       ├── data                 #预处理后数据
          ├── pretrain          #预训练加载的数据
          ├── finetune          #微调加载的数据
       ├──converted_weights     #HuggingFace格式转换magatron格式后权重文件
├── saved_dir_for_ma_output     #训练输出保存权重，根据实际训练需求设置
    ├── Llama2-70B
       ├── logs                 #训练过程中日志（loss、吞吐性能）
       ├── lora                 #lora微调输出权重
       ├── sft                  #增量训练输出权重
       ├── pretrain             #预训练输出权重
├── tokenizers                  #原始权重及tokenizer目录
    ├── Llama2-70B
├── training_data               #原始数据目录
    ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #原始数据文件
```

上传代码到工作环境

1. 使用root用户以SSH的方式登录DevServer。
2. 将AscendSpeed代码包AscendCloud-3rdLLM-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例。
   ```
   unzip AscendCloud-3rdLLM-*.zip  #解压缩
   ```
3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Llama2-{MODEL_TYPE}目录，如Llama2-70B。

   具体步骤如下：

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

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

3.16.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据

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

训练数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。

   ```
   mkdir training_data
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}                    #工作目录，例如/home/ma-user/ws
    |── training_data
        |── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #训练原始数据集
   ```

3.16.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-132 基础镜像地址

镜像用途	镜像地址
基础镜像（训练和推理通用）	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42

说明

本文档兼容cann_7.0.1.1和cann_8.0.rc1的镜像，推荐使用较新版本的cann_8.0.rc1镜像。

表 3-133 模型镜像版本

模型	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**表3-132**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。
   ```
   container_work_dir="/home/ma-user/ws"   # 容器内挂载的目录
   work_dir="/home/ma-user/ws"                 # 宿主机挂载目录，存放了代码、数据、权重
   container_name="ascendspeed"            # 启动的容器名称
   image_name="${container_name} "         # 启动的镜像ID
   ```

```
docker run -itd \
    --device=/dev/davinci0 \
    --device=/dev/davinci1 \
    --device=/dev/davinci2 \
    --device=/dev/davinci3 \
    --device=/dev/davinci4 \
    --device=/dev/davinci5 \
    --device=/dev/davinci6 \
    --device=/dev/davinci7 \
    --device=/dev/davinci_manager \
    --device=/dev/devmm_svm \
    --device=/dev/hisi_hdc \
    -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
    -v /usr/local/dcmi:/usr/local/dcmi \
    -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
    --cpus 192 \
    --memory 1000g \
    --shm-size 200g \
    --net=host \
    -v ${work_dir}:${container_work_dir} \
    --name ${container_name} \
    $image_name \
    /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

2. 通过容器名称进入容器中。

```
docker exec -it ${container_name} bash
```

说明

启动容器时默认用户为ma-user用户。如果需要切换到root用户可以执行以下命令：

```
sudo su
source /home/ma-user/.bashrc
```

如果继续使用ma-user，在使用其他属组如root用户上传的数据和文件时，可能会存在权限不足的问题，因此需要执行如下命令统一文件属主。

```
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
例如：
sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 安装依赖包。

```
#进入scripts目录，xxx为包版本，请按照实际情况替换
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/scripts
#执行安装命令
pip install -r requirements.txt
```

3.16.3 预训练

3.16.3.1 预训练数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

这里以Llama2-70B为例，对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可。

Alpaca 数据处理说明

数据预处理脚本preprocess_data.py存放在代码包的“llm_train/AscendSpeed/ModelLink/tools/”目录中，以Llama2-70B为例， 脚本具体内容如下。

```
#进入到ModelLink目录下，xxx-Ascend请根据实际目录替换
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#数据预处理
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path {work_dir}/tokenizers/Llama2-70B \
 --output-prefix {work_dir}/processed_for_ma_input/Llama2-70B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF
```

参数说明：

- ${work_dir}的路径指容器工作路径：如/home/ma-user/ws/ 。
- - input：原始数据集的存放路径
- - output-prefix：处理后的数据集保存路径+数据集名称前缀（ 例如: alpaca ）
- - tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- - tokenizer-name-or-path：tokenizer的存放路径
- -workers：设置数据处理使用执行卡数量
- -log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出

数据预处理后输出的训练数据如下：

- alpaca_text_document.bin
- alpaca_text_document.idx

训练的时指定的数据路径为${path}/alpaca/llama2-70B/alpaca_text_document， 不加文件类型后缀。

Alpaca 数据处理操作步骤

Alpaca数据处理具体操作步骤如下：

1. 创建数据处理后的输出目录/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/。


```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/Llama2-70B/data/pretrain
```

2. 将获取到的Alpaca预训练数据集传到上一步创建的目录中。如还未下载数据集，请参考**准备数据**获取。
3. 进入“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”目录，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。

```
#加载ascendspeed及megatron模型，xxx-Ascend请根据实际目录替换
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下:
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令:
python ./tools/preprocess_data.py \
 --input /home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path /home/ma-user/ws/tokenizers/Llama2-70B \
 --output-prefix /home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF
```

4. 数据处理完后，在/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/目录下生成alpaca_text_document.bin和alpaca_text_document.idx文件。

   图 3-242 处理后的数据

```
-rw------- 1 root root 26453100 Jan  8 10:26 alpaca_text_document.bin
-rw------- 1 root root  1040082 Jan  8 10:26 alpaca_text_document.idx
```

自定义数据

如果是用户自己准备的数据集，可以使用Ascendspeed代码仓中的转换工具将json格式数据集转换为训练中使用的.idx + .bin格式。

```
#示例:
#1.将准备好的json格式数据集存放于/home/ma-user/ws/training_data目录下: 如data.json
#2.运行转换脚本
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/

#加载ascendspeed及megatron模型，xxx-Ascend请根据实际目录替换
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/data.json \
 --tokenizer-name-or-path {work_dir}/tokenizers/Llama2-70B \
 --output-prefix {work_dir}/processed_for_ma_input/Llama2-70B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF
 #3.执行完成后在 datasets文件夹中可以得到 data_text_document.idx 与data_text_document.bin 两个文件
```

3.16.3.2 预训练任务

配置预训练脚本llama2.sh中的超参，并执行预训练任务。

这里以Llama2-70B 8机64卡训练为例，对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可。

Step1 配置预训练超参

预训练脚本llama2.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/llama2”目录下。训练前，可以根据实际需要修改超参配置。

表 3-134 预训练超参配置

参数	示例值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/alpaca_text_document	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model	必填。加载tokenizer时，tokenizer存放地址。 请根据实际规划修改。
MODEL_TYPE	70B	必填。表示模型加载类型，根据实际填写7B、13B或70B。
TRAIN_ITERS	200	非必填。表示训练迭代周期，根据实际需要修改。
MBS	2	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为2。取值默认值如下： ● Llama2-7B：4 ● Llama2-13B：4 ● Llama2-70B：2
GBS	1024	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。取值默认值： ● Llama2-7B：64 ● Llama2-13B：64 ● Llama2-70B：1024

参数	示例值	参数说明
TP	8	非必填。表示张量并行。默认值为8，取值建议： ● Llama2-7B：8 ● Llama2-13B：8 ● Llama2-70B：8
PP	8	非必填。表示流水线并行。默认值为8，取值建议： ● Llama2-7B：1，一般此值与训练节点数相等。 ● Llama2-13B：1，一般此值与训练节点数相等。 ● Llama2-70B：大于等于4，建议值为8，一般选用几台机器训练则值为几。
RUN_TYPE	pretrain	必填。表示训练类型，根据实际训练任务类型选择。取值说明： ● pretrain：表示预训练 ● retrain：表示断点续训 ● sft：表示SFT微调训练 ● lora：表示LoRA微调训练
MASTER_ADDR	xx.xx.xx.xx	多机必填，单机忽略；指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	8	多机必填，单机忽略；节点总数，单机写1，双机写2，8机写8。
NODE_RANK	0	多机必填，单机忽略；节点序号，当前节点ID，一般从0开始，单机默认是0。以8机训练为例，节点ID依次为（0 1 2 3 4 5 6 7）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。

Step2 启动训练脚本

请根据表3-134修改超参值后，再启动训练脚本。Llama2-70B建议为8机64卡训练。

多机启动

以Llama2-70B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=0 MODEL_TYPE=70B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=1 MODEL_TYPE=70B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
...
...
# 第八台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=7 MODEL_TYPE=70B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH为必填；TRAIN_ITERS、MBS、GBS、 TP、PP、WORK_DIR为非必填，有默认值。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13B为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下，先修改以下命令中的参数，再复制执行。xxx-Ascend请根据实际目录替换。

```
#必填参数
MODEL_TYPE=13B  \
RUN_TYPE=pretrain \
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-13B/data/pretrain/
alpaca_text_document \
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-13B/tokenizer.model \
#非必填参数，有默认值
MBS=4  \
GBS=64 \
TP=8 \
PP=1 \
TRAIN_ITERS=200 \
WORK_DIR=/home/ma-user/ws \
sh scripts/llama2/llama2.sh
```

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-243 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/       50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

如果需要使用断点续训练能力，请参考**断点续训练**章节修改训练脚本。

3.16.3.3 断点续训练

断点续训练是指因为某些原因导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，用于读取前一次训练保存的预训练模型。

断点续训练操作过程

Llama2-70B的断点续训脚本llama2.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/llama2”目录下。

1. 执行命令如下，进入AscendSpeed代码目录。xxx-Ascend请根据实际目录替换。
   ```
   cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/
   ```
2. 修改断点续训练参数。断点续训前，需要在原有训练参数配置**表3-134**中新加“MODEL_PATH”参数，并修改“TRAIN_ITERS”参数和“RUN_TYPE”参数。

   表 3-135 断点续训练修改参数

参数	参考值	参数说明
MODEL_PATH	/home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/pretrain	必填。加载上一步预训练后保存的权重文件。 请根据实际规划修改。
TRAIN_ITERS	300	必填。表示训练周期，必须大于上次保存训练的周期次数。
RUN_TYPE	retrain	必填。训练脚本类型，retrain表示断点续训练。

3. 在AscendSpeed代码目录下执行断点续训练脚本。

   多机启动

   以Llama2-70B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以8机为例。
   ```
   #第一台节点
   MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=0 MODEL_TYPE=70B  RUN_TYPE=retrain
   DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
   alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
   MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/pretrain TRAIN_ITERS=300
   MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
   # 第二台节点
   MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=1 MODEL_TYPE=70B  RUN_TYPE=retrain
   DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
   alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
   MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/pretrain TRAIN_ITERS=300
   MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
   ...
   ```

```
...
# 第八台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=7 MODEL_TYPE=70B  RUN_TYPE=retrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/pretrain TRAIN_ITERS=300
MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
```

以上命令多台机器执行时，只有NODE_RANK的节点ID不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、 TP、PP、WORK_DIR为非必填，有默认值。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13B为例。

进入代码目录/home/ma-user/ws/llm_train/AscendSpeed下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
#非必填参数，有默认值
MBS=4  \
GBS=64 \
TP=8 \
PP=1 \
TRAIN_ITERS=200 \
WORK_DIR=/home/ma-user/ws \
#必填参数
MODEL_TYPE=13B  \
RUN_TYPE=retrain \
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-13B/data/pretrain/
alpaca_text_document \
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-13B/tokenizer.model \
MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Llama2-13B/pretrain \
sh scripts/llama2/llama2.sh
```

图 3-244 保存的 ckpt

```
[root@devserver-modelarts ckpt]# ll
total 24
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 12:34 iter_0000005
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:04 iter_0000010
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:34 iter_0000015
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:04 iter_0000020
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:56 iter_0000025
-rw-r-----  1 HwHiAiUser users    2 Oct 20 14:20 latest_checkpointed_iteration.txt
```

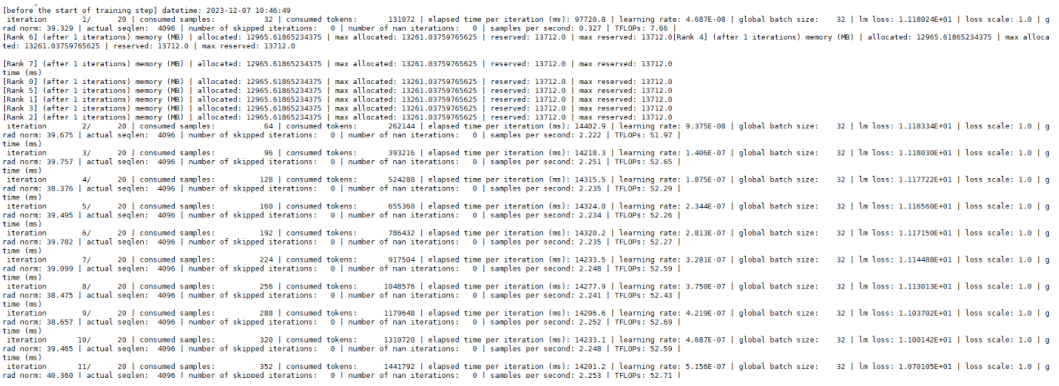
4. 训练完成后，参考**查看日志和性能**操作，查看断点续训练日志和性能。

3.16.3.4 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-245 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为{work_dir}/saved_dir_for_ma_output/Llama2-70B/logs，本实例日志路径为/home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/logs

查看性能

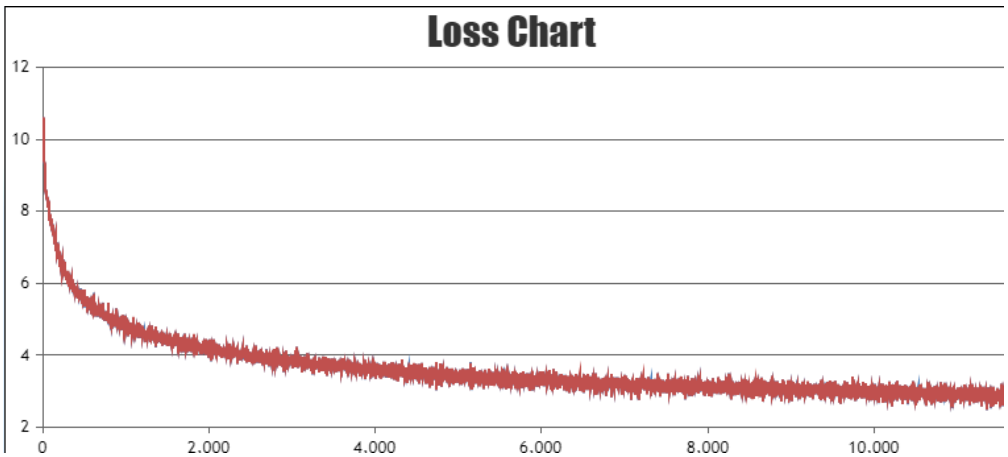
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/（总卡数*elapsed time per iteration）*1000，其参数在日志里可找到，默认seq_len值为4096。Llama2-70B默认global batch size为1024，具体参数查看**表3-134** 中GBS值；其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-246**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-246 Loss 收敛情况（示意图）



3.16.4 SFT 全参微调训练

3.16.4.1 SFT 全参微调数据处理

SFT微调（Supervised Fine-Tuning）前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

这里以LLama2-70B为例，对于LLama2-7B和LLama2-13B，操作过程与LLama2-70B相同，只需修改对应参数即可。

下载数据

SFT全参微调涉及的数据下载地址：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**

如果在**准备数据**章节已下载数据集，此处无需重复操作。

SFT全参微调和LoRA微调训练使用的是同一个数据集，数据处理一次即可，训练时可以共用。

数据预处理说明

使用数据预处理脚本preprocess_data.py脚本重新生成.bin和.idx格式的SFT全参微调数据。preprocess_data.py存放在xxx-Ascend/llm_train/AscendSpeed/ModelLink/tools目录中，脚本具体内容如下。xxx-Ascend请根据实际目录替换。

```
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入ModelLink目录
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
    --input /home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
    --tokenizer-name-or-path $TOKENIZER_PATH \
    --output-prefix $DATASET_PATH\
    --tokenizer-type PretrainedFromHF \
    --workers 8 \
    --handler-name GeneralInstructionHandler \
    --log-interval 1000 \
    --append-eod
```

参数说明：

- input：SFT全参微调数据的存放路径。

- output-prefix：处理后的数据集保存路径+数据集名称前缀（例如：alpaca_ft）。

- tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，设置为PretrainedFromHF。

- tokenizer-name-or-path：tokenizer的存放路径。

- handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。

- workers：数据处理线程数。

-append-eod：用于控制是否在每个输入序列的末尾添加一个特殊的标记。这个标记表示输入序列结束，可以帮助模型更好地理解和处理长序列。

- log-interval：输出处理日志刷新间隔。

输出结果

alpaca_ft_packed_attention_mask_document.bin

alpaca_ft_packed_attention_mask_document.idx

alpaca_ft_packed_input_ids_document.bin

alpaca_ft_packed_input_ids_document.idx

alpaca_ft_packed_labels_document.bin

alpaca_ft_packed_labels_document.idx

数据处理具体操作

SFT全参微调数据处理具体操作步骤如下。

1. 创建处理后的数据存放目录/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/
```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/Llama2-70B/data/finetune
```
2. 进入代码目录“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。
```
#进入到ModelLink目录下，xxx-Ascend请根据实际目录替换。
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#执行以下命令
python ./tools/preprocess_data.py \
  --input /home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
  --tokenizer-name-or-path /home/ma-user/ws/tokenizers/Llama2-70B \
  --output-prefix /home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft \
  --workers 8 \
  --log-interval 1000 \
  --tokenizer-type PretrainedFromHF \
  --handler-name GeneralInstructionHandler \
  --append-eod
```
   数据处理完后，在/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/目录下生成转换后的数据文件。

3.16.4.2 SFT 全参微调权重转换

SFT全参微调需将HuggingFace格式权重转换为megatron格式后再进行SFT全参微调。

本章节主要介绍如何将HuggingFace权重转换为Megatron格式。此处的HuggingFace权重文件和转换操作结果同时适用于SFT全参微调和LoRA微调训练

HuggingFace 权重转换操作

1. 下载Llama2-70B的预训练权重和词表文件，并上传到/home/ma-user/ws/tokenizers/Llama2-70B目录下。具体下载地址请参见**表3-131**。如果已下载，忽略此步骤。
2. 创建权重转换后的输出目录/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights/。
```
cd /home/ma-user/ws/  #进入/home/ma-user/ws/目录
mkdir -p processed_for_ma_input/Llama2-70B/converted_weights
```

3. 进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink，在代码目录中执行util.py脚本。xxx-Ascend请根据实际目录替换。

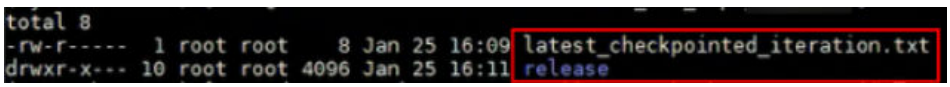
```
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
# 权重格式转换
python tools/checkpoint/util.py --model-type GPT \
    --loader llama2_hf \
    --saver megatron \
    --target-tensor-parallel-size 8 \
    --target-pipeline-parallel-size 8 \
    --load-dir /home/ma-user/ws/tokenizers/Llama2-70B \
    --save-dir /home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights \
    --tokenizer-model /home/ma-user/ws/tokenizers/Llama2-70B/tokenizer.model
```

参数说明如下：

- --model-type：模型类型。
- --loader：权重转换要加载检查点的模型名称。
- --tensor-model-parallel-size：张量并行数，需要与训练脚本中的TP值配置一样。
- --pipeline-model-parallel-size：流水线并行数，需要与训练脚本中的PP值配置一样。
- --saver：检查模型保存名称。
- --load-dir：加载转换模型权重路径。
- --save-dir : 权重转换完成之后保存路径。
- --tokenizer-model : tokenizer路径。

4. 权重转换完成后，在/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights目录下查看转换后的权重文件。

图 3-247 转换后的权重文件



3.16.4.3 SFT 全参微调任务

前提条件

- SFT全参微调使用的数据集为alpaca_data数据，已经完成数据处理，具体参见**SFT全参微调数据处理**。
- 已经将开源HuggingFace权重转换为Megatron格式，具体参见**SFT全参微调权重转换**。

Step1 修改训练超参配置

SFT全参微调脚本llama2.sh，存放在xxx-Ascend/llm_train/AscendSpeed/scripts/llama2目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。SFT微调的计算量与预训练基本一致，故配置可以与预训练相同。

表 3-136 SFT 全参微调超参配置

参数	值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-70B	必填。加载tokenizer时，tokenizer存放地址。请根据实际规划修改。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights	必填。加载的权重文件路径。**SFT全参微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
MODEL_TYPE	70B	必填。模型加载类型，根据实际填写7B、13B或70B。
TRAIN_ITERS	200	非必填。训练迭代周期。根据实际需要修改。
MBS	2	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为2。取值建议如下： • Llama2-7B：4 • Llama2-13B：4 • Llama2-70B：2
GBS	1024	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。取值默认值： • Llama2-7B：64 • Llama2-13B：64 • Llama2-70B：1024
TP	8	非必填。表示张量并行。默认值为8，取值建议： • Llama2-7B：8 • Llama2-13B：8 • Llama2-70B：8

参数	值	参数说明
PP	8	非必填。表示流水线并行。取值建议： • Llama2-7B：1，一般此值与运行节点数相等。 • Llama2-13B：1，一般此值与运行节点个数相等。 • Llama2-70B：大于等于4，建议值为8，一般选用几台机器训练则值为几。
RUN_TYPE	sft	必填。表示训练类型，sft表示SFT微调训练。
MASTER_ADDR	xx.xx.xx.xx	多机必填，单机忽略。指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	8	多机必填，单机忽略。节点总数，单机写1，双机写2，8机写8。
NODE_RANK	0	多机必填，单机忽略。节点序号，当前节点ID，一般从0开始，单机默认是0。以8机训练为例，节点ID依次为（0 1 2 3 4 5 6 7）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。

Step2 启动训练脚本

请根据表3-136修改超参值后，再启动训练脚本。Llama2-70B建议为8机64卡训练。

多机启动

以Llama2-70B为例，多台机器执行训练启动命令如下。进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=0 MODEL_TYPE=70B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=1 MODEL_TYPE=70B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
...
...
# 第八台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=7 MODEL_TYPE=70B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/
```

```
Llama2-70B/converted_weights  TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8 WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填。TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13B为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本，先修改以下命令中的参数，再复制执行。

```
#非必填参数，有默认值
MBS=4  \
GBS=64 \
TP=8 \
PP=1 \
TRAIN_ITERS=200 \
WORK_DIR=/home/ma-user/ws \
#必填参数
MODEL_TYPE=13B  \
RUN_TYPE=sft \
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-13B/data/finetune/alpaca_ft \
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-13B \
MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights \
sh scripts/llama2/llama2.sh
```

训练完成后，请参考**查看日志和性能**章节查看SFT微调的日志和性能。

3.16.5 LoRA 微调训练

本章节以Llama2-70B为例，介绍LoRA微调训练的全过程。对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可。

Step1 LoRA 微调数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

LoRA微调训练与SFT微调使用同一个数据集，如果已经在SFT微调时处理过数据，可以直接使用，无需重复处理。如果未处理过数据，请参见**SFT全参微调数据处理**章节先处理数据。

Step2 LoRA 微调权重转换

LoRA微调训练前，需要先把训练权重文件转换为Megatron格式。

LoRA微调训练和SFT全参微调使用的是同一个HuggingFace权重文件，转换为Megatron格式后的结果也是通用的。

如果在SFT微调任务中已经完成了HuggingFace权重转换操作，此处无需重复操作，可以直接使用SFT微调中的权重转换结果。

如果前面没有执行HuggingFace权重转换任务，可以参考**SFT全参微调权重转换**章节完成。

Step3 LoRA 微调超参配置

LoRA微调训练脚本llama2.sh，存放在xxx-Ascend/llm_train/AscendSpeed/scripts/llama2/目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。

表 3-137 LoRA 微调超参配置

参数	值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Llama2-70B	必填。加载tokenizer时，tokenizer存放地址。请根据实际规划修改。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/Llama2-70B/converted_weights	必填。加载的权重文件路径。**Step2 LoRA微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
MODEL_TYPE	70B	必填。模型加载类型，根据实际填写7B、13B或70B。
TRAIN_ITERS	200	非必填。训练迭代周期。根据实际需要修改。
MBS	2	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为2。取值建议如下： • Llama2-7B：4 • Llama2-13B：4 • Llama2-70B：2
GBS	1024	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。取值默认值： • Llama2-7B：64 • Llama2-13B：64 • Llama2-70B：1024

参数	值	参数说明
TP	8	非必填。表示张量并行。默认值为8，取值建议： ● Llama2-7B：8 ● Llama2-13B：8 ● Llama2-70B：8
PP	8	非必填。表示流水线并行。取值建议： ● Llama2-7B：1，一般此值与运行节点数相等 ● Llama2-13B：1，一般此值与运行节点数相等 ● Llama2-70B：大于等于4，建议值为8，一般选用几台机器训练则值为几。
RUN_TYPE	lora	必填。表示训练类型，lora表示LoRA微调训练。
MASTER_ADDR	xx.xx.xx.xx	多机必填，单机忽略；指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	8	多机必填，单机忽略；节点总数，单机写1，双机写2，8机写8。
NODE_RANK	0	多机必填，单机忽略；节点序号，当前节点ID，一般从0开始，单机默认是0。以8机训练为例，节点ID依次为（0 1 2 3 4 5 6 7）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。

Step4 启动训练脚本

请根据表3-137修改超参值后，再启动训练脚本。Llama2-70B建议为8机64卡训练。

多机启动

以Llama2-70B为例，多台机器执行训练启动命令如下。进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=0 MODEL_TYPE=70B  RUN_TYPE=lora
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/
processed_for_ma_input/Llama2-70B/converted_weights TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8
```

```
WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=1 MODEL_TYPE=70B  RUN_TYPE=lora
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/
processed_for_ma_input/Llama2-70B/converted_weights TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8
WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
...
...
# 第八台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=8 NODE_RANK=7 MODEL_TYPE=70B  RUN_TYPE=lora
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-70B/data/finetune/alpaca_ft
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-70B MODEL_PATH=/home/ma-user/ws/
processed_for_ma_input/Llama2-70B/converted_weights TRAIN_ITERS=200 MBS=2  GBS=1024 TP=8 PP=8
WORK_DIR=/home/ma-user/ws sh scripts/llama2/llama2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填项。

TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

单机启动

对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可，可以选用单机启动，以Llama2-13B为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。先修改以下命令中的参数，再复制执行

```
#非必填参数，有默认值，如需修改请根据实际要求填入以下参数。
MBS=4  \
GBS=64 \
TP=8 \
PP=1 \
TRAIN_ITERS=200 \
WORK_DIR=/home/ma-user/ws \
#必填参数
MODEL_TYPE=13B  \
RUN_TYPE=lora \
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-13B/data/finetune/alpaca_ft \
TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Llama2-13B \
MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Llama2-13B/converted_weights \
sh scripts/llama2/llama2.sh
```

训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

3.16.6 推理前的权重合并转换

模型训练完成后，训练的产物包括模型的权重、优化器状态、loss等信息。这些内容可用于断点续训、模型评测或推理任务等。

在进行模型评测或推理任务前，需要将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。

权重文件的合并转换操作都要求在训练的环境中进行，为下一步推理做准备。

- 如果需要使用本文档中训练后的权重文件进行推理，请参考此章节合并训练权重文件并转换为Huggingface格式。
- 如果无推理任务或者使用开源Huggingface权重文件推理，都可以忽略此章节。

下一步的推理任务请参考文档《**开源大模型基于DevServer的推理通用指导**》。

将多个权重文件合并为一个文件并转换格式

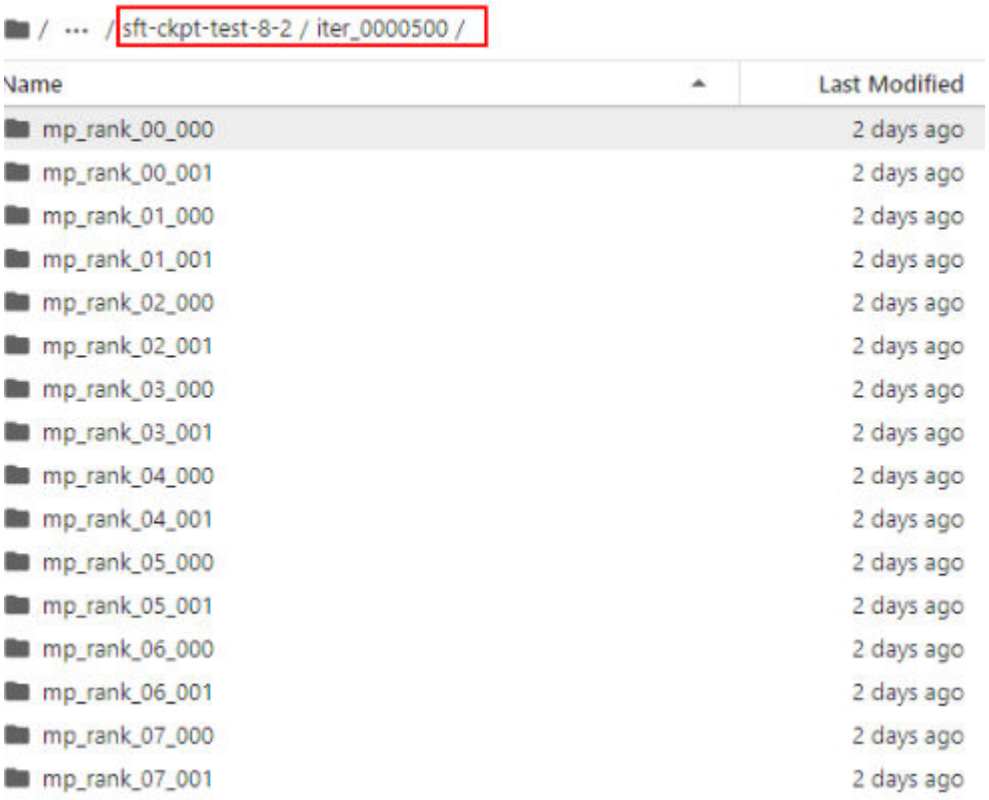
该场景一般用于将预训练、SFT或LoRA训练好的Megatron模型重新转回HuggingFace格式。

本章节以Llama2-70B为例，对于Llama2-7B和Llama2-13B，操作过程与Llama2-70B相同，只需修改对应参数即可。

一般训练都是多卡分布式训练，权重结果文件为多个且文件为Megatron格式，因此需要合并多个文件并转换为HuggingFace格式。

如果是多机训练，转换前需将多机权重目录（iter_xxxxxxx）下的mp_rank_xx_xxx文件夹整合到一起后再进行转换，合并后结果如下图所示。

图 3-248 合并权重文件



该脚本的执行需要在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink目录下执行。

```
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入ModelLink下
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python tools/checkpoint/util.py --model-type GPT \
    --loader megatron \
    --saver megatron \
    --save-model-type save_huggingface_llama \
    --megatron-path /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink \
    --load-dir /home/ma-user/ws/saved_dir_for_ma_output/Llama2-70B/lora \
    --target-tensor-parallel-size 1 \
    --target-pipeline-parallel-size 1 \
    --save-dir /home/ma-user/ws/tokenizers/Llama2-70B/  # <-- 需要填入原始HF模型路径，新权重会存于../Llama2-70B/mg2hg下
```

参数说明：

- save-model-type：输出后权重格式。
- load-dir：训练完成后保存的权重路径。
- save-dir：需要填入原始HF模型路径，新权重会存于../Llama2-70B/mg2hg下。
- target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size，默认为1。
- target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size，默认为1。

3.17 Qwen 系列模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904）

3.17.1 场景介绍

Qwen大模型是一个包含多种参数数量模型的语言模型。

本文档以Qwen-7B/14B/72B为例，利用训练框架Pytorch_npu+华为自研Ascend Snt9b硬件，为用户提供了开箱即用的预训练和微调训练方案。

操作流程

图 3-249 操作流程图

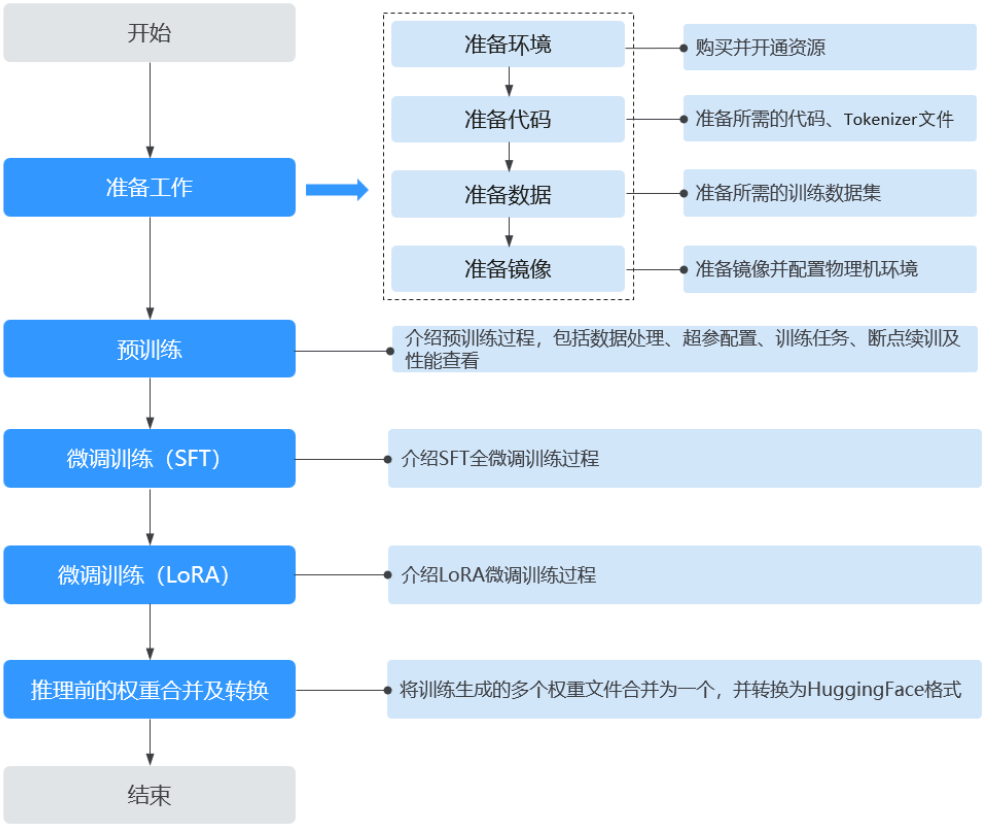


表 3-138 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	购买并开通模型运行所需的资源环境。
	准备代码	准备AscendSpeed代码、分词器Tokenizer和推理代码。
	准备数据	准备数据，可以用Alpaca数据集，也可以使用自己准备的数据集。
	准备镜像	准备模型适用的容器镜像，包括容器内资源检查
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、断点续训及性能查看。
微调训练	SFT微调训练	介绍如何进行SFT微调训练。
	LoRA微调训练	介绍如何进行LoRA微调训练。
推理前的权重转换	-	模型训练完成后，可以将训练产生的权重文件用于推理。推理前参考本章节，将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。 如果无推理任务或者使用开源Huggingface权重文件进行推理，可以忽略此章节。和本文档配套的推理文档请参考《**开源大模型基于DevServer的推理通用指导**》。

微调训练和预训练的区别

微调训练是在预训练权重的基础上使用指令数据集进行的，对模型权重进行学习调整。从而针对特定任务达到预期效果。

微调训练与预训练任务的区别主要包括：

1. 使用的数据不同，微调使用的是指令数据集，在处理数据集时需要将--handler-name 参数指定为GeneralInstructionHandler。
2. 因数据集不同，loss计算方式不同。需要在微调训练时指定--finetune 和--is-instruction-dataset参数，微调任务的脚本里面已经自动识别添加。
3. 启动脚本的环境变量RUN_TYPE需要指定为sft或者lora。

3.17.2 准备工作

3.17.2.1 准备环境

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

资源规格要求

计算规格：对于Qwen-7B和Qwen-14B单机训练需要使用单机8卡，多机训练需要使用2机16卡。对于Qwen-72B至少需要5机40卡才能训练，建议使用8机64卡执行训练相关任务。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通资源

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

说明

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

3.17.2.2 准备代码

本教程中用到的代码和权重文件如下表所示，请提前准备，并按要求在容器中创建工作目录。

获取代码和权重文件

表 3-139 准备代码

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.904-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**代码目录介绍**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E网站**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
权重和词表文件	包含了本教程使用到的HuggingFace原始权重文件和Tokenizer。 标记器(Tokenizer)是NLP管道的核心组件之一。它们有一个目的：将文本转换为模型可以处理的数据。模型只能处理数字，因此标记器(Tokenizer)需要将文本输入转换为数字数据。	**Qwen-14B-Chat** **Qwen-7B-Chat** **Qwen-72B-Chat** 这个路径下既有权重，也有Tokenizer，全部下载。具体内容参见**权重和词表文件介绍**。

说明

本文档前向兼容AscendCloud-3rdLLM-6.3.T041版本，获取路径：**Support网站**。

代码目录介绍

AscendCloud-3rdLLM代码包结构介绍如下：

```
xxx-Ascend                #xxx表示版本号
    ├──llm_evaluation   #推理评测代码包
        ├──benchmark_eval   #精度评测
        ├──benchmark_tools  #性能评测
    ├──llm_train        #模型训练代码包
        ├──AscendSpeed   #基于AscendSpeed的训练代码
            ├──AscendSpeed   #加速库
            ├──ModelLink   #基于ModelLink的训练代码
            ├──scripts     #训练需要的启动脚本
```

本教程需要使用到的训练相关代码存放在llm_train/AscendSpeed目录下，具体文件介绍如下：

```
├──llm_train        #模型训练代码包
        ├──AscendSpeed   #基于AscendSpeed的训练代码
            ├──AscendSpeed   #加速库
            ├──ModelLink   #基于ModelLink的训练代码和数据预处理脚本
            ├──scripts    #训练需要的启动脚本，调用ModelLink
                ├──qwen   #qwen的训练代码
                    ├──qwen.sh                #qwen训练脚本
```

权重和词表文件介绍

下载完毕后的HuggingFace原始权重文件包含以下内容，此处以Qwen-14B为例，仅供参考，以实际下载的最新文件为准。

```
qwen-14b
├── assets
├── cache_autogptq_cuda_256.cpp
├── cache_autogptq_cuda_kernel_256.cu
├── config.json
├── configuration_qwen.py
├── cpp_kernels.py
├── examples
├── generation_config.json
├── LICENSE
├── model-00001-of-00015.safetensors
├── model-00002-of-00015.safetensors
|   ...
├── model-00014-of-00015.safetensors
├── model-00015-of-00015.safetensors
├── modeling_qwen.py
├── model.safetensors.index.json
├── NOTICE
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
├── tokenization_qwen.py
├── tokenizer_config.json
```

工作目录介绍

工作目录结构如下，以下样例都以Qwen-14B为例，请根据实际模型命名，Qwen-7B、Qwen-14B或Qwen-72B。

```
${workdir}（例如/home/ma-user/ws ）
├──llm_train
    ├── AscendSpeed             #代码目录
```

```
    ├── AscendSpeed          #训练依赖的三方模型库
    ├── ModelLink            #AscendSpeed代码目录
    ├── scripts              #训练启动脚本
├── processed_for_ma_input
    ├── Qwen-14B             #请根据实际模型命名，Qwen-7B、Qwen-14B或Qwen-72B
        ├── data             #预处理后数据
            ├── pretrain     #预训练加载的数据
            ├── finetune     #微调加载的数据
        ├──converted_weights  #HuggingFace格式转换magatron格式后权重文件
├── saved_dir_for_ma_output          #训练输出保存权重，根据实际训练需求设置
    ├── Qwen-14B             #请根据实际模型命名，Qwen-7B、Qwen-14B或Qwen-72B
        ├── logs             #训练过程中日志（loss、吞吐性能）
        ├── lora             #lora微调输出权重
        ├── sft              #增量训练输出权重
        ├── pretrain         #预训练输出权重
├── tokenizers                   #原始权重及tokenizer目录
    ├── Qwen-14B             #请根据实际模型命名，Qwen-7B、Qwen-14B或Qwen-72B
├── training_data                #原始数据目录
    ├── pretrain             #预训练加载的数据
        ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预训练原始数据文件
    ├── finetune             #微调训练加载的数据
        ├── alpaca_gpt4_data.json                    #微调训练原始数据文件
```

上传代码到工作环境

1. 使用root用户以SSH的方式登录DevServer。将AscendSpeed代码包AscendCloud-3rdLLM-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例。

   ```
   unzip AscendCloud-3rdLLM-xxx-xxx.zip  #解压缩，-xxx-xxx表示软件包版本号和时间戳
   ```

2. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/Qwen-{MODEL_TYPE}目录，如Qwen-14B。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws，创建tokenizers文件目录将权重和词表文件放置此处，以Qwen-14B为例。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/Qwen-14B
   ```

3. 修改tokenizers/Qwen-{MODEL_TYPE}中的modelling_qwen.py脚本文件。如Qwen-14B：大约39行处SUPPORT_FP16的值。

   修改前：

   ```
   SUPPORT_FP16 = SUPPORT_CUDA and torch.cuda.get_device_capability(0)[0] >= 7
   ```

   修改后：

   ```
   SUPPORT_FP16 = True
   ```

3.17.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据

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

- 训练数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

- SFT全参微调、LoRA微调训练数据集下载： https://github.com/Instruction-Tuning-with-GPT-4/GPT-4-LLM/blob/main/data/alpaca_gpt4_data.json，数据大小：42M左右。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

上传数据到指定目录

将下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data/pretrain”，并将预训练原始数据放置在此处。

   ```
   mkdir -p training_data/pretrain
   ```

   创建目录“training_data/finetune”，并将微调训练原始数据放置在此处

   ```
   mkdir -p training_data/finetune
   ```

   数据存放参考目录结构如下：

   ```
   ${workdir}（例如/home/ma-user/ws）
   ├── training_data                    #原始数据目录
       ├── pretrain                     #预训练加载的数据
           ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预训练原始数据文件
       ├── finetune                     #微调训练加载的数据
           ├── alpaca_gpt4_data.json                          #微调训练原始数据文件
   ```

3.17.2.4 准备镜像

准备训练模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置物理机环境操作。

镜像地址

本教程中用到的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-140 基础镜像地址

镜像用途	镜像地址
基础镜像（训练和推理通用）	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42

说明

本文档兼容cann_7.0.1.1和cann_8.0.rc1的镜像，推荐使用较新版本的cann_8.0.rc1镜像。

表 3-141 模型镜像版本

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0
PyTorch_npu	2.1.0.post3-20240413

Step1 检查系统环境

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

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

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

2. 检查是否安装docker。

```
docker -v   #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表3-140**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
container_work_dir="/home/ma-user/ws"   # 容器内挂载的目录
work_dir="/home/ma-user/ws"             # 宿主机挂载目录，存放了代码、数据、权重
container_name="ascendspeed"            # 启动的容器名称
image_name="${container_name} "         # 启动的镜像ID
docker run -itd \
    --device=/dev/davinci0 \
    --device=/dev/davinci1 \
    --device=/dev/davinci2 \
    --device=/dev/davinci3 \
    --device=/dev/davinci4 \
    --device=/dev/davinci5 \
    --device=/dev/davinci6 \
    --device=/dev/davinci7 \
    --device=/dev/davinci_manager \
```

```
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
-v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
--cpus 192 \
--memory 1000g \
--shm-size 200g \
--net=host \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
$image_name \
/bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

2. 通过容器名称进入容器中。

```
docker exec -it ${container_name} bash
```

说明

启动容器时默认用户为ma-user用户。如果需要切换到root用户可以执行以下命令：

```
sudo su
source /home/ma-user/.bashrc
```

如果继续使用ma-user，在使用其他属组如root用户上传的数据和文件时，可能会存在权限不足的问题，因此需要执行如下命令统一文件属主。

```
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
例如：
sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 安装pip源。

```
#进入scriptsscripts目录
cd /home/ma-user/ws/xxxend/llm_train/AscendSpeed/scripts
#执行安装命令
pip install -r requirements.txt
```

3.17.3 预训练

3.17.3.1 预训练数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

这里以Qwen-14B为例，对于Qwen-7B和Qwen-72B，操作过程与Qwen-14B相同，只需修改对应参数即可。

Alpaca 数据处理说明

数据预处理脚本preprocess_data.py存放在代码包的“llm_train/AscendSpeed/ModelLink/tools/”目录中，脚本具体内容如下。

```
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#数据预处理
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path {work_dir}/tokenizers/Qwen-14B \
 --output-prefix {work_dir}/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF \
 --seq-length 4096
```

参数说明：

- ${work_dir}的路径指容器工作路径：如/home/ma-user/ws/ 。
- - input：原始数据集的存放路径。
- - output-prefix：处理后的数据集保存路径+数据集名称前缀（ 例如: alpaca ），替换为实际模型的路径。
- - tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- - tokenizer-name-or-path：tokenizer的存放路径，替换为实际模型的路径。
- -workers：设置数据处理使用执行卡数量。
- -log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出。
- -seq-length : 是一个用于设置序列长度的参数，表示模型处理的序列长度。在训练大规模模型时，可以通过设置这个参数来优化模型的训练速度和效果。

数据预处理后输出的训练数据如下：

- alpaca_text_document.bin
- alpaca_text_document.idx

训练的时指定的数据路径为${path}/alpaca/qwen-14b/alpaca_text_document， 不加文件类型后缀。

Alpaca 数据处理操作步骤

Alpaca数据处理具体操作步骤如下：

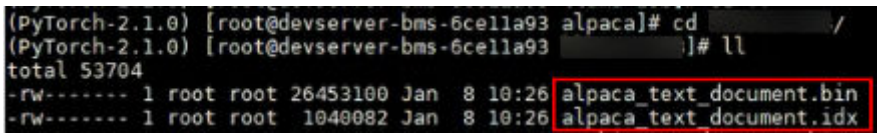
1. 创建数据处理后的输出目录/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/。
   ```
   cd /home/ma-user/ws/  #进入容器工作目录
   mkdir -p processed_for_ma_input/Qwen-14B/data/pretrain
   ```
2. 将获取到的Alpaca预训练数据集传到上一步创建的目录中。如还未下载数据集，请参考**准备数据**获取。
3. 进入“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”目录，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。


```
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/
AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/
ModelLink
#进入到ModelLink目录下
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令
python ./tools/preprocess_data.py \
 --input /home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path /home/ma-user/ws/tokenizers/Qwen-14B \
 --output-prefix /home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF \
 --seq-length 4096
```

4. 数据处理完后，在/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/目录下生成alpaca_text_document.bin和alpaca_text_document.idx文件。

图 3-250 处理后的数据



自定义数据

如果是用户自己准备的数据集，可以使用Ascendspeed代码仓中的转换工具将json格式数据集转换为训练中使用的.idx + .bin格式。

```
#示例
#1.将准备好的json格式数据集存放于/home/ma-user/ws/training_data/pretrain目录下: 如data.json
#2.运行转换脚本
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#运行以下命令
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/pretrain/data.json \
 --tokenizer-name-or-path {work_dir}/tokenizers/Qwen-14B \
 --output-prefix {work_dir}/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF \
 --seq-length 4096
#3.执行完成后在 datasets文件夹中可以得到 data_text_document.idx 与data_text_document.bin 两个文件
```

3.17.3.2 预训练任务

配置预训练脚本qwen.sh中的超参，并执行预训练任务。

这里以Qwen-14B为例，对于Qwen-7B和Qwen-72B，操作过程与Qwen-14B相同，只需修改对应参数即可。

预训练超参配置

预训练脚本qwen.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/qwen”目录下。训练前，需要根据实际需要配置超参。

表 3-142 预训练超参配置

参数	示例值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Qwen-14B	必填。加载tokenizer时，tokenizer存放地址。 请根据实际规划修改。
MODEL_TYPE	14B	必填。表示模型加载类型，根据实际填写7B、14B或72B。
TRAIN_ITERS	200	非必填。表示训练迭代周期，根据实际需要修改。
MBS	2	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为2。取值建议如下： ● Qwen-14B：2 ● Qwen-7B：2 ● Qwen-72B：1
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。 默认值为64。对于PP（流水线并行）值大于1的场景，增大GBS值吞吐性能会有提升。
TP	8	非必填。表示张量并行。默认值为8，取值建议： ● Qwen-14B：8 ● Qwen-7B：4 ● Qwen-72B：8
PP	1	非必填。表示流水线并行。默认值为1，取值建议： ● Qwen-14B：1 ● Qwen-7B：1 ● Qwen-72B：大于等于5，例如5机填写5，8机填8。

参数	示例值	参数说明
RUN_TYPE	pretrain	必填。表示训练类型，根据实际训练任务类型选择。取值说明： ● pretrain：表示预训练 ● retrain：表示断点续训 ● sft：表示SFT微调训练 ● lora：表示LoRA微调训练
MASTER_ADDR	localhost	多机必填。主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填。节点总数，如为双机，则写2。单机默认是1。
NODE_RANK	0	多机必填。节点序号，当前节点ID，一般从0开始，单机默认是0。以Qwen-72B 5机训练为例，节点ID依次为（0 1 2 3 4）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	容器的工作目录。训练的权重文件保存在此路径下。非必填，默认值为：/home/ma-user/ws。
SEQ_LEN	4096	非必填。默认值为4096。

启动训练脚本

请根据表3-142修改超参值后，再启动训练脚本。

单机启动

以Qwen-14B为例，单机训练启动样例命令如下。在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下。

```
MODEL_TYPE=14B  RUN_TYPE=pretrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B TRAIN_ITERS=200 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH为必填，TRAIN_ITERS、MBS、GBS、TP、PP、SEQ_LEN为非必填，有默认值。

多机启动

以Qwen-14B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下执行。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=14B  RUN_TYPE=pretrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B TRAIN_ITERS=200 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
...
```

```
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=14B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B TRAIN_ITERS=200
MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH为必填，TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-251 等待模型载入

```
> finished creating llama datasets ...
time (ms) | model-and-optimizer-setup: 5888.90 | train/valid/test-data-iterators-setup: 837.51
[after dataloaders are built] datetime: 2024-01-25 14:56:51
done with setup ...
training ...
[before the start of training step] datetime: 2024-01-25 14:56:52
 iteration        1/      50 | consumed samples:           32 | consumed tokens:       131072 |
:    32 | lm loss: 1.045802E+01 | loss scale: 1.0 | grad norm: 105.789 | actual seqlen:  4096 |
 0.551 | TFLOPs: 24.34 |
```

更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

如果需要使用断点续训练能力，请参考**断点续训练**章节修改训练脚本。

3.17.3.3 断点续训练

断点续训练是指因为某些原因导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，用于读取前一次训练保存的预训练模型。

训练过程

断点续训脚本qwen.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/qwen”目录下。

1. 执行命令如下，进入AscendSpeed代码目录。
   ```
   cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/
   ```
2. 修改断点续训练参数。断点续训前，需要在原有训练参数配置**表3-142**中新加“MODEL_PATH”参数，并修改“TRAIN_ITERS”参数和“RUN_TYPE”参数。

表 3-143 断点续训练修改参数

参数	示例值	参数说明
MODEL_PATH	/home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/pretrain	必填。加载上一步预训练后保存的权重文件。 请根据实际规划修改。
TRAIN_ITERS	300	必填。表示训练周期，必须大于上次保存训练的周期次数。
RUN_TYPE	retrain	必填。训练脚本类型，retrain表示断点续训练。

3. 在AscendSpeed代码目录下执行断点续训练脚本。

单机启动

```
MODEL_TYPE=14B  RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/pretrain TRAIN_ITERS=300 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

多机启动

```
以Qwen-14B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=14B  RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/pretrain TRAIN_ITERS=300 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=14B  RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/pretrain TRAIN_ITERS=300 MBS=2  GBS=64 TP=8 PP=12 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

图 3-252 保存的 ckpt

```
[root@devserver-modelarts ckpt]# ll
total 24
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 12:34 iter_0000005
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:04 iter_0000010
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:34 iter_0000015
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:04 iter_0000020
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:56 iter_0000025
-rw-r-----  1 HwHiAiUser users    2 Oct 20 14:20 latest_checkpointed_iteration.txt
```

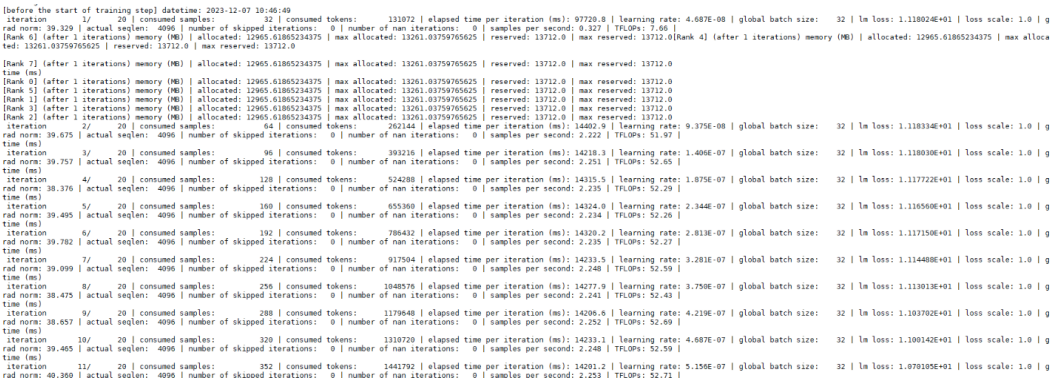
4. 训练完成后，参考**查看日志和性能**，查看断点续训练日志和性能。

3.17.3.4 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-253 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。

本示例日志路径为/home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/logs

查看性能

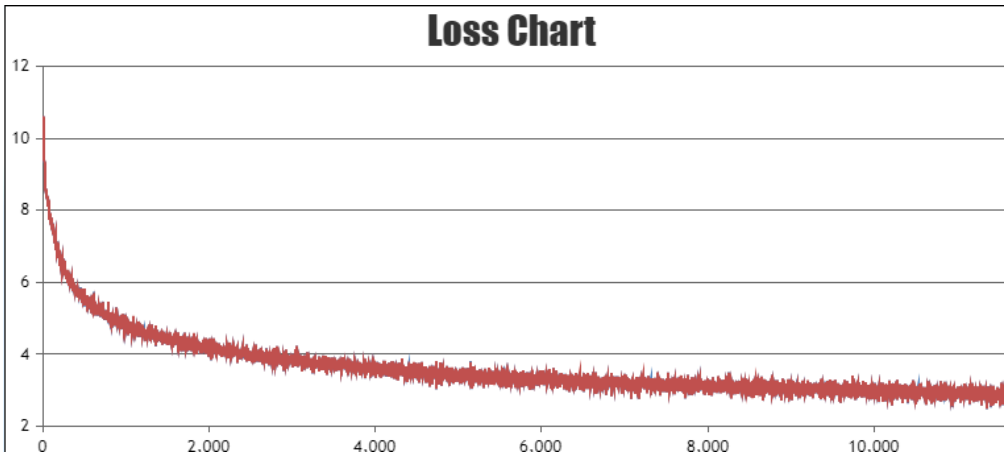
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/（总卡数*elapsed time per iteration）*1000，其参数在日志里可找到，默认seq_len值为4096，默认global batch size为64；其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具TrainingLogParser查看loss收敛情况，如图3-254所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-254 Loss 收敛情况（示意图）



3.17.4 SFT 微调训练

3.17.4.1 SFT 微调数据处理

SFT微调（Supervised Fine-Tuning）前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

这里以Qwen-14B为例，对于Qwen-7B和Qwen-72B，操作过程与Qwen-14B相同，只需修改对应参数即可。

下载数据

SFT微调涉及的数据下载地址：https://github.com/Instruction-Tuning-with-GPT-4/GPT-4-LLM/blob/main/data/alpaca_gpt4_data.json

如果在**准备数据**章节已下载数据集，此处无需重复操作。

SFT微调和LoRA微调训练使用的是同一个数据集，数据处理一次即可，训练时可以共用。

数据预处理说明

使用数据预处理脚本preprocess_data.py脚本重新生成.bin和.idx格式的SFT全参微调数据。preprocess_data.py存放在llm_train/AscendSpeed/ModelLink/tools目录中，脚本具体内容如下。

```
#进入ModelLink目录
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#加载ascendspeed及megatron模型
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#执行以下命令
python ./tools/preprocess_data.py \
    --input /home/ma-user/ws/training_data/finetune/alpaca_gpt4_data.json \
    --tokenizer-name-or-path $TOKENIZER_PATH \
    --output-prefix $DATASET_PATH\
    --tokenizer-type PretrainedFromHF \
    --seq-length 4096 \
    --workers 8 \
    --handler-name GeneralInstructionHandler \
    --make-vocab-size-divisible-by 128 \
    --log-interval 1000
```

参数说明：

- input：SFT微调数据的存放路径。

- output-prefix：处理后的数据集保存路径+数据集名称前缀（例如：alpaca_ft）。

- tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase', 'BertWordPieceCase', 'GPT2BPETokenizer', 'PretrainedFromHF']，设置为PretrainedFromHF。

- tokenizer-name-or-path：tokenizer的存放路径。

- handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。

- seq-length：是一个用于计算序列长度的函数。它接收一个序列作为输入，并返回序列的长度，需和训练时参数保持一致。

- workers：数据处理线程数。

--make-vocab-size-divisible-by：填充词汇大小，使模型中padded-vocab-size的值可被该值整除。这是出于计算效率的原因而添加的。

- log-interval：输出处理日志刷新间隔。

输出结果

alpaca_ft_packed_attention_mask_document.bin

alpaca_ft_packed_attention_mask_document.idx

alpaca_ft_packed_input_ids_document.bin

alpaca_ft_packed_input_ids_document.idx

alpaca_ft_packed_labels_document.bin

alpaca_ft_packed_labels_document.idx

数据处理具体操作

SFT全参微调数据处理具体操作步骤如下。

1. 创建处理后的数据存放目录/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/
   ```
   cd /home/ma-user/ws/  #进入容器工作目录
   mkdir -p processed_for_ma_input/Qwen-14B/data/finetune
   ```
2. 进入代码目录“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。
   ```
   #加载ascendspeed及megatron模型
   export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
   export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
   #进入到ModelLink目录下
   cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
   #执行以下命令
   python ./tools/preprocess_data.py \
     --input /home/ma-user/ws/training_data/finetune/alpaca_gpt4_data.json \
     --tokenizer-name-or-path /home/ma-user/ws/tokenizers/Qwen-14B \
     --output-prefix /home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft \
     --workers 8 \
     --log-interval 1000 \
     --tokenizer-type PretrainedFromHF \
     --handler-name GeneralInstructionHandler \
     --make-vocab-size-divisible-by 128 \
     --seq-length 4096 \
   ```

   数据处理完后，在/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/目录下生成转换后的数据文件。

3.17.4.2 SFT 微调权重转换

微调训练前需将HuggingFace格式权重转换为Megatron格式后再进行SFT微调训练。

本章节主要介绍如何将HuggingFace权重转换为Megatron格式。此处的HuggingFace权重文件和转换操作结果同时适用于SFT微调和LoRA微调训练。

HuggingFace 权重转换操作

这里以Qwen-14B为例，Qwen-7B和Qwen-72B只需按照实际情况修改环境变量参数即可。

1. 下载Qwen-14B的预训练权重和词表文件，并上传到/home/ma-user/ws/tokenizers/Qwen-14B目录下。具体下载地址请参见**表3-139**。如果已下载，忽略此步骤。
2. 创建权重转换后的输出目录/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights/。
```
cd /home/ma-user/ws/  #进入/home/ma-user/ws/目录
mkdir -p processed_for_ma_input/Qwen-14B/converted_weights
```
3. 进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink，在代码目录中执行util.py脚本。
```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下：
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
# 权重格式转换
python tools/checkpoint/util.py --model-type GPT \
    --loader qwen_hf \
    --saver megatron \
    --target-tensor-parallel-size 8 \   #与微调TP值保持一致
    --target-pipeline-parallel-size 1 \ #与微调PP值保持一致
    --load-dir /home/ma-user/ws/tokenizers/Qwen-14B \
    --save-dir /home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights \
    --tokenizer-model /home/ma-user/ws/tokenizers/Qwen-14B/qwen.tiktoken \
    --add-qkv-bias
```
参数说明：
   - --model-type：模型类型。
   - --loader：权重转换要加载检查点的模型名称。
   - --tensor-model-parallel-size：张量并行数，需要与训练脚本中的配置一样。
   - --pipeline-model-parallel-size：流水线并行数，需要与训练脚本中的配置一样。
   - --saver：检查模型保存名称。
   - --load-dir：加载转换模型权重路径。
   - --save-dir：权重转换完成之后保存路径。
   - --tokenizer-model：tokenizer 路径。
   - --add-qkv-bias：为qkv这样的键和值添加偏差。
4. 权重转换完成后，在/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights目录下查看转换后的权重文件。

图 3-255 转换后的权重文件

3.17.4.3 SFT 微调训练任务

本章节以Qwen-14B为例，介绍SFT微调训练全过程。对于Qwen-7B和Qwen-72B，操作过程与Qwen-14B相同，只需修改对应参数即可。

前提条件

- SFT微调训练使用的数据集为alpaca_data数据，已经完成数据处理，具体参见**SFT微调数据处理**。
- 已经将开源的原始HuggingFace权重转换为Megatron格式，具体参见**SFT微调权重转换**。

Step1 修改训练超参配置

SFT微调脚本qwen.sh，存放在xxx-Ascend/llm_train/AscendSpeed/scripts/qwen目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。SFT微调的计算量与预训练基本一致，故配置可以与预训练相同。

表 3-144 SFT 微调超参配置

参数	示例值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Qwen-14B	必填。加载tokenizer时，tokenizer存放地址。请根据实际规划修改。
MODEL_TYPE	14B	必填。模型加载类型，根据实际填写7B、14B或72B。
TRAIN_ITERS	300	非必填。训练迭代周期。根据实际需要修改。
MBS	2	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为2。取值建议如下： • Qwen-14B：2 • Qwen-7B：2 • Qwen-72B：1
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长；对于PP（流水线并行）值大于1的场景，适当增大GBS值吞吐性能会有所提升。

参数	示例值	参数说明
TP	8	非必填。表示张量并行。默认值为8，取值建议： • Qwen-14B：8 • Qwen-7B：4 • Qwen-72B：8
PP	1	非必填。表示流水线并行。默认值为1，取值建议： • Qwen-14B：1 • Qwen-7B：1 • Qwen-72B：大于等于5，例如5机填写5，8机填8。
RUN_TYPE	sft	必填。表示训练类型。sft表示SFT微调。
MASTER_ADDR	localhost	多机必填。主节点IP地址，多台机器中指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填。节点总数，如为双机，则写2。单机默认是1。
NODE_RANK	0	多机必填。节点序号，当前节点ID，一般从0开始。单机默认是0。以Qwen-72B 5机训练为例，节点ID依次为（0 1 2 3 4）；一般ID为0的节点设置为主节点IP。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights	必填。加载的权重文件路径。**SFT微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录，训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。
SEQ_LEN	4096	非必填。默认值为4096。

Step2 启动训练脚本

请根据**表3-144**修改超参值后，再启动训练脚本。

单机启动

以Qwen-14B为例，单机SFT微调启动命令如下。在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下执行。

```
MODEL_TYPE=14B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights
```

```
 TRAIN_ITERS=300 MBS=2 GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、MODEL_PATH为必填，TRAIN_ITERS、MBS、GBS、TP、PP、SEQ_LEN为非必填，有默认值。

多机启动

以Qwen-14B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，此处以双机为例。

在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下执行。

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=14B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights TRAIN_ITERS=300 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
 ...
 ...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=14B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights TRAIN_ITERS=300 MBS=2  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

训练完成后，请参考**查看日志和性能**章节，查看SFT微调的日志和性能。

3.17.5 LoRA 微调训练

本章节以Qwen-14B为例，介绍LoRA微调训练的全过程。对于Qwen-7B和Qwen-72B，操作过程与Qwen-14B相同，只需修改对应参数即可。

Step1 LoRA 微调数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

LoRA微调训练与SFT微调使用同一个数据集，如果已经在SFT微调时处理过数据，可以直接使用，无需重复处理。如果未处理过数据，请参见**SFT微调数据处理**章节先处理数据。

Step2 LoRA 微调权重转换

LoRA微调训练前，需要先把训练权重文件转换为Megatron格式。

LoRA微调训练和SFT全参微调使用的是同一个HuggingFace权重文件转换为Megatron格式后的结果也是通用的。

如果在SFT微调任务中已经完成了HuggingFace权重转换操作，此处无需重复操作，可以直接使用SFT微调中的权重转换结果。

如果前面没有执行HuggingFace权重转换任务，可以参考**SFT微调权重转换**章节完成。

Step3 LoRA 微调超参配置

LoRA微调训练脚本qwen.sh，存放在llm_train/AscendSpeed/scripts/qwen/目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。

表 3-145 LoRA 微调超参配置

参数	示例值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/Qwen-14B	必填。加载tokenizer时，tokenizer存放地址。 请根据实际规划修改。
MODEL_TYPE	14B	必填。表示模型加载类型，根据实际填写7B、14B或72B。
TRAIN_ITERS	300	非必填。训练迭代周期。根据实际需要修改。
MBS	4	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。默认值为4。取值建议如下： ● Qwen-14B：4 ● Qwen-7B：2 ● Qwen-72B：1
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量，影响每一次训练迭代的时长。对于PP（流水线并行）值大于1的场景，适当增大GBS值吞吐性能会有所提升。
TP	8	非必填。表示张量并行。默认值为8，取值建议： ● Qwen-14B：8 ● Qwen-7B：4 ● Qwen-72B：8

参数	示例值	参数说明
PP	1	非必填。表示流水线并行。默认值为1，取值建议： • Qwen-14B：1 • Qwen-7B：1 • Qwen-72B：大于等于5，例如5机填写5，8机填8。
RUN_TYPE	lora	必填。表示训练类型。lora表示LoRA微调。
MASTER_ADDR	localhost	多机必填。主节点IP地址，多台机器中指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填。节点总数，如为双机，则写2。单机默认是1。
NODE_RANK	0	多机必填。节点序号，当前节点ID，一般从0开始。单机默认是0。以Qwen-72B 5机训练为例，节点ID依次为（0 1 2 3 4）；一般ID为0的节点设置为主节点IP。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights	必填。加载的权重文件路径。**SFT微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录，训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。
SEQ_LEN	4096	非必填。默认值为4096。

说明

在qwen.sh脚本默认情况下Lora微调的配置为：

--lora-r 16

--lora-alpha 32

LoRA微调训练的计算量要小于预训练，可以适当增加MBS的值，这里建议：

- 对于7B：TP=4 PP=1 MBS=2
- 对于14B：TP=8 PP=1 MBS=4
- 对于72B：TP=8 PP=5 MBS=1

Step4 启动训练脚本

请根据**表3-145**修改超参值后，再启动训练脚本。

单机启动

以Qwen-14B为例，单机SFT微调启动命令如下。在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下执行。

```
MODEL_TYPE=14B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights TRAIN_ITERS=300 MBS=4  GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、SEQ_LEN为非必填，有默认值。

多机启动

以Qwen-14B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，此处以双机为例。在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/代码目录下执行。

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=14B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights TRAIN_ITERS=300 MBS=4 GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
 ...
 ...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=14B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/Qwen-14B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/Qwen-14B/converted_weights TRAIN_ITERS=300 MBS=4 GBS=64 TP=8 PP=1 SEQ_LEN=4096 WORK_DIR=/home/ma-user/ws sh scripts/qwen/qwen.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、 NODE_RANK、 NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

训练完成后，请参考**查看日志和性能**章节，查看LoRA微调训练的日志和性能。

3.17.6 推理前的权重合并转换

模型训练完成后，训练的产物包括模型的权重、优化器状态、loss等信息。这些内容可用于断点续训、模型评测或推理任务等。

在进行模型评测或推理任务前，需要将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。

权重文件的合并转换操作都要求在训练的环境中进行，为下一步推理做准备。

- 如果需要使用本文档中训练后的权重文件进行推理，请参考此章节合并训练权重文件并转换为Huggingface格式。
- 如果无推理任务或者使用开源Huggingface权重文件推理，都可以忽略此章节。

下一步的推理任务请参考文档《**开源大模型基于DevServer的推理通用指导**》。

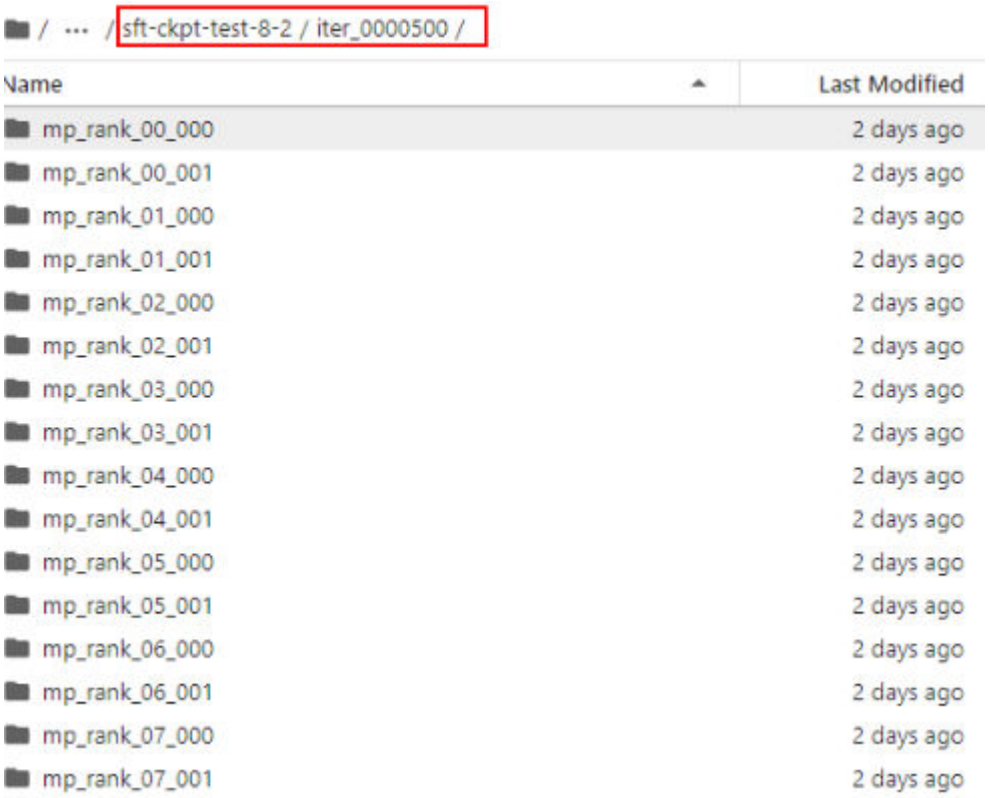
将多个权重文件合并为一个文件并转换格式

该场景一般用于将预训练、SFT或LoRA训练好的Megatron模型重新转回HuggingFace格式。

一般训练都是多卡分布式训练，权重结果文件为多个且文件为Megatron格式，因此需要合并多个文件并转换为HuggingFace格式。

如果是多机训练，转换前需将多机权重目录（iter_xxxxxxx）下的mp_rank_xx_xxx文件夹整合到一起后再进行转换，合并后结果如下图所示。

图 3-256 合并权重文件



该脚本的执行需要在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink目录下执行。

```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入ModelLink下：
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#执行以下命令
python tools/checkpoint/util.py --model-type GPT \
    --loader megatron \
    --saver megatron \
    --save-model-type save_huggingface_qwen \
    --load-dir /home/ma-user/ws/saved_dir_for_ma_output/Qwen-14B/lora \
    --target-tensor-parallel-size 1 \
    --target-pipeline-parallel-size 1 \
     --add-qkv-bias \
    --save-dir /home/ma-user/ws/tokenizers/Qwen-14B/  # <-- 需要填入原始HF模型路径，新权重会存于../Qwen-14B/mg2hg下
```

参数说明：

- save-model-type：输出后权重格式如（save_huggingface_qwen、save_huggingface_llama等）。
- load-dir：训练完成后保存的权重路径
- save-dir：需要填入原始HF模型路径，新权重会存于../Qwen-14B/mg2hg下。
- target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size。默认为1
- target-pipeline-parallel-size ：任务不同调整参数target-pipeline-parallel-size。默认为1
- add-qkv-bias：为像qkv这样的键和值添加偏差。
- loader：权重转换时要加载检查点的模型名称。
- saver：权重转换时加载检查模型保存名称。

转换后的权重文件结构

```
├── config.json
├── configuration_baichuan.py
├── generation_config.json
├── generation_utils.py
├── model-00001-of-00006.safetensors
├── model-00002-of-00006.safetensors
├── model-00003-of-00006.safetensors
├── model-00004-of-00006.safetensors
├── model-00005-of-00006.safetensors
├── model-00006-of-00006.safetensors
├── model.safetensors.index.json
├── modeling_baichuan.py
├── quantizer.py
```

3.17.7 常见问题

3.17.7.1 访问容器目录时提示 Permission denied

由于在容器中没有相应目录的权限，会导致访问时提示Permission denied。可以在宿主机中对相关目录做权限放开，执行命令如下。

```
chmod 777 -R ${dir}
```

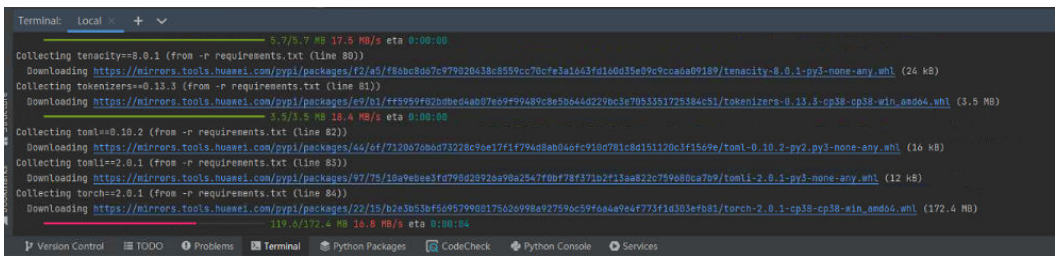
3.17.7.2 如何在容器中安装依赖包

在pycharm项目中打开Terminal窗口，在项目根目录执行以下命令安装依赖包。

```
pip install -r requirements.txt
```

安装成功后的示意图如图3-257所示。

图 3-257 依赖包安装成功



3.17.7.3 训练时报“EI0006: Getting socket times out”

该报错为HCCL通讯时间超时，默认时间为120s；因此需在启动训练任务前执行，在容器内设置HCCL通讯超时时间。

```
export HCCL_CONNECT_TIMEOUT=7200
```

3.18 GLM3-6B 模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904）

3.18.1 场景介绍

ChatGLM3-6B大模型是一个包含多种参数数量模型的语言模型。

方案概览

本文档以ChatGLM3-6B（以下简称GLM3-6B）为例，利用训练框架Pytorch_npu+华为自研Ascend Snt9b硬件，为用户提供了开箱即用的预训练和全量微调方案。

本方案目前配套的是AscendCloud-3rdLLM-6.3.T041版本，仅适用于部分企业客户，完成本方案的部署，需要先联系您所在企业的华为方技术支持。

操作流程

图 3-258 操作流程图

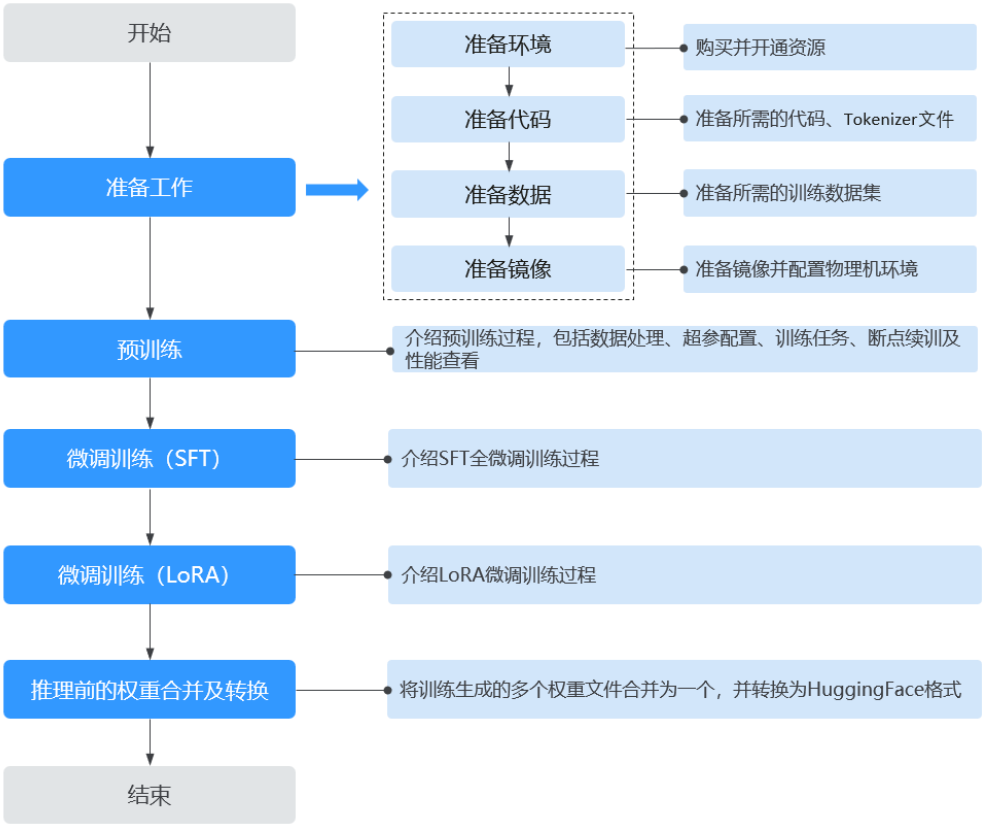


表 3-146 操作任务流程说明

阶段	任务	说明
准备工作	准备环境	本教程案例是基于ModelArts Lite DevServer运行的，需要购买并开通DevServer资源。
	准备代码	准备AscendSpeed训练代码、分词器Tokenizer和推理代码。
	准备数据	准备训练数据，可以用Alpaca数据集，也可以使用自己准备的数据集。
	准备镜像	准备训练模型适用的容器镜像。
预训练	预训练	介绍如何进行预训练，包括训练数据处理、超参配置、训练任务、断点续训及性能查看。
微调训练	SFT全参微调	介绍如何进行SFT全参微调。
	LoRA微调训练	介绍如何进行LoRA微调训练。
推理前的权重转换	-	模型训练完成后，可以将训练产生的权重文件用于推理。推理前参考本章节，将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。 如果无推理任务或者使用开源Huggingface权重文件进行推理，可以忽略此章节。和本文档配套的推理文档请参考《**开源大模型基于DevServer的推理通用指导**》。

3.18.2 准备工作

3.18.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的DevServer。请参考本文档要求准备DevServer机器。

资源规格要求

计算规格：单机训练需要使用单机8卡，多机训练需要使用2机16卡。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通 DevServer 资源

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

说明

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

3.18.2.2 准备代码

本教程中用到的数据和代码如下表所示，请提前准备好。

获取数据及代码

表 3-147 准备代码

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.904-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**代码目录介绍**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support-E网站**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
权重和词表文件	包含了本教程使用到的HuggingFace原始权重文件和Tokenizer。 标记器(Tokenizer)是NLP管道的核心组件之一。它们有一个目的：将文本转换为模型可以处理的数据。模型只能处理数字，因此标记器(Tokenizer)需要将文本输入转换为数字数据。	**chatglm3-6b-hf** 这个路径下既有权重，也有Tokenizer，全部下载。具体内容参见**权重和词表文件介绍**。

说明

本文档前向兼容AscendCloud-3rdLLM-6.3.T041版本，获取路径：**Support网站**。

代码目录介绍

AscendCloud-3rdLLM代码包结构介绍如下：

```
xxx-Ascend              #xxx表示版本号，例如6.3.T041
    ├──llm_evaluation  #推理评测代码包
        ├──benchmark_eval   #精度评测
        ├──benchmark_tools  #性能评测
    ├──llm_train       #模型训练代码包
        ├──AscendSpeed  #基于AscendSpeed的训练代码
            ├──AscendSpeed  #加速库
            ├──ModelLink   #基于ModelLink的训练代码
            ├──scripts/    #训练需要的启动脚本
```

本教程需要使用到的训练相关代码存放在llm_train/AscendSpeed目录下，具体文件介绍如下：

```
├──llm_train       #模型训练代码包
        ├──AscendSpeed   #基于AscendSpeed的训练代码
            ├──AscendSpeed  #加速库
```

```
        ├──ModelLink  #基于ModelLink的训练代码，数据预处理脚本
        ├──scripts/   #训练需要的启动脚本，调用ModelLink
            ├──glm3  #glm3的训练代码
                ├──glm3_base.sh              #glm3训练脚本
```

权重和词表文件介绍

下载完毕后的HuggingFace原始权重文件包含以下内容，此处以GLM3-6B为例。

```
GLM3-6B
├── config.json
├── configuration_chatglm.py
├── model-00001-of-00007.safetensors
├── model-00002-of-00007.safetensors
├── model-00003-of-00007.safetensors
├── model-00004-of-00007.safetensors
├── model-00005-of-00007.safetensors
├── model-00006-of-00007.safetensors
├── model-00007-of-00007.safetensors
├── modeling_chatglm.py
├── MODEL_LICENSE
├── pytorch_model-00001-of-00007.bin
├── pytorch_model-00002-of-00007.bin
├── pytorch_model-00003-of-00007.bin
├── pytorch_model-00004-of-00007.bin
├── pytorch_model-00005-of-00007.bin
├── pytorch_model-00006-of-00007.bin
├── pytorch_model-00007-of-00007.bin
├── pytorch_model.bin.index.json
├── quantization.py
├── README.md
├── special_tokens_map.json
├── tokenization_chatglm.py
├── tokenizer_config.json
├── tokenizer.model
```

工作目录结构如下

```
${workdir}（例如/home/ma-user/ws ）
├──llm_train
   ├── AscendSpeed             #代码目录
      ├── AscendSpeed          #训练依赖的三方模型库
      ├── ModelLink            #AscendSpeed代码目录
      ├── scripts/             #训练启动脚本
├── processed_for_ma_input
   ├── GLM3-6B
      ├── data              #预处理后数据
         ├── pretrain    #预训练加载的数据
         ├── finetune    #微调加载的数据
      ├──converted_weights  #HuggingFace格式转换magatron格式后权重文件
├── saved_dir_for_ma_output         #训练输出保存权重，根据实际训练需求设置
   ├── GLM3-6B
      ├── logs              #训练过程中日志（loss、吞吐性能）
      ├── lora              #lora微调输出权重
      ├── sft               #增量训练输出权重
      ├── pretrain          #预训练输出权重
├── tokenizers                    #原始权重及tokenizer目录
   ├── GLM3-6B
├── training_data                  #原始数据目录
   ├── pretrain                 #预训练加载的数据
      ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预训练原始数据文件
   ├── finetune                  #微调训练加载的数据
      ├── Alpaca_data_gpt4_zh.jsonl                      #微调训练原始数据文件
```

上传代码到工作环境

1. 使用root用户以SSH的方式登录DevServer。将AscendSpeed代码包AscendCloud-3rdLLM-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例。
   ```
   unzip AscendCloud-3rdLLM-xxx-xxx.zip  #解压缩，-xxx-xxx表示软件包版本号和时间戳
   ```
2. 上传tokenizers及权重和词表文件到工作目录中的/home/ma-user/ws/tokenizers/GLM3-6B目录。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws。将tokenizers及权重和词表文件放置此处。
   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/GLM3-6B
   ```

3.18.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据

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

- 训练数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。
- SFT全参微调、LoRA微调训练数据集下载： **https://huggingface.co/datasets/silk-road/alpaca-data-gpt4-chinese/blob/main/Alpaca_data_gpt4_zh.jsonl**，数据大小：42M左右。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

经下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data/pretrain”，并将预训练原始数据放置在此处。
   ```
   mkdir -p training_data/pretrain
   ```
   创建目录“training_data/finetune”，并将微调训练原始数据放置在此处
   ```
   mkdir -p training_data/finetune
   ```

数据存放参考目录结构如下：

```
${workdir}（例如/home/ma-user/ws）
  ├── training_data                    #原始数据目录
      ├── pretrain                     #预训练加载的数据
          ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预训练原始数据文件
      ├── finetune                     #微调训练加载的数据
          ├── Alpaca_data_gpt4_zh.jsonl                      #微调训练原始数据文件
```

3.18.2.4 准备镜像

准备训练GLM3-6B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置DevServer物理机环境操作。

镜像地址

本教程中用到的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-148 基础容器镜像地址

镜像用途	镜像地址
基础镜像（训练和推理通用）	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42

说明

本文档兼容cann_7.0.1.1和cann_8.0.rc1的镜像，推荐使用较新版本的cann_8.0.rc1镜像。

表 3-149 模型镜像版本

模型	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0
PyTorch_npu	2.1.0.post3-20240413

Step1 检查环境

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

2. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**表3-148**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
container_work_dir="/home/ma-user/ws"   # 容器内挂载的目录
work_dir="/home/ma-user/ws"             # 宿主机挂载目录，存放了代码、数据、权重
container_name="${container_name}"      # ${container_name}为启动的容器名称
image_name="${image_name}"              # ${image_name}启动的镜像ID或name
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 32g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

2. 通过容器名称进入容器中。
```
docker exec -it ${container_name} bash
```

说明

启动容器时默认用户为ma-user用户。如果需要切换到root用户可以执行以下命令：

```
sudo su
source /home/ma-user/.bashrc
```

如果继续使用ma-user，在使用其他属组如root用户上传的数据和文件时，可能会存在权限不足的问题，因此需要执行如下命令统一文件属主。

```
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
例如：
sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 安装依赖包。
```
#进入scriptsscripts目录，xxx为包版本，请按照实际情况替换
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/scripts
#执行安装命令
pip install -r requirements.txt
```

3.18.3 预训练

3.18.3.1 预训练数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

Alpaca 数据处理说明

数据预处理脚本preprocess_data.py存放在代码包的“llm_train/AscendSpeed/ModelLink/tools”目录中，脚本样例命令及参数详解如下，详细执行步骤请参考下一段落。

```
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path {work_dir}/tokenizers/GLM3-6B \
 --output-prefix {work_dir}/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca \
 --workers 4 \
 --tokenizer-type PretrainedFromHF \
 --append-eod \
 --seq-length 8192 \
 --tokenizer-not-use-fast
```

参数说明：

- ${work_dir}的路径指容器工作路径：如/home/ma-user/ws/ 。
- - input：原始数据集的存放路径
- - output-prefix：处理后的数据集保存路径+数据集名称前缀（ 例如: alpaca ），该目录路径需提前创建
- - tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- - tokenizer-name-or-path：tokenizer的存放路径
- -workers：设置数据处理使用执行卡数量
- -append-eod：参数用于控制是否在每个输入序列的末尾添加一个特殊的标记。这个标记表示输入序列的结束，可以帮助模型更好地理解和处理长序列。

- seq-length：是一个用于计算序列长度的函数。它接收一个序列作为输入，并返回序列的长度，需和训练时参数保持一致。

数据预处理后输出的训练数据如下：

- alpaca_text_document.bin
- alpaca_text_document.idx

训练的时指定的数据路径为${path}/alpaca/GLM3-6B/alpaca_text_document， 不加文件类型后缀。

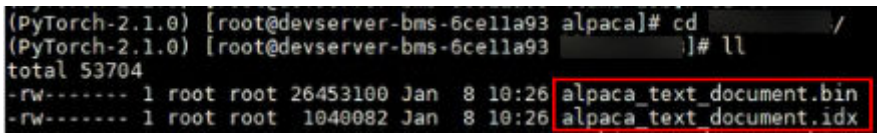
Alpaca 数据处理操作步骤

Alpaca数据处理具体操作步骤如下：

1. 创建数据处理后的输出目录/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/。
```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/GLM3-6B/data/pretrain
```
2. 将获取到的Alpaca预训练数据集传到上一步创建的目录中。如还未下载数据集，请参考**准备数据**获取。
3. 进入“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”目录，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。
```
#加载ascendspeed及megatron模型，xxx-Ascend请根据实际目录替换
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下:
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令:
python ./tools/preprocess_data.py \
 --input /home/ma-user/ws/training_data/pretrain/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path /home/ma-user/ws/tokenizers/GLM3-6B \
 --output-prefix /home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca \
 --workers 4 \
 --tokenizer-type PretrainedFromHF \
 --append-eod \
 --seq-length 8192 \
 --tokenizer-not-use-fast
```
4. 数据处理完后，在/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/目录下生成alpaca_text_document.bin和alpaca_text_document.idx文件。

   图 3-259 处理后的数据



自定义数据

如果是用户自己准备的数据集，可以使用Ascendspeed代码仓中的转换工具将json格式数据集转换为训练中使用的.idx + .bin格式。

```
#示例:
#1.将准备好的json格式数据集存放于/home/ma-user/ws/training_data/pretrain目录下: 如data.json
```



```
#2.运行转换脚本
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/

#加载ascendspeed及megatron模型，xxx-Ascend请根据实际目录替换
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/pretrain/data.json \
 --tokenizer-name-or-path {work_dir}/tokenizers/GLM3-6B \
 --output-prefix {work_dir}/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca \
 --workers 4 \
 --tokenizer-type PretrainedFromHF \
 --append-eod \
 --seq-length 4096 \
 --tokenizer-not-use-fast
#3.执行完成后在 datasets文件夹中可以得到 data_text_document.idx 与data_text_document.bin 两个文件
```

3.18.3.2 预训练任务

配置预训练脚本glm3_base.sh中的超参，并执行预训练任务。

Step1 配置预训练超参

预训练脚本glm3_base.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/glm3”目录下。训练前，可以根据实际需要修改超参配置。xxx-Ascend请根据实际目录替换。

表 3-150 预训练超参配置

参数	示例值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca_text_document	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/GLM3-6B	必填。加载tokenizer时，tokenizer存放地址。 请根据实际规划修改。
MODEL_TYPE	6B	必填。表示模型加载类型。
TRAIN_ITERS	200	非必填。表示训练迭代周期，根据实际需要修改。
MBS	1	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。 默认值1。单机建议为1，双机建议为2。

参数	示例值	参数说明
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。默认值64。单机建议为64，双机建议为128。
TP	2	非必填。表示张量并行。默认值为2。
PP	4	非必填。表示流水线并行。默认值为4。单机建议为4，双机建议为8。
RUN_TYPE	pretrain	必填。表示训练类型，根据实际训练任务类型选择。取值说明： ● pretrain：表示预训练 ● retrain：表示断点续训 ● sft：表示SFT微调训练 ● lora：表示LoRA微调训练
MASTER_ADDR	localhost	多机必填，单机忽略；指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填，单机忽略；节点总数，单机写1，双机写2。
NODE_RANK	0	多机必填，单机忽略；节点序号，当前节点ID，一般从0开始，单机默认是0。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。
SEQ_LEN	8192	非必填。默认值为8192。

Step2 启动训练脚本

请根据表3-150修改超参值后，再启动训练脚本。

单机启动

以GLM3-6B为例，单机训练启动样例命令如下，以自己实际为准。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
MODEL_TYPE=6B  RUN_TYPE=pretrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B TRAIN_ITERS=200 MBS=1  GBS=64 TP=2 PP=4 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

其中MODEL_TYPE、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH为必填。TRAIN_ITERS、MBS、GBS、TP、PP、SEQ_LEN 为非必填，有默认值。

多机启动

以GLM3-6B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以下命令以双机为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=6B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B TRAIN_ITERS=200
MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
...
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=6B  RUN_TYPE=pretrain
DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/
alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B TRAIN_ITERS=200
MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

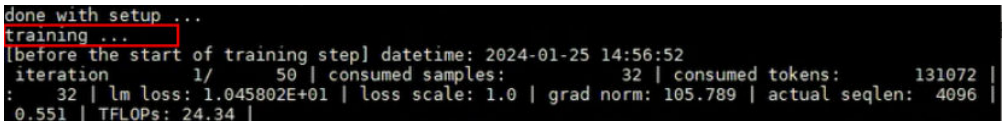
以上命令多台机器执行时，只有${NODE_RANK}：节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-260 等待模型载入



更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

如果需要使用断点续训练能力，请参考**断点续训练**章节修改训练脚本。

3.18.3.3 断点续训练

断点续训练是指因为某些原因导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

断点续训练操作过程

GLM3-6B的断点续训脚本glm3_base.sh，存放在“xxx-Ascend/llm_train/AscendSpeed/scripts/glm3”目录下。

1. 执行命令如下，进入AscendSpeed代码目录。xxx-Ascend请根据实际目录替换。
```
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/
```
2. 修改断点续训练参数。断点续训前，需要在原有训练参数配置表3-150中新加“MODEL_PATH”参数，并修改“TRAIN_ITERS”参数和“RUN_TYPE”参数。

表 3-151 断点续训练修改参数

参数	参考值	参数说明
MODEL_PATH	/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/pretrain	必填。加载上一步预训练后保存的权重文件。 请根据实际规划修改。
TRAIN_ITERS	300	必填。表示训练周期，必须大于上次保存训练的周期次数。
RUN_TYPE	retrain	必填。训练脚本类型，retrain表示断点续训练。

3. 在AscendSpeed代码目录下执行断点续训练脚本。

单机启动
```
MODEL_TYPE=6B RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/pretrain TRAIN_ITERS=300 MBS=1 GBS=64 TP=2 PP=4 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

多机启动

以GLM3-6B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。
```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=6B  RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/pretrain TRAIN_ITERS=300 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=6B  RUN_TYPE=retrain DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/pretrain/alpaca_text_document TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/pretrain TRAIN_ITERS=300 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

图 3-261 保存的 ckpt

```
[root@devserver-modelarts ckpt]# ll
total 24
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 12:34 iter_0000005
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:04 iter_0000010
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:34 iter_0000015
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:04 iter_0000020
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:56 iter_0000025
-rw-r-----  1 HwHiAiUser users    2 Oct 20 14:20 latest_checkpointed_iteration.txt
```

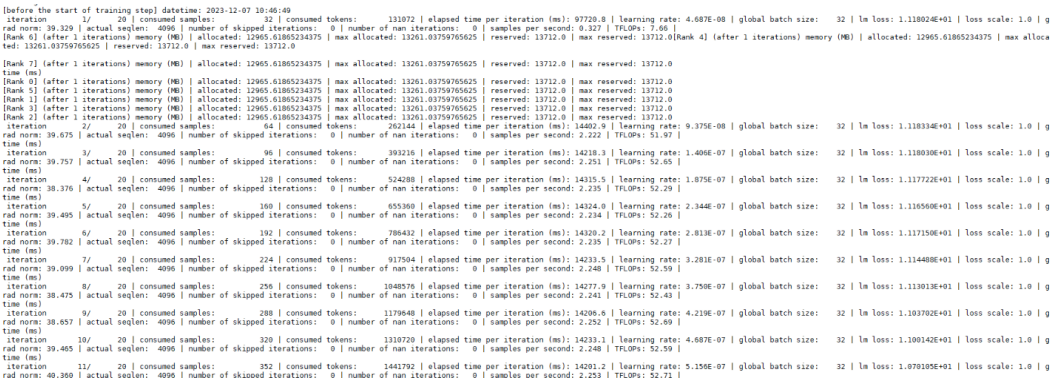
4. 训练完成后，可以参考**查看日志和性能**操作，查看断点续训练日志和性能。

3.18.3.4 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-262 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为{work_dir}/saved_dir_for_ma_output/GLM3-6B/logs,本实例日志路径为/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/logs

查看性能

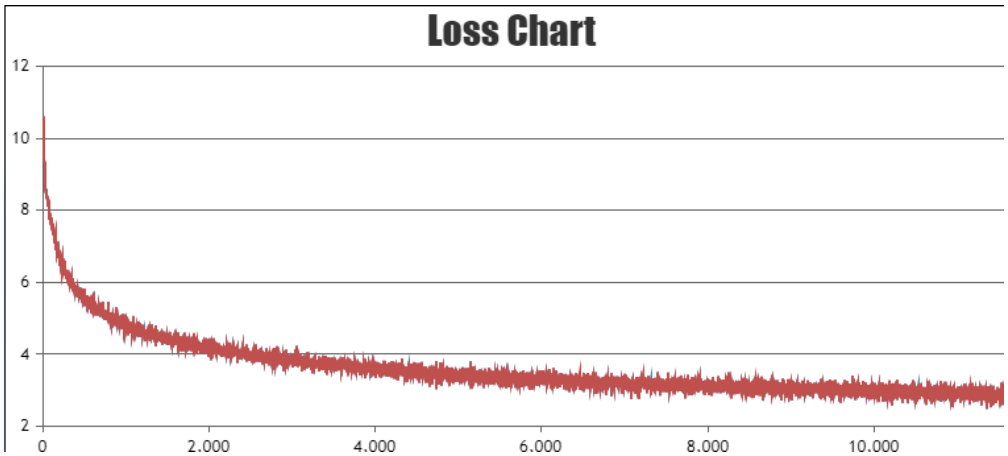
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/（总卡数*elapsed time per iteration）*1000，其参数在日志里可找到，默认seq_len值为8192，默认global batch size为64；其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-263**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

图 3-263 Loss 收敛情况（示意图）



3.18.4 SFT 全参微调训练

3.18.4.1 SFT 全参微调数据处理

SFT全参微调（SFT fine-tuning）前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

下载数据

SFT全参微调涉及的数据下载地址：**https://huggingface.co/datasets/silk-road/alpaca-data-gpt4-chinese/blob/main/Alpaca_data_gpt4_zh.jsonl**

如果在**准备数据**章节已下载数据集，此处无需重复操作。

SFT全参微调和LoRA微调训练使用的是同一个数据集，数据处理一次即可，训练时可以共用。

数据预处理说明

使用数据预处理脚本preprocess_data.py脚本重新生成.bin和.idx格式的SFT全参微调数据。preprocess_data.py存放在xxx-Ascend/llm_train/AscendSpeed/ModelLink/tools目录中，脚本具体内容如下。xxx-Ascend请根据实际目录替换。

```
#进入ModelLink目录:
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
    --input /home/ma-user/ws/training_data/finetune/Alpaca_data_gpt4_zh.jsonl \
    --tokenizer-name-or-path $TOKENIZER_PATH \
    --output-prefix $DATASET_PATH\
    --tokenizer-type PretrainedFromHF \
    --workers 8 \
    --seq-length 8192 \
    --handler-name GeneralInstructionHandler \
    --append-eod \
    --tokenizer-not-use-fast
```

参数说明：

- - input：SFT全参微调数据的存放路径。

- - output-prefix：处理后的数据集保存路径+数据集名称前缀（例如：alpaca_ft）。
- - tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'， 'PretrainedFromHF']，设置为PretrainedFromHF。
- - tokenizer-name-or-path：tokenizer的存放路径。
- - handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。
- - workers：数据处理线程数。
- seq-length：是一个用于计算序列长度的函数。它接收一个序列作为输入，并返回序列的长度，需和训练时参数保持一致。
- -append-eod：参数用于控制是否在每个输入序列的末尾添加一个特殊的标记。这个标记表示输入序列的结束，可以帮助模型更好地理解和处理长序列。

输出结果

alpaca_ft_packed_attention_mask_document.bin

alpaca_ft_packed_attention_mask_document.idx

alpaca_ft_packed_input_ids_document.bin

alpaca_ft_packed_input_ids_document.idx

alpaca_ft_packed_labels_document.bin

alpaca_ft_packed_labels_document.idx

数据处理具体操作

SFT全参微调数据处理具体操作步骤如下。

1. 创建处理后的数据存放目录/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/

```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/GLM3-6B/data/finetune
```

2. 进入代码目录“/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink/”，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。

```
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
python ./tools/preprocess_data.py \
  --input /home/ma-user/ws/training_data/finetune/Alpaca_data_gpt4_zh.jsonl \
  --tokenizer-name-or-path /home/ma-user/ws/tokenizers/GLM3-6B \
  --output-prefix /home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft \
  --workers 8 \
  --tokenizer-type PretrainedFromHF \
  --handler-name GeneralInstructionHandler \
  --seq-length 8192 \
  --append-eod \
  --tokenizer-not-use-fast
```

   数据处理完后，在/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/目录下生成转换后的数据文件。

3.18.4.2 SFT 全参微调权重转换

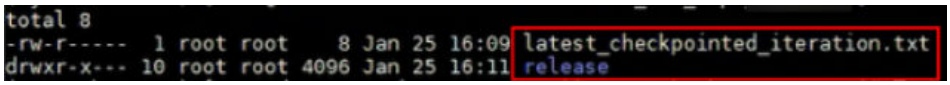
增量训练前需将HuggingFace格式权重转换为Megatron格式后再进行SFT全参微调。

本章节主要介绍如何将HuggingFace权重转换为Megatron格式。此处的HuggingFace权重文件和转换操作结果同时适用于SFT全参微调和LoRA微调训练。

HuggingFace 权重转换操作

1. 下载GLM3-6B的预训练权重和词表文件，并上传到/home/ma-user/ws/tokenizers/GLM3-6B目录下。具体下载地址请参见**表3-147**。如果已下载，忽略此步骤。
2. 创建权重转换后的输出目录/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights/。
   ```
   cd /home/ma-user/ws/  #进入/home/ma-user/ws/目录
   mkdir -p processed_for_ma_input/GLM3-6B/converted_weights
   ```
3. 进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink，在此代码目录下执行2_convert_mg_hf.sh脚本。xxx-Ascend请根据实际目录替换。
   ```
   #进入ModelLink目录下
   cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
   # 执行权重格式转换脚本
   TP=2 PP=4 LOAD_DIR=/home/ma-user/ws/tokenizers/GLM3-6B SAVE_DIR=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B CONVERT_HFtoMG=True  sh ../scripts/glm3/2_convert_mg_hf.sh
   ```
   其脚本2_convert_mg_hf.sh参数说明：
   - --model-type：模型类型。
   - --loader：权重转换要加载检查点的模型名称。
   - --tensor-model-parallel-size：${TP} 张量并行数，需要与训练脚本中的配置一样。
   - --pipeline-model-parallel-size：${PP} 流水线并行数，需要与训练脚本中的配置一样。
   - --saver：检查模型保存名称。
   - --load-dir：${LOAD_DIR} 加载转换模型权重路径。
   - --save-dir : ${SAVE_DIR} 权重转换完成之后保存路径。
   - --tokenizer-model : ${TOKENIZER_PATH} tokenizer路径。
   - --add-qkv-bias : 为qkv这样的键和值添加偏差。
   - CONVERT_HFtoMG：权重转换类型是否为HuggingFace权重转换为Megatron格式，True表示HuggingFace权重转换为Megatron，反之False为Megatron格式转换HuggingFace格式。
4. 权重转换完成后，在/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights目录下查看转换后的权重文件。

   图 3-264 转换后的权重文件

3.18.4.3 SFT 全参微调任务

前提条件

- SFT全参微调使用的数据集为alpaca_data数据，已经完成数据处理，具体参见**SFT全参微调数据处理**。
- 已经将开源原始HuggingFace权重转换为Megatron格式，具体参见**SFT全参微调权重转换**。

Step1 修改训练超参配置

SFT全参微调脚本glm3_base.sh，存放在Ascenxxx-Ascend/llm_train/AscendSpeed/scripts/glm3目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。SFT微调的计算量与预训练基本一致，故配置可以与预训练相同。

表 3-152 SFT 全参微调超参配置

参数	值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/GLM3-6B	必填。加载tokenizer时，tokenizer存放地址。请根据实际规划修改。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights	必填。加载的权重文件路径。**SFT全参微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
MODEL_TYPE	6B	必填。模型加载类型。
TRAIN_ITERS	200	非必填。训练迭代周期。根据实际需要修改。
MBS	1	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。 建议值单机1，双机2。

参数	值	参数说明
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。 建议值单机64，双机128。
TP	2	非必填。表示张量并行。默认值为2。
PP	4	非必填。表示流水线并行。建议值单机4，双机8。
RUN_TYPE	sft	必填。表示训练类型，sft表示SFT微调训练。
MASTER_ADDR	localhost	多机必填，单机忽略。指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	q	多机必填，单机忽略。节点总数，单机写1，双机写2，8机写8。
NODE_RANK	0	多机必填，单机忽略。节点序号，当前节点ID，一般从0开始，单机默认是0。以8机训练为例，节点ID依次为（0 1 2 3 4 5 6 7）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。
SEQ_LEN	8192	非必填。默认值为8192。

Step2 启动训练脚本

请根据表3-152修改超参值后，再启动训练脚本。

单机启动

以GLM3-6B为例，单机SFT全参微调启动命令如下。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
MODEL_TYPE=6B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=200 MBS=1 GBS=64 TP=2 PP=4 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、SEQ_LEN为非必填，有默认值。

多机启动

以GLM3-6B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以下命令以双机为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=6B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=200 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
 ...
 ...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=6B  RUN_TYPE=sft DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=200 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

训练完成后，请参考**查看日志和性能**章节查看日志和性能。

3.18.5 LoRA 微调训练

本章节介绍LoRA微调训练的全过程。

Step1 LoRA 微调数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

LoRA微调训练与SFT微调使用同一个数据集，如果已经在SFT微调时处理过数据，可以直接使用，无需重复处理。如果未处理过数据，请参见**SFT全参微调数据处理**章节先处理数据。

Step2 LoRA 微调权重转换

LoRA微调训练前，需要先把训练权重文件转换为Megatron格式。

LoRA微调训练和SFT全参微调使用的是同一个HuggingFace权重文件转换为Megatron格式后的结果也是通用的。

如果在SFT微调任务中已经完成了HuggingFace权重转换操作，此处无需重复操作，可以直接使用SFT微调中的权重转换结果。

如果前面没有执行HuggingFace权重转换任务，可以参考**SFT全参微调权重转换**章节完成。

Step3 LoRA 微调超参配置

LoRA微调训练脚本glm3_base.sh，存放在xxx-Ascend/llm_train/AscendSpeed/scripts/glm3/目录下。训练前，可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。

表 3-153 LoRA 微调超参配置

参数	值	参数说明
DATASET_PATH	/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_PATH	/home/ma-user/ws/tokenizers/GLM3-6B	必填。加载tokenizer时，tokenizer存放地址。请根据实际规划修改。
MODEL_PATH	/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights	必填。加载的权重文件路径。**Step2 LoRA微调权重转换**章节中将HuggingFace格式转化为Megatron格式的权重文件。
MODEL_TYPE	6B	必填。模型加载类型。
TRAIN_ITERS	300	非必填。训练迭代周期。根据实际需要修改。
MBS	1	非必填。表示流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。 默认值为1。单机建议值为1，双机为2。
GBS	64	非必填。表示训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长。 建议值单机64，双机128。
TP	2	非必填。表示张量并行。默认值为2。
PP	4	非必填。表示流水线并行。建议值单机4，双机8。
RUN_TYPE	lora	必填。表示训练类型，lora表示LoRA微调训练。
MASTER_ADDR	localhost	多机必填，单机忽略；指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填，单机忽略；节点总数，单机写1，双机写2，8机写8。

参数	值	参数说明
NODE_RANK	0	多机必填，单机忽略；节点序号，当前节点ID，一般从0开始，单机默认是0。以8机训练为例，节点ID依次为（0 1 2 3 4 5 6 7）；一般ID为0的节点设置为主节点IP。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。
SEQ_LEN	8192	非必填。默认值为8192。

Step4 启动训练脚本

请根据表3-153修改超参值后，再启动训练脚本。

单机启动

以GLM3-6B为例，单机SFT全参微调启动命令如下。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
MODEL_TYPE=6B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=300 MBS=1 GBS=64 TP=2 PP=4 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、 TP、PP 、SEQ_LEN为非必填，有默认值。

多机启动

以GLM3-6B为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，此处以双机为例。

进入代码目录/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed下执行启动脚本。xxx-Ascend请根据实际目录替换。

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=6B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=300 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
 ...
 ...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=6B  RUN_TYPE=lora DATASET_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/data/finetune/alpaca_ft TOKENIZER_PATH=/home/ma-user/ws/tokenizers/GLM3-6B MODEL_PATH=/home/ma-user/ws/processed_for_ma_input/GLM3-6B/converted_weights TRAIN_ITERS=300 MBS=2 GBS=128 TP=2 PP=8 SEQ_LEN=8192 WORK_DIR=/home/ma-user/ws sh scripts/glm3/glm3_base.sh
```

以上命令多台机器执行时，只有${NODE_RANK}的节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、

DATASET_PATH、TOKENIZER_PATH、MODEL_PATH为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR、SEQ_LEN为非必填，有默认值。

训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

3.18.6 推理前的权重合并转换

模型训练完成后，训练的产物包括模型的权重、优化器状态、loss等信息。这些内容可用于断点续训、模型评测或推理任务等。

在进行模型评测或推理任务前，需要将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。

权重文件的合并转换操作都要求在训练的环境中进行，为下一步推理做准备。

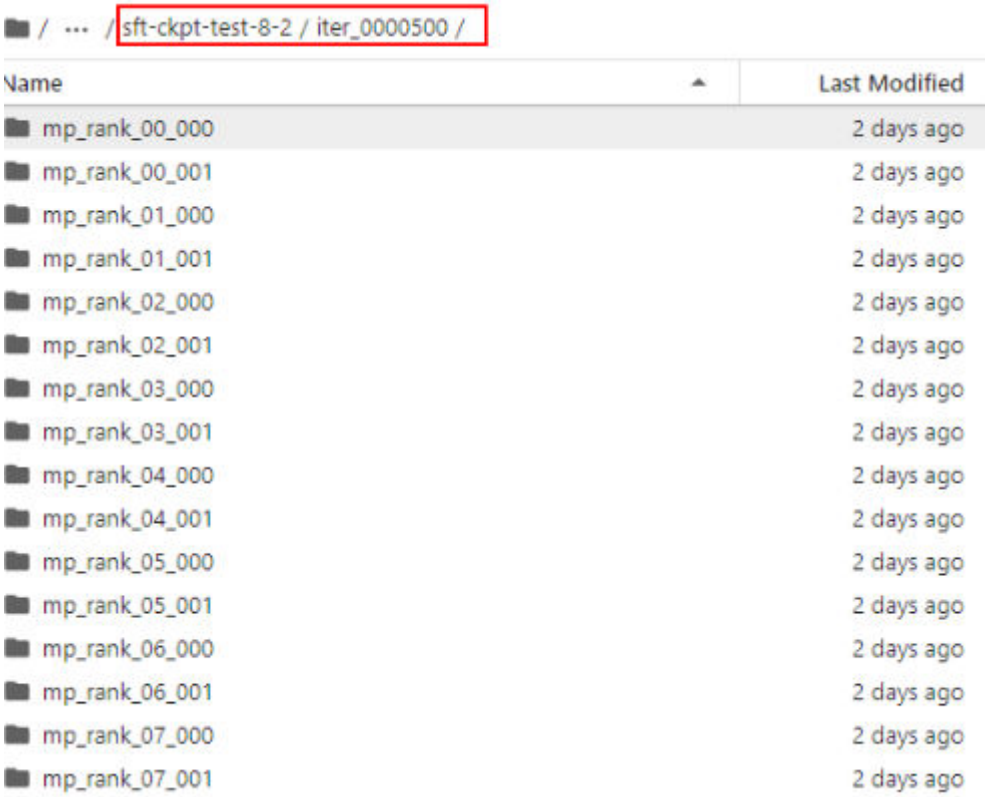
- 如果需要使用本文档中训练后的权重文件进行推理，请参考此章节合并训练权重文件并转换为Huggingface格式。
- 如果无推理任务或者使用开源Huggingface权重文件推理，都可以忽略此章节。

下一步的推理任务请参考文档《**开源大模型基于DevServer的推理通用指导**》。

将多个权重文件合并为一个文件并转换格式

任意并行切分策略的Megatron权重 格式转化为 HuggingFace权重（该场景一般用于将训练好的megatron模型：预训练、lora、sft 重新转回HuggingFace格式）为下一步推理使用准备，无推理任务忽略此章节。一般训练都是多卡分布式训练权重结果文件为多个且文件为Megatron格式，因此需要合并多个文件转换为huggingface格式。

如果是多机训练，训练产生的权重文件分布在多个节点，转换前需将多机权重目录（iter_xxxxxxx）下mp_rank_xx_xxx文件夹整合到一起后进行转换，合并后结果如图所示。单机多卡场景下，权重文件已经在同一个节点的目录下，不需要执行此操作。



该脚本的执行需要在/home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink目录下执行。具体执行步骤如下：

以lora微调训练权重结果Megatron权重 格式转化为 HuggingFace权重为例

```
#进入ModelLink目录下：
cd /home/ma-user/ws/xxx-Ascend/llm_train/AscendSpeed/ModelLink
# 执行权重格式转换脚本：2_convert_mg_hf.sh
LOAD_DIR=/home/ma-user/ws/saved_dir_for_ma_output/GLM3-6B/lora SAVE_DIR=/home/ma-user/ws/tokenizers/GLM3-6B CONVERT_HFTOMG=False  sh ../scripts/glm3/2_convert_mg_hf.sh
```

其脚本2_convert_mg_hf.sh参数说明：

- save-model-type：输出后权重格式如（ save_huggingface_qwen、save_huggingface_llama等 ）。
- megatron-path：megatron模型路径，在代码xxx-Ascend/llm_train/AscendSpeed/ModelLink目录下
- load-dir：${LOAD_DIR} 训练完成后保存的权重路径.如lora微调、sft、预训练生成的权重结果。
- save-dir：${SAVE_DIR} 需要填入原始HF模型路径，新权重会存于../GLM3-6B/mg2hg下。
- target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size。默认为1
- target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size。默认为1
- add-qkv-bias：为像qkv这样的键和值添加偏差。
- loader：权重转换时要加载检查点的模型名称。
- saver：权重转换时加载检查模型保存名称。
- CONVERT_HFtoMG：权重转换类型是否为HuggingFace权重转换为Megatron格式，True ： HuggingFace权重转换为Megatron，反之False为Megatron格式转换HuggingFace格式

转换后的权重文件结构

```
├── config.json
├── configuration_chatglm.py
├── generation_config.json
├── model-00001-of-00003.safetensors
├── model-00002-of-00003.safetensors
├── model-00003-of-00003.safetensors
├── model.safetensors.index.json
├── modeling_chatglm.py
├── quantization.py
```

3.19 Baichuan2-13B 模型基于 DevServer 适配 PyTorch NPU 训练指导（6.3.904）

3.19.1 场景介绍

Baichuan2是百川智能推出的 新一代Q开源大语言模型,采用 2.6 万亿 Tokens 的高质量语料训练。在多个权威的中文、英文和多语言的通用、领域 benchmark 上取得同尺寸最佳的效果。包含有 7B、13B 的 Base 和 Chat 版本,并提供了 Chat 版本的 4bits 量化。

本文档以Baichuan2-13B为例，利用训练框架Pytorch_npu+华为自研Ascend Snt9b硬件，为用户提供了开箱即用的预训练和全量微调方案。同时利用昇腾高性能算子库Ascend Transformer Boost（ATB）和适配昇腾平台的大模型推理服务Text Generation Inference（TGI） + 华为自研Ascend Snt9b硬件，为用户提供了开箱即用的推理部署方案，包括推理的性能和精度测试等，为用户提供端到端的大模型解决方案，帮助用户使能大模型业务。

操作流程

图 3-265 操作流程图

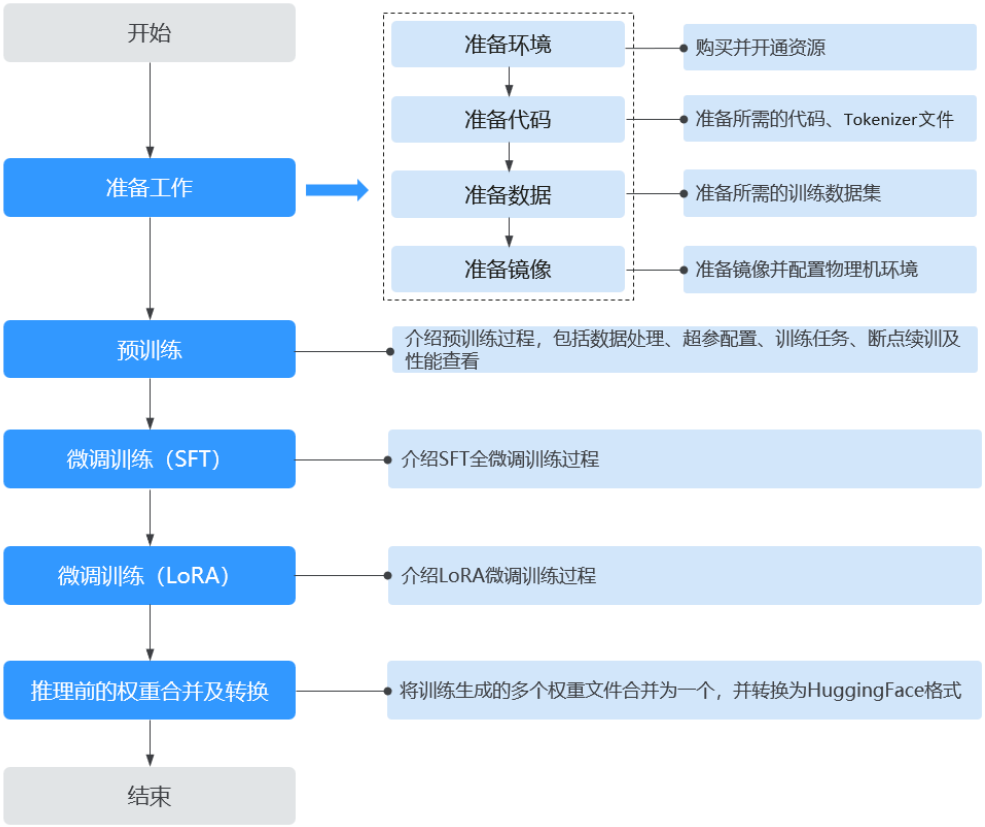


表 3-154 操作任务流程说明

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

阶段	任务	说明
微调训练	SFT全参微调	介绍如何进行SFT全参微调。
	LoRA微调训练	介绍如何进行LoRA微调训练。
推理前的权重转换	-	模型训练完成后，可以将训练产生的权重文件用于推理。推理前参考本章节，将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。 如果无推理任务或者使用开源Huggingface权重文件进行推理，可以忽略此章节。和本文档配套的推理文档请参考《**开源大模型基于DevServer的推理通用指导**》。

3.19.2 准备工作

3.19.2.1 准备环境

本文档中的模型运行环境是ModelArts Lite的DevServer。请参考本文档要求准备DevServer机器。

资源规格要求

计算规格：单机训练需要使用单机8卡，多机训练需要使用2机16卡。推理部署如果是376T规格，推荐使用单机单卡；280T规格推荐使用单机2卡。

硬盘空间：至少200GB。

Ascend资源规格：

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

购买并开通 DevServer 资源

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

说明

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

3.19.2.2 准备代码

本教程中用到的训练推理代码和如下表所示，请提前准备好。

获取数据及代码

表 3-155 准备代码

代码包名称	代码说明	下载地址
AscendCloud-3rdLLM-6.3.904-xxx.zip **说明** 软件包名称中的xxx表示时间戳。	包含了本教程中使用到的模型训练代码、推理部署代码和推理评测代码。代码包具体说明请参见**代码目录介绍**。 AscendSpeed是用于模型并行计算的框架，其中包含了许多模型的输入处理方法。	获取路径：**Support网站** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
权重和词表文件	包含了本教程使用到的HuggingFace原始权重文件和Tokenizer。 标记器(Tokenizer)是NLP管道的核心组件之一。它们有一个目的：将文本转换为模型可以处理的数据。模型只能处理数字，因此标记器(Tokenizer)需要将文本输入转换为数字数据。	**baichuan2-13b-chat** 这个路径下既有权重，也有Tokenizer，全部下载。具体内容参见**权重和词表文件介绍**。

代码目录介绍

AscendCloud-3rdLLM代码包结构介绍如下：

```
xxx-Ascend                #xxx表示版本号
  ├──llm_evaluation  #推理评测代码包
      ├──benchmark_eval   #精度评测
      ├──benchmark_tools  #性能评测
  ├──llm_train       #模型训练代码包
      ├──AscendSpeed  #基于AscendSpeed的训练代码
          ├──AscendSpeed  #加速库
          ├──ModelLink   #基于ModelLink的训练代码
          ├──scripts/    #训练需要的启动脚本
```

本教程需要使用到的训练相关代码存放在llm_train/AscendSpeed目录下，具体文件介绍如下：

```
├──llm_train       #模型训练代码包
      ├──AscendSpeed   #基于AscendSpeed的训练代码
          ├──AscendSpeed  #加速库
          ├──ModelLink   #基于ModelLink的训练代码，数据预处理脚本
          ├──scripts/    #训练需要的启动脚本，调用ModelLink
              ├──baichuan2   #Baichuan2的训练代码
                  ├──baichuan2.sh                #Baichuan2训练脚本
```

权重和词表文件介绍

下载完毕后的HuggingFace原始权重文件包含以下内容，此处以baichuan2-13B为例。

```
baichuan2-13B
├── config.json
├── configuration_baichuan.py
├── generation_config.json
├── generation_utils.py
├── handler.py
├── modeling_baichuan.py
├── pytorch_model-00001-of-00003.bin
```

```
├── pytorch_model-00002-of-00003.bin
├── pytorch_model-00003-of-00003.bin
├── pytorch_model.bin.index.json
├── quantizer.py
├── README.md
├── special_tokens_map.json
├── tokenization_baichuan.py
├── tokenizer_config.json
├── tokenizer.model
├── transform.ckpt
├── transformed.ckpt
```

工作目录结构如下

```
${workdir}（例如/home/ma-user/ws ）
├──llm_train
   ├── AscendSpeed              #代码目录
      ├── AscendSpeed          #训练依赖的三方模型库
      ├── ModelLink           #AscendSpeed代码目录
      ├── scripts/            #训练启动脚本
# 以下目录结构自己创建
├── processed_for_ma_input
    ├── BaiChuan2-13B
       ├── data               #预处理后数据
          ├── pretrain   #预训练加载的数据
          ├── finetune   #微调加载的数据
       ├──converted_weights  #HuggingFace格式转换magatron格式后权重文件
├── saved_dir_for_ma_output         #训练输出保存权重，根据实际训练需求设置
    ├── BaiChuan2-13B
       ├── logs              #训练过程中日志（loss、吞吐性能）
       ├── lora              #lora微调输出权重
       ├── sft              #增量训练输出权重
       ├── pretrain          #预训练输出权重
├── tokenizers                    #原始权重及tokenizer目录
    ├── BaiChuan2-13B
├── training_data                 #原始数据目录
    ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预原始数据文件
```

上传代码到工作环境

1. 使用root用户以SSH的方式登录DevServer。
2. 将AscendSpeed代码包AscendCloud-3rdLLM-xxx-xxx.zip上传到${workdir}目录下并解压缩，如：/home/ma-user/ws目录下，以下都以/home/ma-user/ws为例。
   ```
   unzip AscendCloud-3rdLLM-xxx-xxx.zip  #解压缩，-xxx-xxx表示软件包版本号和时间戳
   ```
3. 上传tokenizers文件到工作目录中的/home/ma-user/ws/tokenizers/BaiChuan2-13B目录。

   具体步骤如下：

   进入到${workdir}目录下，如：/home/ma-user/ws。

   ```
   cd /home/ma-user/ws
   mkdir -p tokenizers/BaiChuan2-13B
   ```

   将 权重和词表文件 文件放置此处。
4. 修改tokenizer目录下tokenization_baichuan.py中约71行内容。

   调整 super().__init__( ) 位置：将super().__init__( ) 放置def __init__( ) 方法最底层，如下图所示。

图 3-266 修改 tokenization_baichuan.py

```
        pad_token = (
            AddedToken(pad_token, lstrip=False, rstrip=False)
            if isinstance(pad_token, str)
            else pad_token
        )
#        super().__init__(
#            bos_token=bos_token,
#            eos_token=eos_token,
#            unk_token=unk_token,
#            pad_token=pad_token,
#            add_bos_token=add_bos_token,
#            add_eos_token=add_eos_token,
#            sp_model_kwargs=self.sp_model_kwargs,
#            clean_up_tokenization_spaces=clean_up_tokenization_spaces,
#            **kwargs,
#        )
        self.vocab_file = vocab_file
        self.add_bos_token = add_bos_token
        self.add_eos_token = add_eos_token
        self.sp_model = spm.SentencePieceProcessor(**self.sp_model_kwargs)
        self.sp_model.Load(vocab_file)
        super().__init__(
            bos_token=bos_token,
            eos_token=eos_token,
            unk_token=unk_token,
            pad_token=pad_token,
            add_bos_token=add_bos_token,
            add_eos_token=add_eos_token,
            sp_model_kwargs=self.sp_model_kwargs,
            clean_up_tokenization_spaces=clean_up_tokenization_spaces,
            **kwargs,
        )
```

3.19.2.3 准备数据

本教程使用到的训练数据集是Alpaca数据集。您也可以自行准备数据集。

Alpaca 数据

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

- 训练数据集下载：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**，数据大小：24M左右。

自定义数据

用户也可以自行准备训练数据。数据要求如下：

使用标准的.json格式的数据，通过设置--json-key来指定需要参与训练的列。

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

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

经下载的原始数据存放在/home/ma-user/ws/training_data目录下。具体步骤如下：

1. 进入到/home/ma-user/ws/目录下。
2. 创建目录“training_data”，并将原始数据放置在此处。
   ```
   mkdir  training_data
   ```

数据存放参考目录结构如下：

```
${workdir}（例如/home/ma-user/ws）
  ├── training_data                    #原始数据目录
      ├── train-00000-of-00001-a09b74b3ef9c3b56.parquet  #预训练原始数据文件
```

3.19.2.4 准备镜像

准备训练Baichuan2-13B模型适用的容器镜像，包括获取镜像地址，了解镜像中包含的各类固件版本，配置DevServer物理机环境操作。

镜像地址

本教程中用到的基础镜像地址和配套版本关系如下表所示，请提前了解。

表 3-156 基础容器镜像地址

镜像用途	镜像地址
基础镜像（训练和推理通用）	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42

表 3-157 模型镜像版本

模型	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

Step1 检查环境

1. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
2. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```
3. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```
4. 执行如下命令统一文件属组。启动容器时默认用户为ma-user用户，使用其他属组如root用户上传的数据和文件等，可能会存在权限不足的问题，因此需要执行如下命令统一文件属主。

```
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
例如：
sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

Step2 获取训练镜像

建议使用官方提供的镜像部署训练服务。镜像地址{image_url}参见**镜像地址**。

```
docker pull {image_url}
```

Step3 启动容器镜像

启动容器镜像前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。启动容器命令如下。

```
container_work_dir="/home/ma-user/ws"   # 容器内挂载的目录
work_dir="/home/ma-user/ws"                 # 宿主机挂载目录，存放了代码、数据、权重
container_name="ascendspeed"            # 启动的容器名称
image_name="${container_name} "         # 启动的镜像ID
docker run -itd \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   --cpus 192 \
   --memory 1000g \
   --shm-size 32g \
   --net=host \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   $image_name \
   /bin/bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如ascendspeed。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
- ${image_name} 为docker镜像的ID，在宿主机上可通过docker images查询得到。

通过容器名称进入容器中。

```
docker exec -it ${container_name} bash
```

安装依赖包。

```
#进入scriptsscripts目录
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/scripts
#执行安装命令
pip install -r requirements.txt
```

3.19.3 预训练

3.19.3.1 预训练数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

Alpaca 数据处理说明

数据预处理脚本preprocess_data.py存放在代码包的“llm_train/AscendSpeed/ModelLink/tools/”目录中，脚本具体内容如下。

```
#数据预处理
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path {work_dir}/tokenizers/BaiChuan2-13B \
 --output-prefix {work_dir}/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --seq-length 4096 \
 --tokenizer-type PretrainedFromHF
```

参数说明：

- ${work_dir}的路径指容器工作路径：如/home/ma-user/ws/ 。
- - input：原始数据集的存放路径
- - output-prefix：处理后的数据集保存路径+数据集名称前缀（ 例如: alpaca ）
- - tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'，'PretrainedFromHF']，一般为PretrainedFromHF。
- - tokenizer-name-or-path：tokenizer的存放路径
- -workers：设置数据处理使用执行卡数量
- -log-interval：是一个用于设置日志输出间隔的参数，表示输出日志的频率。在训练大规模模型时，可以通过设置这个参数来控制日志的输出
- seq-length：是一个用于计算序列长度的函数。它接收一个序列作为输入，并返回序列的长度，需和训练时参数保持一致。

数据预处理后输出的训练数据如下：

- alpaca_text_document.bin
- alpaca_text_document.idx

Alpaca 数据处理具体操作

Alpaca数据处理具体操作步骤如下：

1. 创建数据处理后的输出目录/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/。

```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/BaiChuan2-13B/data/pretrain
```

2. 将获取到的Alpaca预训练数据集传到上一步创建的目录中。如还未下载数据集，请参考**准备数据**获取。
3. 进入“/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/”目录，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。

```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令：
python ./tools/preprocess_data.py \
 --input /home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
 --tokenizer-name-or-path /home/ma-user/ws/tokenizers/BaiChuan2-13B \
 --output-prefix /home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca \
 --workers 8 \
 --log-interval 1000 \
 --seq-length 4096 \
 --tokenizer-type PretrainedFromHF
```

4. 数据处理完后，在/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/目录下生成alpaca_text_document.bin和alpaca_text_document.idx文件。

自定义数据

如果是用户自己准备的数据集，可以使用Ascendspeed代码仓中的转换工具将json格式数据集转换为训练中使用的.idx + .bin格式。

```
#示例：
#1.将准备好的json格式数据集存放于/home/ma-user/ws/training_data目录下: data.json
#2.运行转换脚本
#进入到ModelLink目录下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#执行以下命令：
python ./tools/preprocess_data.py \
 --input {work_dir}/training_data/data.json \
 --tokenizer-name-or-path {work_dir}/tokenizers/BaiChuan2-13B \
 --output-prefix {work_dir}/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca \
 --workers 8 \
 --seq-length 4096 \
 --log-interval 1000 \
 --tokenizer-type PretrainedFromHF
#3.执行完成后在 datasets文件夹中可以得到 data_text_document.idx 与data_text_document.bin 两个文件
```

3.19.3.2 预训练超参配置

本章节介绍预训练前的超参配置，可以根据实际需要修改。

预训练脚本baichuan2.sh，存放在“6.3.904-Ascend/llm_train/AscendSpeed/scripts/baichuan2”目录下。训练前，可以根据实际需要修改超参配置。

表 3-158 超参配置

参数	值	参数说明
DATA_PATH	/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca_text_document	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_MODEL	/home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model	必填。加载tokenizer时，tokenizer存放地址。
MODEL_TYPE	13B	必填。模型加载类型，默认为13B。
TRAIN_ITERS	200	非必填。训练迭代周期。根据实际需要修改。默认值为1000
MBS	1	非必填。流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch 默认值1。建议值单机1，双机2。
GBS	16	非必填。默认值 16 训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长，建议值单机16，双机32。
TP	8	非必填。张量并行。默认值为8
PP	1	非必填。默认值为1 流水线并行。建议值单机1，双机2。
RUN_TYPE	pretrain	必填。表示训练类型，根据实际训练任务类型选择。取值说明： ● pretrain：表示预训练 ● retrain：表示断点续训 ● sft：表示SFT微调训练 ● lora：表示LoRA微调训练
MASTER_ADDR	localhost	多机必填。主节点IP地址，多台机器中指定一个节点ip为主节点ip，一般指定第一个节点ip为主节点IP。
NNODES	1	多机必填。节点总数，如为双机，则写2。
NODE_RANK	0	多机必填。在节点序号，当前节点id，一般从0开始。

参数	值	参数说明
WORK_DIR	/home/ma-user/ws	容器的工作目录。训练的权重文件保存在此路径下。非必填，默认值为：/home/ma-user/ws。

3.19.3.3 预训练任务

启动训练脚本

单机启动

以baichuan2-13b为例，单机训练启动样例命令如下，以自己实际为准。在/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/代码目录下执行。超参详解参考**表3-158**。

```
MODEL_TYPE=13B  RUN_TYPE=pretrain DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca_text_document TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model TRAIN_ITERS=200 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
```

以上超参配置中，其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL为必填；TRAIN_ITERS、MBS、GBS、 TP、PP 、WORK_DIR为非必填，有默认值。

多机启动

以baichuan2-13b为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。超参详解参考**表3-158**。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=13B  RUN_TYPE=pretrain DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca_text_document TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model  TRAIN_ITERS=200 MBS=2  GBS=32 TP=8 PP=2 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
...
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=13B  RUN_TYPE=pretrain DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/alpaca_text_document TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model  TRAIN_ITERS=200 MBS=2  GBS=32 TP=8 PP=2 sh scripts/baichuan2/baichuan2.sh
```

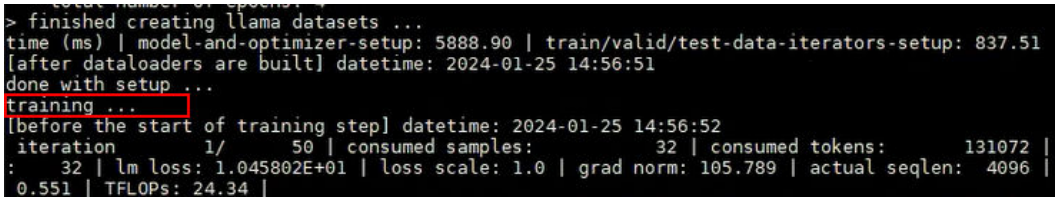
以上命令多台机器执行时，只有${NODE_RANK}：节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATASET_PATHDATA_PATH、TOKENIZER_PATHTOKENIZER_MODEL为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

等待模型载入

执行训练启动命令后，等待模型载入，当出现“training”关键字时，表示开始训练。训练过程中，训练日志会在最后的Rank节点打印。

图 3-267 等待模型载入



更多查看训练日志和性能操作，请参考**查看日志和性能**章节。

如果需要使用断点续训练能力，请参考**断点续训练**章节修改训练脚本。

3.19.3.4 断点续训练

断点续训练是指因为某些原因导致训练作业还未完成就被中断，下一次训练可以在上一次的训练基础上继续进行。这种方式对于需要长时间训练的模型而言比较友好。

断点续训练是通过checkpoint机制实现。checkpoint机制是在模型训练的过程中，不断地保存训练结果（包括但不限于EPOCH、模型权重、优化器状态、调度器状态）。即便模型训练中断，也可以基于checkpoint接续训练。

当需要从训练中断的位置接续训练，只需要加载checkpoint，并用checkpoint信息初始化训练状态即可。用户需要在代码里加上reload ckpt的代码，使能读取前一次训练保存的预训练模型。

原有训练参数配置**表3-158断点续训练**中新加**MODEL_PATH**参数，并修改**TRAIN_ITERS**参数值。

表 3-159 断点续训练修改参数

参数	参考值	参数说明
CKPT_LOAD_DIR	/home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/pretrain	加载上一步预训练后保存的权重文件。
TRAIN_ITERS	300	训练周期，必须大于上次保存训练的周期次数。
RUN_TYPE	retrain	必填。训练脚本类型，retrain表示断点续训练。

断点续训练操作过程

baichuan2-13b的断点续训脚本baichuan2.sh，存放在“6.3.904-Ascend/llm_train/AscendSpeed/scripts/baichuan2”目录下。

1. 执行命令如下，进入AscendSpeed代码目录。
   ```
   cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/
   ```
2. 在AscendSpeed代码目录下执行断点续训练脚本。

   单机启动
   ```
   MODEL_TYPE=13B  RUN_TYPE=retrain DATA_PATH=
   /home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/retrain/alpaca_text_document
   TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model
   ```

```
CKPT_LOAD_DIR=/home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/pretrain
TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 sh scripts/baichuan2/baichuan2.sh
```

多机启动

以baichuan2-13b为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行,已双机为例。

```
#第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=13B  RUN_TYPE=retrain
DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/
alpaca_text_document TOKENIZER_MODEL=/home/ma-user/code/model/Baichuan2-13B-Chat/
tokenizer.model CKPT_LOAD_DIR=/home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/
pretrain TRAIN_ITERS=300 MBS=2  GBS=32 TP=8 PP=2 sh scripts/baichuan2/baichuan2.sh
...
...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=13B  RUN_TYPE=retrain
DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/pretrain/
alpaca_text_document TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B/
tokenizer.model CKPT_LOAD_DIR=/home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/
pretrain TRAIN_ITERS=300 MBS=2  GBS=32 TP=8 PP=2 sh scripts/baichuan2/baichuan2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}：节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、CKPT_LOAD_DIR为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

图 3-268 保存的 ckpt

```
[root@devserver-modelarts ckpt]# ll
total 24
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 12:34 iter_0000005
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:04 iter_0000010
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 13:34 iter_0000015
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:04 iter_0000020
drwxr-x--- 42 HwHiAiUser users 4096 Oct 20 14:56 iter_0000025
-rw-r-----  1 HwHiAiUser users    2 Oct 20 14:20 latest_checkpointed_iteration.txt
```

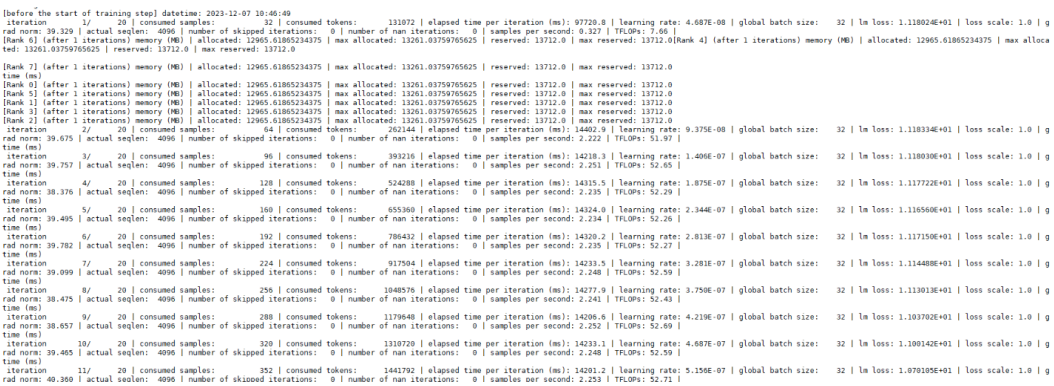
3. 可以参考**查看日志和性能**操作，查看断点续训练日志和性能。

3.19.3.5 查看日志和性能

查看日志

训练过程中，训练日志会在最后的Rank节点打印。

图 3-269 打印训练日志



训练完成后，如果需要单独获取训练日志文件，可以在${SAVE_PATH}/logs路径下获取。日志存放路径为{work_dir}/saved_dir_for_ma_output/BaiChuan2-13B/logs,本实例日志路径为/home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/logs。

查看性能

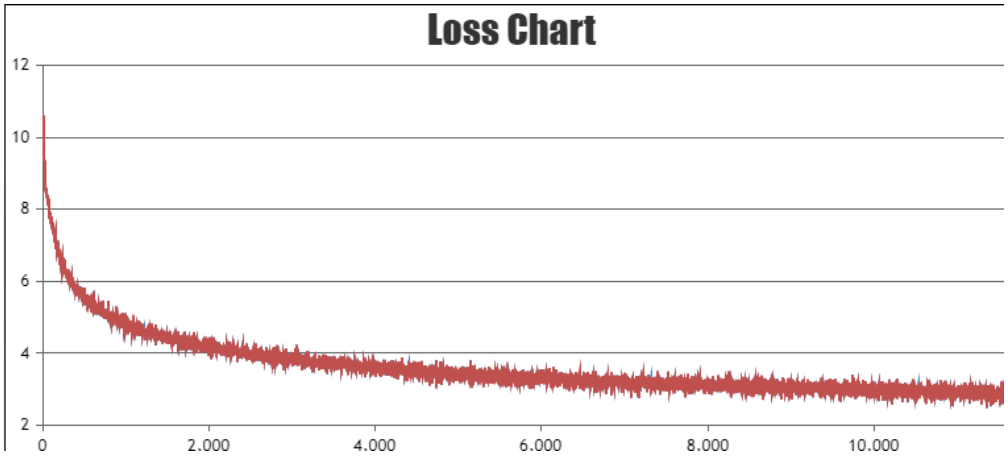
训练性能主要通过训练日志中的2个指标查看，吞吐量和loss收敛情况。

- 吞吐量（tokens/s/p）：global batch size*seq_length/（总卡数*elapsed time per iteration）*1000，其参数在日志里可找到，默认seq_len值为4096，默认global batch size为64；其global batch size（GBS）、seq_len（SEQ_LEN）为训练时设置的参数。
- loss收敛情况：日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。也可以使用可视化工具**TrainingLogParser**查看loss收敛情况，如**图3-270**所示。

  单节点训练：训练过程中的loss直接打印在窗口上。

  多节点训练：训练过程中的loss打印在最后一个节点上。

  图 3-270 Loss 收敛情况



3.19.4 SFT 全参微调

3.19.4.1 SFT 全参微调数据处理

SFT全参微调（Supervised Fine-Tuning）前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

下载数据

SFT全参微调涉及的数据下载地址：**https://huggingface.co/datasets/tatsu-lab/alpaca/resolve/main/data/train-00000-of-00001-a09b74b3ef9c3b56.parquet**

如果在**准备数据**章节已下载数据集，此处无需重复操作。

SFT全参微调和LoRA微调训练使用的是同一个数据集，数据处理一次即可，训练时可以共用。

数据预处理说明

使用数据预处理脚本preprocess_data.py脚本重新生成.bin和.idx格式的SFT全参微调数据。preprocess_data.py存放在6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/tools目录中，脚本具体内容如下。

```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令：
python ./tools/preprocess_data.py \
  --input /home/ma-user/code/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
  --tokenizer-name-or-path $TOKENIZER_PATH \
  --output-prefix $DATA_PATH \
  --workers 8 \
  --log-interval 1000 \
  --tokenizer-type PretrainedFromHF \
  --handler-name GeneralInstructionHandler \
  --seq-length 4096 \
  --append-eod
```

参数说明：

- input：用于微调的原始数据。

- output-prefix：处理后的数据集保存路径+数据集名称前缀（例如：alpaca-ft）。

- tokenizer-type：tokenizer的类型，可选项有['BertWordPieceLowerCase'，'BertWordPieceCase'，'GPT2BPETokenizer'， 'PretrainedFromHF']，设置为PretrainedFromHF。

- tokenizer-name-or-path：tokenizer的存放路径。

- handler-name：生成数据集的用途，这里是生成的指令数据集，用于微调。

- append-eod:参数用于控制是否在每个输入序列的末尾添加一个特殊的标记。这个标记表示输入序列的结束,可以帮助模型更好地理解和处理长序列

- workers 需要使用的卡数

- seq-length：是一个用于计算序列长度的函数。它接收一个序列作为输入，并返回序列的长度，需和训练时参数保持一致。

输出结果

alpaca_ft_packed_attention_mask_document.bin

alpaca_ft_packed_attention_mask_document.idx

alpaca_ft_packed_input_ids_document.bin

alpaca_ft_packed_input_ids_document.idx

alpaca_ft_packed_labels_document.bin

alpaca_ft_packed_labels_document.idx

数据处理具体操作

SFT全参微调数据处理具体操作步骤如下。

1. 创建处理后的数据存放目录/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/。
```
cd /home/ma-user/ws/  #进入容器工作目录
mkdir -p processed_for_ma_input/BaiChuan2-13B/data/finetune
```
2. 进入代码目录“/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/”，在代码目录中执行preprocess_data.py脚本处理数据。

   此处提供一段实际的数据处理代码示例如下。
```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink/
#执行以下命令：
python ./tools/preprocess_data.py \
  --input /home/ma-user/ws/training_data/train-00000-of-00001-a09b74b3ef9c3b56.parquet \
  --tokenizer-name-or-path /home/ma-user/ws/tokenizers/BaiChuan2-13B \
  --output-prefix /home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft \
  --workers 8 \
  --log-interval 1000 \
  --tokenizer-type PretrainedFromHF \
  --handler-name GeneralInstructionHandler \
  --seq-length 4096 \
  --append-eod
```
   数据处理完后，在 /home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/目录下生成转换后的数据文件。

3.19.4.2 SFT 全参微调权重转换

支持HuggingFace格式权重转换为Megatron格式后再进行SFT全参微调。本章节主要介绍如何将HuggingFace权重转换为Megatron格式。此处的HuggingFace权重文件和转换操作结果同时适用于SFT全参微调和LoRA微调训练。

HuggingFace 权重转换操作

1. 下载baichuan2-13b的预训练权重和词表文件，并上传到/home/ma-user/ws/tokenizers/baichuan2-13b-hf目录下。具体下载地址请参见**表3-155**。如果已下载，忽略此步骤。
2. 创建权重转换后的输出目录/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights/。
```
cd /home/ma-user/ws/  #进入/home/ma-user/ws/目录
mkdir -p processed_for_ma_input/BaiChuan2-13B/converted_weights
```
3. 进入代码目录/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink，在代码目录中执行util.py脚本。
```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#进入到ModelLink目录下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
# 权重格式转换
python tools/checkpoint/util.py --model-type GPT \
    --loader llama2_hf \
    --saver megatron \
    --target-tensor-parallel-size 8 \   #与微调TP值保持一致
    --target-pipeline-parallel-size 1 \ #与微调PP值保持一致
    --load-dir /home/ma-user/ws/tokenizers/BaiChuan2-13B \
    --save-dir /home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights \
```

```
    --tokenizer-model /home/ma-user/ws/tokenizers/BaiChuan2-13B/tokenizer.model
    --w-pack True
```

参数说明：

- -target-tensor-parallel-size：与后续微调TP值保持一致
- -target-pipeline-parallel-size：与后续微调PP值保持一致
- -load-dir：原始HuggingFace权重
- -tokenizer-model:tokenizer路径
- -save-dir:从 huggingface 格式转化为 magatron 格式输出路径
- -w-pack ： True

4. 权重转换完成后，在/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights目录下查看转换后的权重文件。

3.19.4.3 SFT 全参微调超参配置

本章节介绍SFT全参微调前的超参配置，可以根据实际需要修改。

SFT全参微调脚本baichuan2.sh，存放在6.3.904-Ascend/llm_train/AscendSpeed/scripts/baichuan2目录下。可以根据实际需要修改超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型不同，以及输入输出路径的配置的不同。

表 3-160 SFT 全参微调超参配置

参数	值	参数说明
DATA_PATH	/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_MODEL	/home/ma-user/ws/tokenizers/BaiChuan2-13B/	必填。加载tokenizer时，tokenizer存放地址。
MODEL_TYPE	13B	必填。模型加载类型，默认为13B。
TRAIN_ITERS	2000	非必填。训练迭代周期。根据实际需要修改。
MBS	1	非必填。流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 默认值1。建议值单机1，双机32。
GBS	16	非必填。训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长，建议值单机16，双机32。
TP	8	非必填。张量并行。默认值为8。

参数	值	参数说明
PP	1	非必填。默认值为1 流水线并行。建议值单机1，双机2。
RUN_TYPE	sft	必填。表示训练类型。sft表示SFT微调。
MASTER_ADDR	localhost	多机必填。主节点IP地址，多台机器中指定一个节点ip为主节点ip，一般指定第一个节点ip为主节点IP。
NNODES	1	多机必填。节点总数，如为双机，则写2。
NODE_RANK	0	多机必填。在节点序号，当前节点id，一般从0开始。
CKPT_LOAD_DIR	/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights	从 huggingface 格式转化为 magatron 格式的权重文件。
WORK_DIR	/home/ma-user/ws	容器的工作目录。训练的权重文件保存在此路径下。非必填，默认值为：/home/ma-user/ws

3.19.4.4 SFT 全参微调任务

前提条件

- SFT全参微调使用的数据集为alpaca_data数据，已经完成数据处理，具体参见**SFT全参微调数据处理**。
- 原始的HuggingFace权重，已将原始的HuggingFace权重转换为Megatron格式，具体参见**SFT全参微调权重转换**

Step2 启动训练脚本

单机启动

以baichuan2-13b为例，单机SFT全参微调启动命令如下。进入代码目录/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed下执行启动脚本，超参详解参考**表1 增量预训练超参配置**

```
MODEL_TYPE=13B  RUN_TYPE=sft DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights  TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL为必填；TRAIN_ITERS、MBS、GBS、 TP、PP WORK_DIR为非必填，有默认值。

多机启动

以baichuan2-13b为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。进入代码目录/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed下执行启动脚本，超参详解参考**表1 增量预训练超参配置**

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=13B  RUN_TYPE=sft DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
 ...
 ...
# 第二台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=13B  RUN_TYPE=sft DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}：节点ID值不同，其他参数都保持一致。

其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、CKPT_LOAD_DIR为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

可以参考**查看日志和性能**操作，查看训练日志。

训练完成后，请参考**查看日志和性能**章节查看性能。

3.19.4.5 查看性能

查看SFT全参微调的日志和性能，具体方法请参见**查看日志和性能**。

3.19.5 LoRA 微调训练

本章节以Baichuan2-13B为例，介绍LoRA微调训练的全过程。

Step1 LoRA 微调数据处理

训练前需要对数据集进行预处理，转化为.bin和.idx格式文件，以满足训练要求。

LoRA微调训练与SFT微调使用同一个数据集，如果已经在SFT微调时处理过数据，可以直接使用，无需重复处理。如果未处理过数据，请参见**SFT全参微调数据处理**章节先处理数据。

Step2 LoRA 微调权重转换

LoRA微调训练前，需要先把训练权重文件转换为Megatron格式。

LoRA微调训练和SFT全参微调使用的是同一个HuggingFace权重文件转换为Megatron格式后的结果也是通用的。

如果在SFT微调任务中已经完成了HuggingFace权重转换操作，如果在SFT全参微调任务中已经完成了**HuggingFace权重转换操作**，此处无需重复操作，可以直接使用SFT全参微调中的权重转换结果。如果前面没有执行HuggingFace权重转换任务，可以参考**SFT全参微调权重转换**章节完成。

Step3 LoRA 微调超参配置

本章节介绍LoRA微调训练前的超参配置，可以根据实际需要修改。

LoRA微调训练脚本baichuan2.sh，存放在llm_train/AscendSpeed/script/baichuan2/目录下。训练前，可以根据实际需要配置超参配置。

微调任务配置，操作同预训练配置类似，不同点为RUN_TYPE类型和输入输出路径，微调还需要加载权重文件。

表 3-161 LoRA 微调超参配置

参数	示例值	参数说明
DATA_PATH	/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft	必填。训练时指定的输入数据路径。一般为数据地址/处理后的数据前缀名，不加文件类型后缀。 请根据实际规划修改。
TOKENIZER_MODEL	/home/ma-user/ws/tokenizers/BaiChuan2-13B/	必填。加载tokenizer时，tokenizer存放地址。 请根据实际规划修改。
MODEL_TYPE	13B	必填。模型加载类型，默认为13B。
TRAIN_ITERS	1000	非必填。训练迭代周期。根据实际需要修改。默认值为1000。
MBS	1	非必填。流水线并行中一个micro batch所处理的样本量。在流水线并行中，为了减少气泡时间，会将一个step的数据切分成多个micro batch。 该值与TP和PP以及模型大小相关，可根据实际情况进行调整。 默认值1。建议值单机1，双机2。
GBS	16	非必填。默认值：16；训练中所有机器一个step所处理的样本量。影响每一次训练迭代的时长，建议值单机16，双机32。
TP	8	非必填。张量并行。默认值为8。
PP	1	非必填。表示流水线并行。建议值单机1，双机2。
RUN_TYPE	lora	必填。表示训练类型。lora表示LoRA微调。

参数	示例值	参数说明
MASTER_ADDR	localhost	多机必填。 单机忽略；指定主节点IP地址，多台机器中需要指定一个节点IP为主节点IP。 一般指定第一个节点IP为主节点IP。
NNODES	1	多机必填，单机忽略；，单机写1，双机写2。
NODE_RANK	0	多机必填，单机忽略；节点序号，当前节点ID，一般从0开始，单机默认是0。
CKPT_LOAD_DIR	/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights	从 huggingface 格式转化为 magatron 格式的权重文件。
WORK_DIR	/home/ma-user/ws	非必填。容器的工作目录。训练的权重文件保存在此路径下。默认值为：/home/ma-user/ws。

Step4 启动训练脚本

请根据表3-161修改超参值后，再启动训练脚本。

单机启动

以baichuan2-13b为例，单机LoRA微调启动命令如下。进入代码目录/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed下执行启动脚本。

```
MODEL_TYPE=13B  RUN_TYPE=lora DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights  TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
```

其中 MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、CKPT_LOAD_DIR为必填；TRAIN_ITERS、MBS、GBS、TP、PP 为非必填，有默认值

多机启动

以baichuan2-13b为例，多台机器执行训练启动命令如下。多机启动需要在每个节点上执行，以双机为例。进入代码目录/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed下执行启动脚本。

```
第一台节点
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=0 MODEL_TYPE=13B  RUN_TYPE=lora DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
 ...
 ...
# 第二台节点
```

```
MASTER_ADDR=xx.xx.xx.xx NNODES=2 NODE_RANK=1 MODEL_TYPE=13B  RUN_TYPE=lora DATA_PATH=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/data/finetune/alpaca_ft TOKENIZER_MODEL=/home/ma-user/ws/tokenizers/BaiChuan2-13B CKPT_LOAD_DIR=/home/ma-user/ws/processed_for_ma_input/BaiChuan2-13B/converted_weights TRAIN_ITERS=300 MBS=1  GBS=16 TP=8 PP=1 WORK_DIR=/home/ma-user/ws sh scripts/baichuan2/baichuan2.sh
```

以上命令多台机器执行时，只有${NODE_RANK}：节点ID值不同，其他参数都保持一致；其中MASTER_ADDR、NODE_RANK、MODEL_TYPE 、RUN_TYPE、DATA_PATH、TOKENIZER_MODEL、CKPT_LOAD_DIR为必填；TRAIN_ITERS、MBS、GBS、TP、PP、WORK_DIR为非必填，有默认值。

训练完成后，请参考**查看日志和性能**章节查看LoRA微调训练的日志和性能。

3.19.6 推理前的权重合并转换

模型训练完成后，训练的产物包括模型的权重、优化器状态、loss等信息。这些内容可用于断点续训、模型评测或推理任务等。

在进行模型评测或推理任务前，需要将训练后生成的多个权重文件合并，并转换成Huggingface格式的权重文件。

权重文件的合并转换操作都要求在训练的环境中进行，为下一步推理做准备。

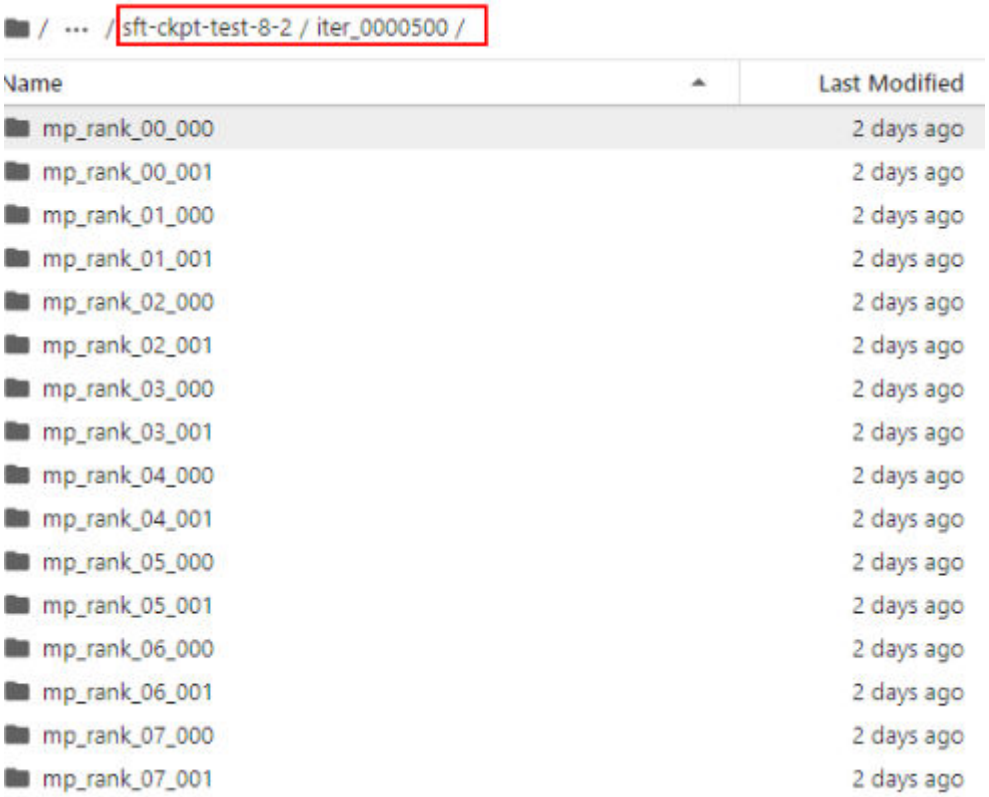
- 如果需要使用本文档中训练后的权重文件进行推理，请参考此章节合并训练权重文件并转换为Huggingface格式。
- 如果无推理任务或者使用开源Huggingface权重文件推理，都可以忽略此章节。

下一步的推理任务请参考文档《**开源大模型基于DevServer的推理通用指导**》。

将多个权重文件合并为一个文件并转换格式

任意并行切分策略的Megatron权重 格式转化为 HuggingFace权重 *（该场景一般用于将训练好的megatron模型：预训练、lora、sft 重新转回HuggingFace格式）*为下一步推理使用准备，无推理任务忽略此章节,一般训练都是多卡分布式训练权重结果文件为多个且文件为*Megatron*格式，因此需要合并多个文件转换为huggingface格式

如是多机训练转换前需将多机权重目录（iter_xxxxxxx）下mp_rank_xx_xxx文件夹整合到一起后进行转换，合并后结果如图所示：



该脚本的执行需要在/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink目录下执行。

```
#加载ascendspeed及megatron模型：
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/AscendSpeed
export PYTHONPATH=$PYTHONPATH:/home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
#进入ModelLink下：
cd /home/ma-user/ws/6.3.904-Ascend/llm_train/AscendSpeed/ModelLink
python tools/checkpoint/util.py --model-type GPT \
    --loader megatron \
    --saver megatron \
    --save-model-type save_huggingface_llama \
    --load-dir /home/ma-user/ws/saved_dir_for_ma_output/BaiChuan2-13B/lora \
    --target-tensor-parallel-size 1 \
    --target-pipeline-parallel-size 1 \
    --w-pack True \
    --save-dir /home/ma-user/ws/tokenizers/BaiChuan2-13B  # <-- 需要填入原始HF模型路径，新权重会存于../Baichuan2-13B/mg2hg下
```

参数说明：

- save-model-type：输出后权重格式
- load-dir：训练完成后保存的权重路径
- save-dir：需要填入原始HF模型路径，新权重会存于../Baichuan2-13B/mg2hg下。
- target-tensor-parallel-size：任务不同调整参数target-tensor-parallel-size。默认为1
- target-pipeline-parallel-size ： 任务不同调整参数target-pipeline-parallel-size。默认为1

4 AIGC 模型训练推理

4.1 SDXL 基于 Standard 适配 PyTorch NPU 的 LoRA 训练指导（6.3.907）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。SDXL LoRA是指在已经训练好的SDXL模型基础上，使用新的数据集进行LoRA微调。

本文档主要介绍如何在ModelArts Standard上，利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL LoRA训练。

获取软件和镜像

表 4-1 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	SWR上拉取。

表 4-2 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
驱动	23.0.6
PyTorch	2.1.0

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练作业使用单机单卡资源。
- 确保容器可以访问公网。
- 本案例仅支持在专属资源池上运行。

Step1 创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

- 硬盘空间：至少200GB。
- 昇腾资源规格：Ascend: 8*ascend-snt9b表示昇腾8卡规格。
- 推荐使用“西南-贵阳一”Region上的昇腾资源。

Step2 创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档需要将运行代码以及输入输出数据存放OBS，请提前创建OBS（参考**创建OBS桶**），例如桶名：sdxl-train。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

Step3 准备代码

在**获取软件和镜像**中，下载并解压代码包。本文档主要使用aigc_train->torch_npu->diffusers下的部分文件，请利用**OBS Browser+工具**将文件夹中内容上传至OBS的代码文件夹code中。

```
obs://<bucket_name>/code
├── diffusers-train.patch
├── prepare.sh
├── diffusers_sdxl_lora_train.sh
```

Step4 下载模型依赖包

请在如下链接中下载好模型依赖包。

- 下载stable-diffusion-xl-base-1.0，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
- 下载vae-fp16-fix，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**

Step5 下载数据集

本案例使用Huggingface提供的**naruto-blip-captions**数据集，官网下载地址：**https://huggingface.co/datasets/lambdalabs/naruto-blip-captions/tree/main**

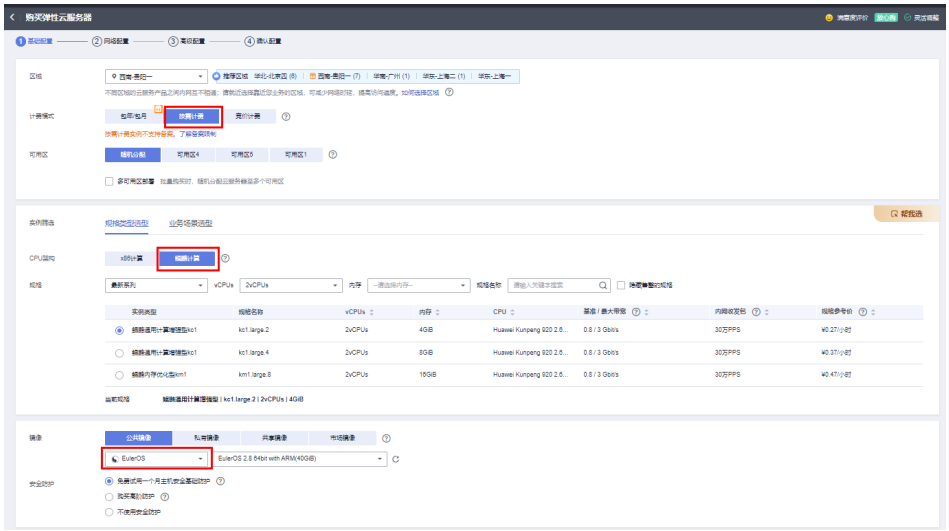
Step6 准备镜像

步骤1 创建ECS。

参考**ECS文档**购买弹性云服务器。网络配置、高级配置等后续步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，并在控制台发送后续步骤中的远程命令。

注意：创建的ECS虚拟机使用ARM镜像创建。

图 4-1 购买 ECS



步骤2 安装Docker。

1. 检查docker是否安装。
   ```
   docker -v  #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker
   ```
2. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

步骤3 构建自定义镜像。

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**获取软件和镜像**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /
home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_sdxl_lora_train.sh /home/ma-user/sdxl-train/user-job-dir/code/
diffusers_sdxl_lora_train.sh

COPY --chown=ma-user:ma-group diffusers-train.patch /home/ma-user/sdxl-train/diffusers-train.patch
COPY --chown=ma-user:ma-group prepare.sh /home/ma-user/sdxl-train/prepare.sh
RUN cd /home/ma-user/sdxl-train && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
```

把上述代码文件、模型依赖包、数据集、Dockerfile文件都上传至ECS，上传步骤可参考**本地Windows主机使用WinSCP上传文件到Linux云服务器**。

文件上传后目录如下：

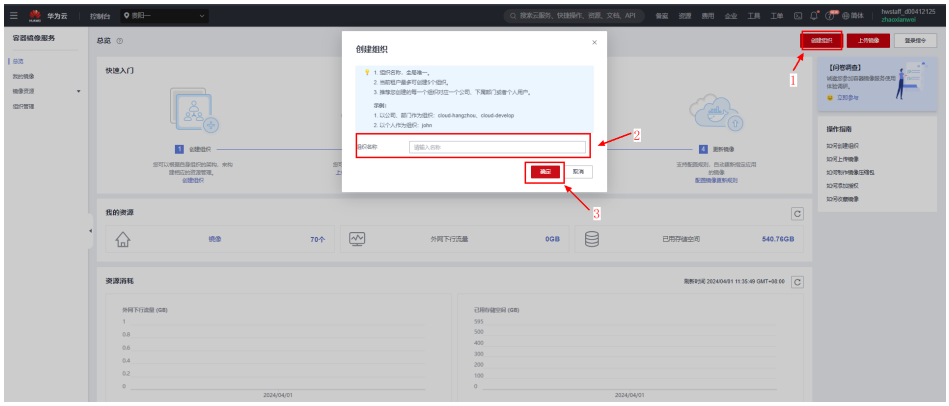
```
<ECS_folder>
├── diffusers_sdxl_lora_train.sh        #  华为侧提供的代码文件
├── diffusers-train.patch     #  华为侧提供的代码文件
├── prepare.sh                          #  华为侧提供的代码文件
├── Dockerfile                           #  Dockerfile文件
├── vae-fp16-fix                         #  模型依赖包vae-fp16-fix
├── stable-diffusion-xl-base-1.0        #  模型依赖包stable-diffusion-xl-base-1.0
├── datasets                            #  新建datasets文件夹，naruto-blip-captions数据集放在该目录下
    ├── naruto-blip-captions
```

在该目录下执行命令构建自定义镜像：

```
docker build -t sdxl-train:0.0.1 .
```

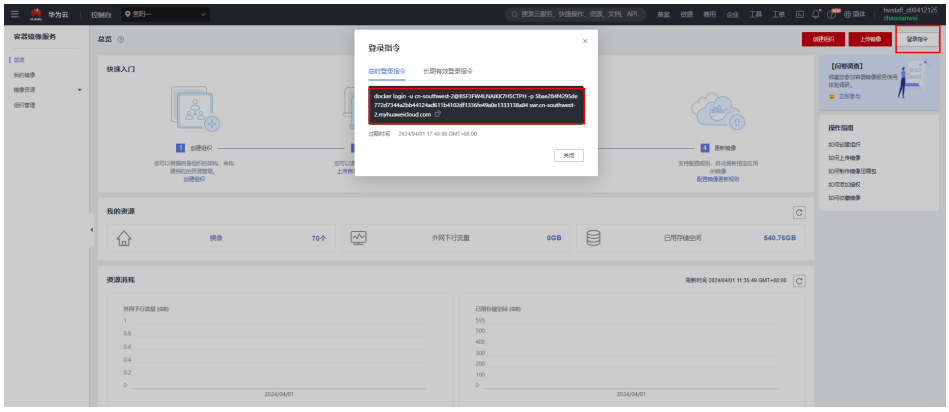
步骤4 在SWR服务页面创建镜像组织。

图 4-2 创建镜像组织



步骤5 在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中复制临时登录指令，即可完成登录。

图 4-3 复制登录指令



步骤6 修改并上传镜像。

在ECS中输入上一步的登录指令后，使用下列示例命令：

```
docker tag {image_url} swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
docker push swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

<组织名称>：步骤4中创建的组织名称。

<镜像名称>:<tag>：定义镜像名称。示例：sdxl-train:0.0.1。

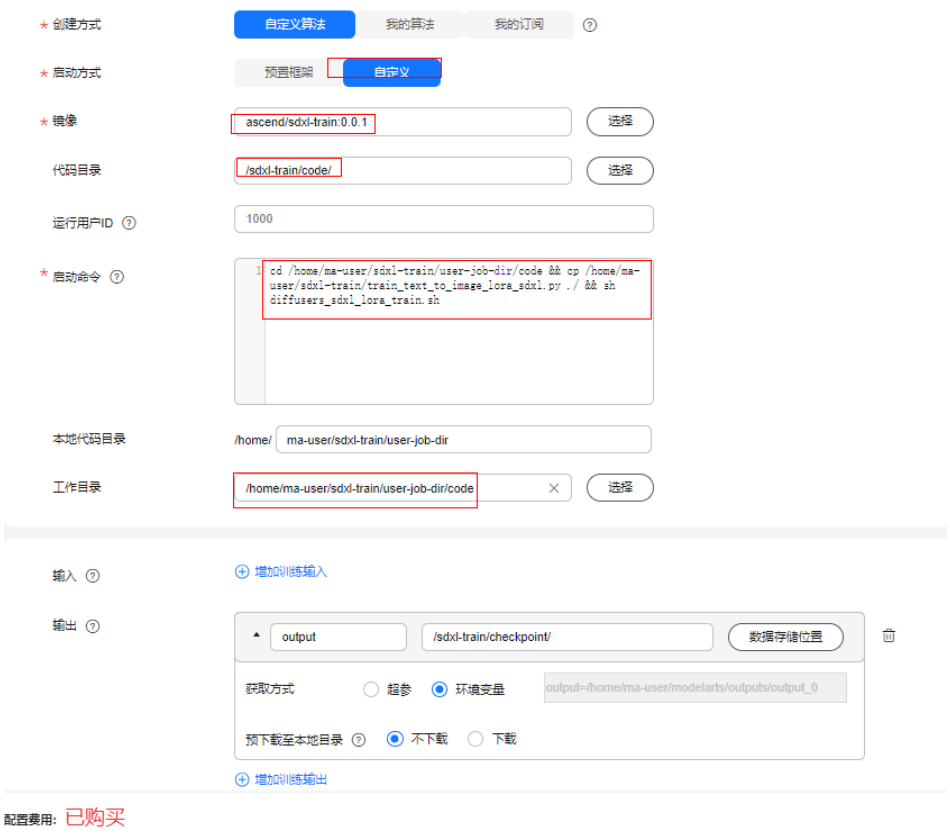
----结束

Step7 创建训练作业

创建训练作业，填下如下参数。

- 创建方式：选择自定义算法，启动方式选择自定义，然后选择上传到SWR的自定义镜像。
- 代码目录：选择上传到OBS的代码文件夹，例如/sdxl-train/code。若用户需要修改代码文件，可修改OBS桶中代码文件，创建训练作业时，会将OBS的code目录复制到训练容器的/home/ma-user/sdxl-train/user-job-dir/目录下，覆盖容器中原有的code目录。
- 启动命令：将华为侧优化后代码文件复制到工作目录后，运行启动脚本文件diffusers_sdxl_lora_train.sh。
  ```
  cd /home/ma-user/sdxl-train/user-job-dir/code && cp /home/ma-user/sdxl-train/
  train_text_to_image_lora_sdxl.py ./ && sh diffusers_sdxl_lora_train.sh
  ```
- 本地代码目录：保持默认即可。
- 工作目录：选择代码文件目录，例如/home/ma-user/sdxl-train/user-job-dir/code/。
- 输出：单击“增加训练输出”，将模型保存到OBS中。参数名称为output，数据存储位置选择OBS桶中制定文件夹，例如sdxl-train/checkpoint，获取方式选择环境变量，/home/ma-user/modelarts/outputs/output_0下的模型文件会保存到OBS中。

图 4-4 选择镜像



- 资源池：选择专属资源池，规格选择Ascend: 1*ascend-snt9b。

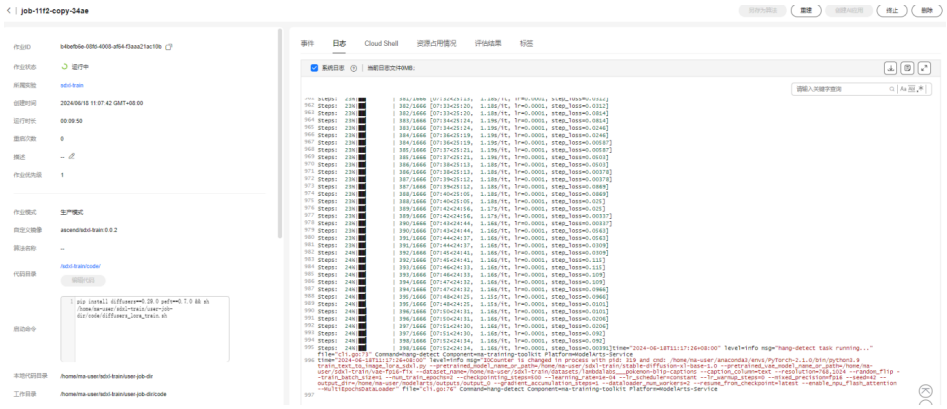
图 4-5 选择资源池规格

- 作业日志路径：选择输出日志到OBS的指定目录。

图 4-6 选择作业日志路径

填写参数完成后，提交创建训练任务，训练完成后，作业状态会显示为已完成。

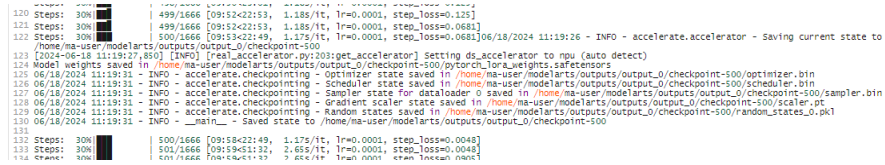
图 4-7 训练启动成功



Step8 断点续训

查看训练日志，在训练任务启动后，当训练超过500步后开始保存checkpoint文件，保存成功后，手动终止训练任务。

图 4-8 保存 checkpoint 文件

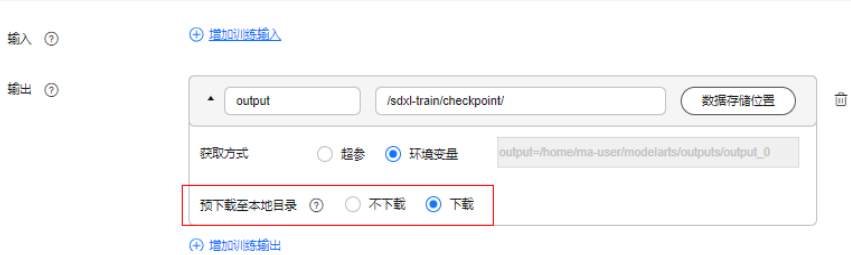


然后点击重建后提交。

图 4-9 重建训练作业

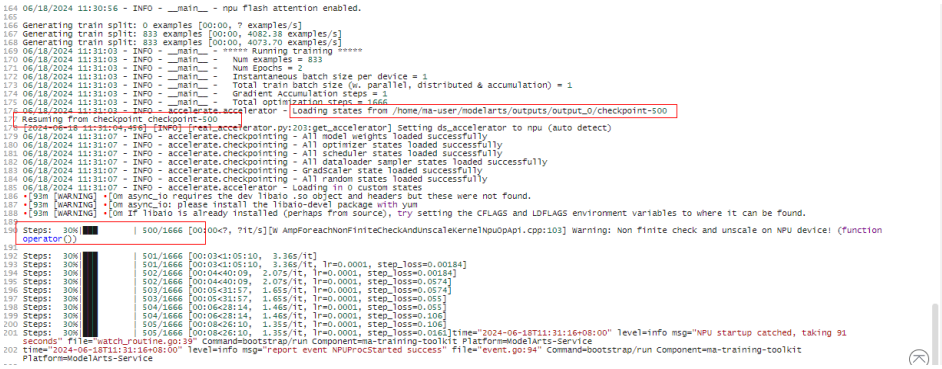
提交新的任务时，注意将预下载到本地目录勾上。

图 4-10 勾选预下载到本地目录



观察启动日志，启动会读取最新的checkpoint模型文件，接着上次保存的step位置开始训练。

图 4-11 读取最新的 checkpoint 模型文件



4.2 SD1.5&SDXL Diffusers 框架基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907）

4.2.1 训练场景和方案介绍

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。

方案概览

本方案介绍了在ModelArts Lite DevServer上使用昇腾计算资源Ascend Snt9B开展SDXL和SD1.5模型的训练过程，包括Finetune训练、LoRA训练和Controlnet训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表4-3**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Finutune训练使用单机8卡资源。
- Lora训练使用单机单卡资源。
- Controlnet训练使用单机单卡资源。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

软件配套版本

表 4-3 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-4 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

4.2.2 准备镜像环境

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载模型包、依赖代码包和数据集并上传到宿主机

1. 下载**stable-diffusion-v1-5模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/runwayml/stable-diffusion-v1-5/tree/main**
2. 下载**stable-diffusion-xl-base-1.0模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
3. 下载**vae-fp16-fix模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
4. 下载开源数据集**naruto-blip-captions**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/naruto-blip-captions/tree/main**。用户也可以使用自己的数据集。
5. 下载开源数据集**fill50k**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/fusing/fill50k/tree/main**。用户也可以使用自己的数据集。
6. 下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表4-3**。本案例使用的是解压到子目录aigc_train->torch_npu->diffusers的所有文件，将diffusers整个目录上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。
```
[root@devserver docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 diffusers
drwx------  3 root root   4096 May  7 10:50 datasets
  drwx------  3 root root   4096 May  7 10:50 naruto-blip-captions
  drwx------  3 root root   4096 May  7 10:50 fill50k
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3 构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-v1-5
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像diffusers-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-4**。
```
FROM {image_url}
```



```
COPY --chown=ma-user:ma-group diffusers /home/ma-user/diffusers
RUN cd /home/ma-user/diffusers && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-v1-5 /home/ma-user/stable-diffusion-v1-5
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
WORKDIR /home/ma-user/diffusers
```

构建自定义镜像diffusers-train:0.0.1。

```
docker build -t diffusers-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像，fintune全量微调需要启动8卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

启动容器镜像，lora微调和controlnet训练只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

参数说明：

--name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，fintune全量微调示例中挂载了8张卡davinci0~davinci7。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step5 进入容器

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```

4.2.3 Finetune 训练

本章节介绍SDXL&SD 1.5模型的Finetune训练过程。Finetune是指在已经训练好的模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能。

启动 SD1.5 Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_finetune_train.sh
```

启动 SDXL Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_sdxl_finetune_train.sh
```

说明

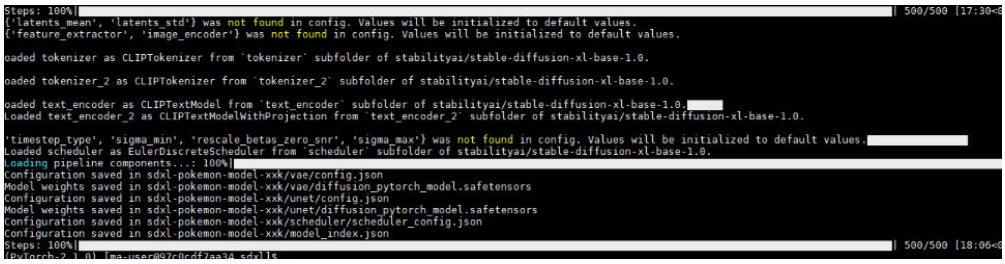
训练执行脚本中配置了保存checkpoint的频率，每500steps保存一次，如果磁盘空间较小，这个值可以改大到5000，避免磁盘空间写满，导致训练失败终止。

checkpoint保存频率的修改命令如下：

```
--checkpointing_steps=5000
```

训练执行成功如下图所示。

图 4-12 训练执行成功



4.2.4 LoRA 训练

本章节介绍SDXL&SD 1.5模型的LoRA训练过程。LoRA训练是指在已经训练好的模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。

启动 SD1.5 LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_lora_train.sh
```

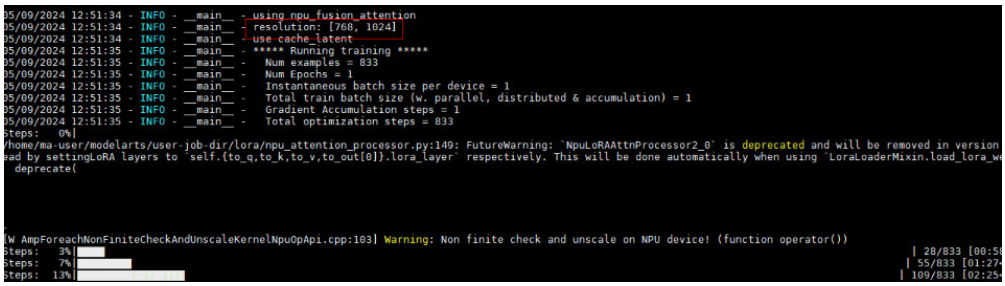
启动 SDXL LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh diffusers_sdxl_lora_train.sh
```

训练执行成功如下图所示。

图 4-13 训练执行成功



4.2.5 Controlnet 训练

使用文本提示词可以生成一副精美的画作，然而无论再怎么精细地使用提示词来指导模型，也无法描述清楚人物四肢的角度、背景中物体的位置、光线照射的角度，使用Controlnet可以通过图像特征来为扩散模型的生成过程提供更加精细控制的方式。

将Controlnet适配到昇腾卡进行训练，可以提高能效、支持更大模型和多样化部署环境，提升昇腾云在图像生成和编辑场景下的竞争力。

本章节介绍SDXL&SD 1.5模型的Controlnet训练过程。

Step1 处理 fill50k 数据集

使用ma-user用户在容器上执行如下命令解压数据集。

```
cd /home/ma-user/datasets/fill50k
unzip conditioning_images.zip
unzip images.zip
```

接着修改fill50k.py文件，如果机器无法访问huggingface网站，则需要将脚本文件中下载地址替换为容器本地目录。

```
def _split_generators(self, dl_manager):
    #metadata_path = dl_manager.download(METADATA_URL)
    #images_dir = dl_manager.download_and_extract(IMAGES_URL)
    #conditioning_images_dir = dl_manager.download_and_extract(
    #     CONDITIONING_IMAGES_URL
    #)
    metadata_path = "/home/ma-user/datasets/fill50k/train.jsonl"
    images_dir = "/home/ma-user/datasets/fill50k"
    conditioning_images_dir = "/home/ma-user/datasets/fill50k"
```

Step2 启动 SD1.5 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
cd /home/ma-user/diffusers
sh diffusers_controlnet_train.sh
```

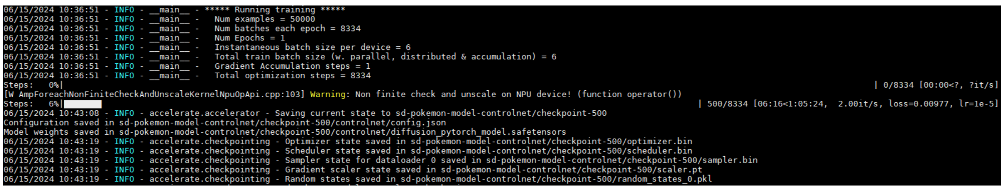
Step3 启动 sdxl 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
cd /home/ma-user/diffusers
sh diffusers_sdxl_controlnet_train.sh
```

训练执行成功如下图所示。

图 4-14 训练执行成功



4.3 SD1.5&SDXL Diffusers 框架基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907）

本文档主要介绍如何在ModelArts Lite的DevServer环境中部署Stable Diffusion模型对应SD1.5和SDXL的Diffusers框架，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署Diffusers框架用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推理部署推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

获取软件和镜像

表 4-5 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	从SWR拉取。

Step1 准备环境

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

说明

购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 检查环境。

   a. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   b. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   c. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

3. 获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参见**表4-5**。
   ```
   docker pull {image_url}
   ```

4. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。
   ```
   docker run -itd \
   --name sdxl-diffusers \
   -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
   -p 8443:8443 \
   -v /etc/localtime:/etc/localtime \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   --shm-size 60g \
   --device=/dev/davinci_manager \
   --device=/dev/hisi_hdc \
   --device=/dev/devmm_svm \
   --device=/dev/davinci3 \
   --network=bridge \
   ${image_name} bash
   ```
   参数说明：
   - --name ${conta iner_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如sdxl-diffusers。
   - --device=/dev/davinci3：挂载主机的/dev/davinci3到容器的/dev/davinci3。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
   - ${image_name} 代表 ${image_name}。

5. 进入容器。需要将${container_name}替换为实际的容器名称，例如：sdxl-diffusers。
   ```
   docker exec -it ${container_name} bash
   ```

Step2 安装依赖和模型包

1. 安装Diffusers相关依赖。
   ```
   pip install -i https://pypi.tuna.tsinghua.edu.cn/simple diffusers bottle invisible_watermark transformers accelerate safetensors
   ```

2. 获取SDXL模型包并解压到/home/ma-user目录下。提供2种模型包下载方式。
   - 模型包直接下载（如果不能访问HuggingFace官网，推荐此方式）

     下载到容器/home/ma-user目录下后，解压。

     ```
     cd  /home/ma-user/
     wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/stable-diffusion-xl-model.tar.gz
     tar -zxvf stable-diffusion-xl-model.tar.gz
     rm -rf stable-diffusion-xl-model.tar.gz
     ```

   - 也可以从HuggingFace官网下载到本地后，通过docker cp命令复制到容器中/home/ma-user目录下，如下图所示。

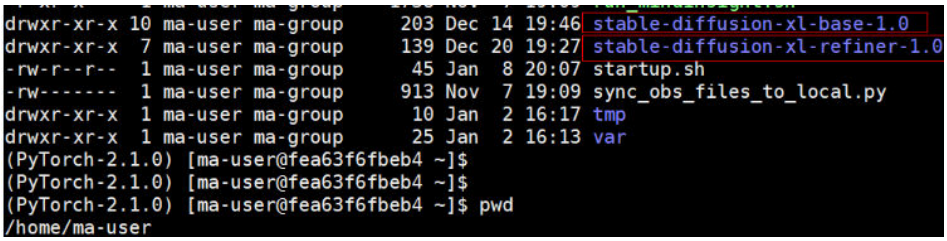
     在线下载地址：

     https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main

     https://huggingface.co/stabilityai/stable-diffusion-xl-refiner-1.0/tree/main

     由于本实例采用的都是FP16的模型，相应模型建议都只下载FP16的，节约下载和传送时间。

     图 4-15 下载 SDXL 模型包并解压

3. 获取controlnet模型包并解压到/home/ma-user目录下。提供2种模型包下载方式。
   - 模型包直接下载（如果不能访问HuggingFace官网，推荐此方式）

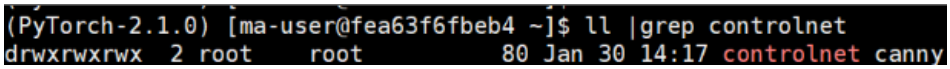
     下载到容器/home/ma-user目录下后，解压。

     ```
     cd  /home/ma-user/
     wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/controlnet_canny.zip
     unzip controlnet_canny.zip
     ```

   - 也可以从HuggingFace官网下载到本地后，通过docker cp命令复制到容器中/home/ma-user目录下。

     在线下载地址：**https://huggingface.co/diffusers/controlnet-canny-sdxl-1.0/tree/main**

     图 4-16 下载 controlnet 模型包并解压

4. 安装插件代码包。
   a. 将获取到的插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件上传到容器的/home/ma-user/temp目录下，并解压。

      ```
      cd /home/ma-user/temp
      -
      unzip AscendCloud-AIGC-6.3.907-xxx.zip #解压
      ```

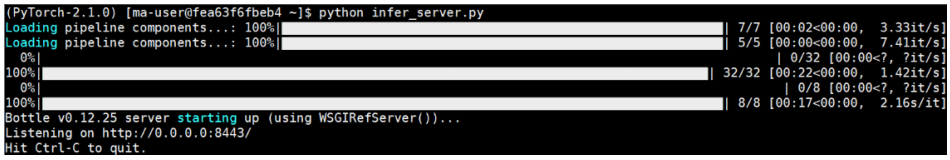
   b. 将获取到的ascendcloud-aigc-extensions-diffusers.tar.gz包复制到/home/ma-user下后解压。

```
docker cp ascendcloud-aigc-extensions-diffusers.tar.gz sdxl-diffusers:/home/ma-user/
tar -zxvf ascendcloud-aigc-extensions-diffusers.tar.gz
```

Step3 运行并验证 SDXL 模型

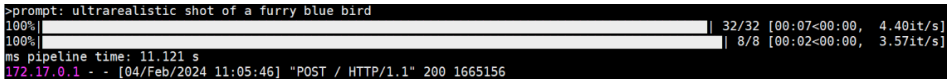
1. 首先在容器中运行命令。
   ```
   source /usr/local/Ascend/ascend-toolkit/set_env.sh
   ```
2. 在/home/ma-user目录下已经存在infer_server.py脚本文件，启动infer_server.py命令如下。
   ```
   python infer_server.py
   ```

   图 4-17 启动脚本

3. 在宿主机上另外打开一个终端，使用curl命令发送请求。完整的请求参数请参考**表4-6**。
   ```
   curl -kv -X POST localhost:8443/ -H "Content-Type: application/json" -d '{"prompt":"ultrarealistic shot of a furry blue bird"}'
   ```
   服务端打印如下信息，表示发送请求成功。

   图 4-18 发送请求

   客户端返回图像的base64编码。

   图 4-19 图像的 base64 编码

4. 将客户端返回的base64编码转换为图片。
   ```
   from PIL import Image
   from io import BytesIO
   import base64

   def base64_to_image(base64_str):
       image = base64.b64decode(base64_str, altchars=None, validate=False)
       image = BytesIO(image)
       image = Image.open(image)
       image.save(“./out_put_image.png”)
   ```

Step4 运行并验证带 controlnet 的模型

1. 首先下载一个默认输入文件。

   https://huggingface.co/lllyasviel/sd-controlnet-canny/blob/main/images/bird_canny.png

2. 文件下载后重命名为canny_input_bird.png，然后复制到容器/home/ma-user目录下，在宿主机上的执行命令如下。
```
mv bird_canny.png canny_input_bird.png
chmod 777 canny_input_bird.png
docker cp canny_input_bird.png sdxl-diffusers:/home/ma-user/
```
3. 在/home/ma-user目录下已经存在infer_server_with_controlnet.py脚本文件，运行带controlnet的sdxl，运行命令如下。
```
python infer_server_with_controlnet.py
```
4. 在宿主机上另外打开一个终端，使用curl命令发送请求。完整的请求参数请参考**表4-6**。
```
curl -kv -X POST localhost:8443/ -H "Content-Type: application/json" -d '{"prompt":"ultrarealistic shot of a furry blue bird"}'
```
服务端打印如下信息，表示发送请求成功。

带controlnet时，可以读取本地图片得到输入参数。
```
from diffusers.utils import load_image
from io import BytesIO
import base64

def image_to_base64(img_path):
    image = load_image(img_path)
    buffered = BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue())
```

附录 1：请求参数表

使用curl命令发送请求的请求参数表如下。

表 4-6 请求参数列表

参数	说明
prompt	正向文本，必选
negative_prompt	负向文本，非必选
height	图像高度，非必选
width	图像宽度，非必选
num_inference_steps	对图片进行噪声优化的次数，非必选
denoising_end	二阶段去噪，非必选
refiner_switch	refiner模型开关，是否开启refiner，非必选
seed	添加噪音的随机数种子，非必选
image_path	带controlnet时需要，此时image_path需要赋值null，传入图片的base64编码值，非必选
image_base64	带controlnet时需要，和image_path二选一，传入图片的base64编码值，非必选

4.4 SD3 Diffusers 框架基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。

方案概览

本方案介绍了在ModelArts Lite DevServer上使用昇腾计算资源Ascend Snt9B开展SD3模型的推理过程。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考表4-7获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

软件配套版本

表 4-7 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-8 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表4-7**。本案例使用的是解压到子目录aigc_inference/torch_npu/diffusers/0.29.2/目录下的所有文件，将该目录上传到宿主机上。

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像diffusers-sd3-inference:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见表4-8。

```
FROM {image_url}

COPY --chown=ma-user:ma-group diffusers/0.29.2 /home/ma-user/diffusers

RUN cd /home/ma-user/diffusers && sh prepare.sh

COPY -r --chown=ma-user:ma-group attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py

RUN pip install transformers
RUN pip install accelerate
RUN pip install sentencepiece

WORKDIR /home/ma-user/diffusers
```

构建自定义镜像diffusers-sd3-inference:0.0.1。

```
docker build -t diffusers-sd3-inference:0.0.1 .
```

Step4 启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge diffusers-train:0.0.1 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step4 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

Step5 启动推理

本章节介绍SD3模型的推理过程。使用官方提供的已经训练好的模型进行推理，输入prompt生成指定像素的图片。

1. 使用如下命令登录huggingface，并输入个人账号的token：

   ```
   huggingface-cli login
   ```

2. 执行如下命令运行推理脚本启动SD3服务：

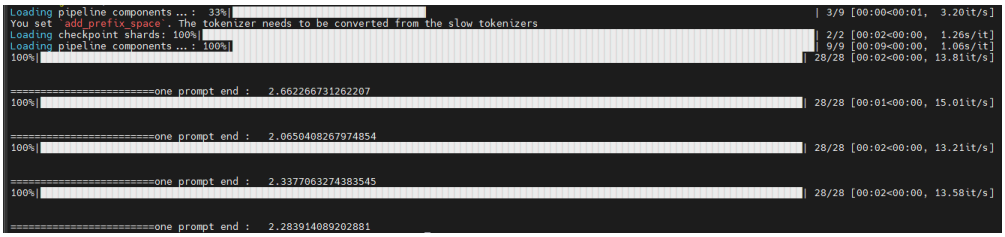
   ```
   #配置环境变量
   export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

   python run_inference.py
   ```

   参数说明：

- height、width: 指定生成图片的长和宽，例如：512、960、1024
- prompt_list: prompt列表，可以自行修改。

推理执行成功如下图所示。

图 4-20 推理执行成功



4.5 SD WEBUI 套件适配 PyTorch NPU 的推理指导（6.3.907）

4.5.1 SD WebUI 推理方案概览

本文档主要介绍如何在ModelArts的DevServer和ModelArts Standard环境上部署Stable Diffusion的WebUI套件，使用NPU卡进行推理。

约束限制

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

适配的Cann版本是cann_8.0.rc2。

资源规格要求

ModelArts Lite DevServer或ModelArts Stanard专属资源池的资源：

- 使用Ascend Snt9B单机单卡规格。
- 推荐使用“西南-贵阳一”Region上的昇腾资源。

软件配套版本

本方案支持的软件配套版本和依赖包获取地址如表4-9所示。

表 4-9 软件配套版本和获取地址

软件名称	说明	下载地址
插件代码包	AscendCloud-3rdAIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-10 基础容器镜像地址

镜像用途	镜像地址	Cann版本
基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2

4.5.2 在 DevServer 上部署 SD WebUI 推理服务

本章节主要介绍如何在ModelArts的DevServer环境上部署Stable Diffusion的WebUI套件，使用NPU卡进行推理。

Step1 准备环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 检查环境。

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

   ```
   npu-smi info
   ```

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

   b. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 制作自定义镜像

准备以下文件用于制作镜像。

- 下载并解压**表4-9**中的AscendCloud插件包，进入aigc_inference/torch_npu/webui/v1_9_0_RC/ 和aigc_inference/torch_npu/diffusers/0_21_2/：

```
v1_9_0_RC
├── gradio_adapt
    ├── gradio-3.14.2
        ├── Button-748313a7.js
        ├── index-2519a27e.js
├── ascend_extension
    ├── scripts
        ├── AscendPlugin.py
    ├── config.py
    ├── ...
0_21_2
├── ascend_diffusers
    ├── src
    ├── setup.py
    ├── ...
```

- 下载sd基础模型。

  下载v1-5模型：https://huggingface.co/runwayml/stable-diffusion-v1-5/blob/main/v1-5-pruned-emaonly.safetensors

  下载sdxl_base模型：https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors

- 编写Dockerfile文件：

  基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-10**。

```
FROM {image_url}


# 下载sd webui源码
RUN mkdir /home/ma-user/sdwebui
RUN cd /home/ma-user/sdwebui && git config --global http.sslVerify false && git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui.git

# 切换到1.9.0版本
RUN cd /home/ma-user/sdwebui/stable-diffusion-webui && git checkout e164031

# 下载controlnet插件
RUN cd /home/ma-user/sdwebui/stable-diffusion-webui/extensions && git clone https://github.com/Mikubill/sd-webui-controlnet.git
RUN cd /home/ma-user/sdwebui/stable-diffusion-webui/extensions/sd-webui-controlnet && git checkout 92e4b12a73e61db6c1332dd52d9c35d59a7ebee1

# 下载nsfw插件
RUN cd /home/ma-user/sdwebui/stable-diffusion-webui/extensions && git clone https://github.com/w-e-w/sd-webui-nudenet-nsfw-censor.git

# 安装依赖
WORKDIR /home/ma-user/sdwebui/stable-diffusion-webui
RUN pip install -r requirements.txt -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
```

```
COPY --chown=ma-user:ma-group v1-5-pruned-emaonly.safetensors /home/ma-user/sdwebui/stable-diffusion-webui/models/Stable-diffusion
COPY --chown=ma-user:ma-group sd_xl_base_1.0.safetensors /home/ma-user/sdwebui/stable-diffusion-webui/models/Stable-diffusion

# 复制华为侧代码包和插件
COPY --chown=ma-user:ma-group index-2519a27e.js /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/gradio/templates/frontend/assets
COPY --chown=ma-user:ma-group Button-748313a7.js /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/gradio/templates/frontend/assets
COPY --chown=ma-user:ma-group ascend_extension /home/ma-user/sdwebui/stable-diffusion-webui/extensions/ascend_extension
COPY --chown=ma-user:ma-group ascend_diffusers /home/ma-user/sdwebui/ascend_diffusers

# 安装ascend_diffusers
RUN cd /home/ma-user/sdwebui/ascend_diffusers && pip install -e .
WORKDIR /home/ma-user/sdwebui/stable-diffusion-webui

# 禁用ssl验证
RUN pip install requests==2.27
RUN sed -i 's/self.verify = True/self.verify = False/g' /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/requests/sessions.py
# 禁止github上ssl验证
RUN sed -i 's/-m pip {command} --prefer-binary{index_url_line}/-m pip {command} --prefer-binary{index_url_line} --trusted-host github.com --trusted-host codeload.github.com/g' /home/ma-user/sdwebui/stable-diffusion-webui/modules/launch_utils.py
# 禁用ssl验证
RUN sed -i '1i\import ssl\' launch.py && sed -i '2i\ssl._create_default_https_context = ssl._create_unverified_context\' launch.py && sed -i 's#\r##g' launch.py
```

宿主机上文件目录如下：

```
<docker_build>
├── v1-5-pruned-emaonly.safetensors   #sd基础模型
├── sd_xl_base_1.0.safetensors  #sd基础模型
├── index-2519a27e.js                    #  华为侧提供的代码文件
├── Button-748313a7.js                    #  华为侧提供的代码文件
├── ascend_extension        #  华为侧提供的插件包
├── ascend_diffusers  #  华为侧提供的插件包
├── Dockerfile                    #  Dockerfile文件
```

在该目录下执行命令构建自定义镜像：

```
docker build -t sdxl-train:0.0.1 .
```

Step3 启动自定义镜像

执行以下命令启动自定义镜像。

```
docker run -itd --name ${container_name} -p 8183:8183 -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge sdxl-train:0.0.1 bash
```

参数说明：

--name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。

--device=/dev/davinci0：挂载NPU设备，示例中挂载了单张卡davinci0。

说明

- driver及npu-smi需同时挂载至容器。
- 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

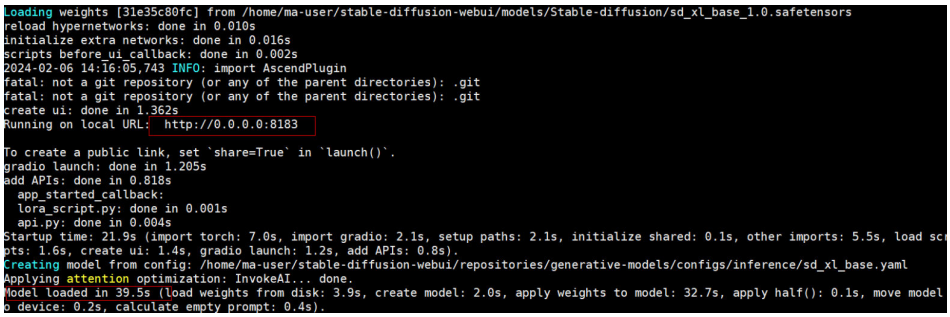
Step4 进入容器运行

1. 进入容器后执行启动命令。

```
docker exec -it ${container_name} bash
python3 launch.py --port 8183 --skip-torch-cuda-test --enable-insecure-extension-access --listen --log-startup --disable-safe-unpickle --api
```

等待克隆仓库，下载依赖模型，启动成功后显示

图 4-21 启动成功后显示



2. 验证效果。

a. 新开启一个终端，执行以下命令。

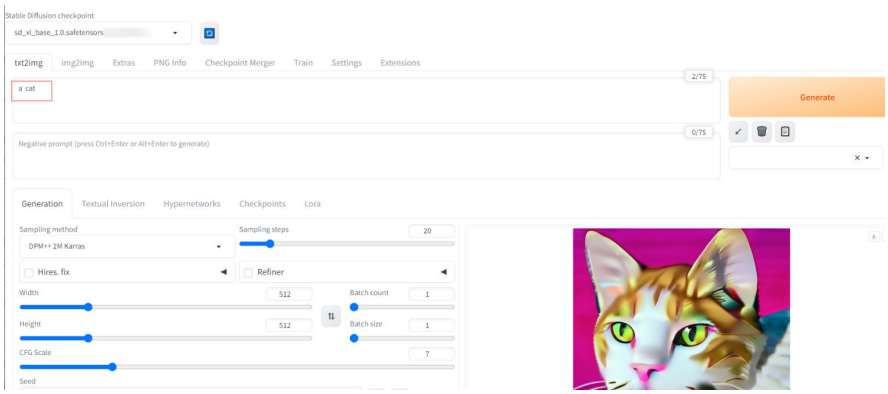
```
curl --noproxy '*' -kv -X POST localhost:8183/sdapi/v1/txt2img -H "Content-Type: application/json" -d '{"prompt":"ultrarealistic shot of a furry blue bird"}'
```

执行成功显示：

图 4-22 执行成功显示

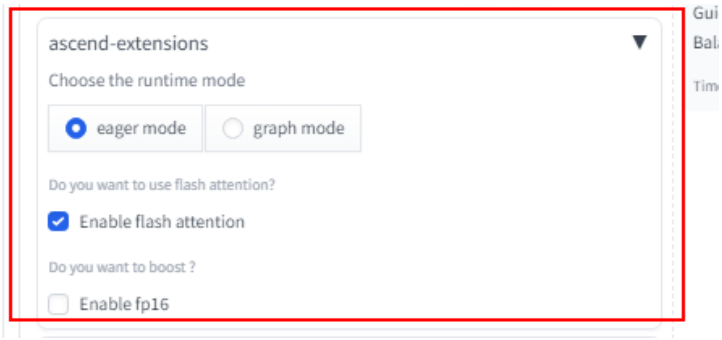
b. 在浏览器输入http://{宿主机ip}:8183，可以访问前端页面，通过输入文字生成图片。

图 4-23 输入文字生成图片



注意需要勾选Enable Flash Attention按钮。

图 4-24 Enable Flash Attention 优化按钮



4.5.3 在 Standard 上部署 SD WebUI 推理服务

本文档主要介绍如何在ModelArts Standard的推理环境上部署Stable Diffusion的WebUI套件，使用NPU卡进行推理。

完成**在DevServer上部署SD WebUI推理服务**章节的任务后，如果还需要在ModelArts的推理生产环境（ModelArts控制台的在线服务模块）中部署推理服务，可参考下述步骤。

Step1 导出镜像

完成**在DevServer上部署SD WebUI推理服务**章节的任务后，在宿主机上执行以下命令，导出镜像。

```
docker commit ${container_name} sdxl-train:0.0.1
```

Step2 创建镜像组织

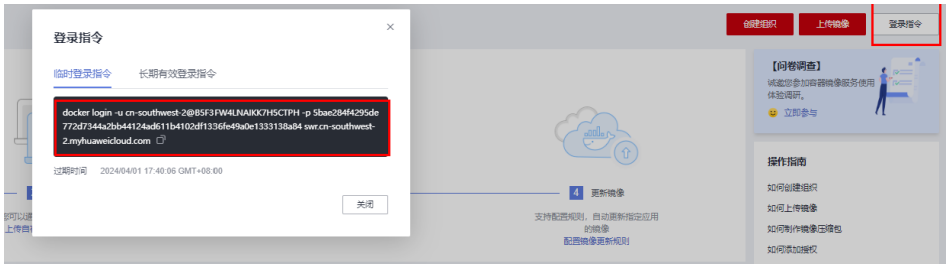
在SWR服务页面创建镜像组织。

图 4-25 创建镜像组织

Step3 在宿主机上传镜像到 SWR

在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中复制临时登录指令，即可完成登录。

图 4-26 复制登录指令



登录指令输入之后，使用下列示例命令：

```
docker tag sdxl-train:0.0.1 <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
docker push <镜像仓库地址>/<组织名称>/<镜像名称>:<版本名称>
```

参数说明：

- <镜像仓库地址>：可在SWR控制台上查询，容器镜像服务中登录指令末尾的域名即为镜像仓库地址。
- <组织名称>：前面步骤中自己创建的组织名称。示例：ma-group
- <镜像名称>:<版本名称>：定义镜像名称。示例：sdxl-train:0.0.1

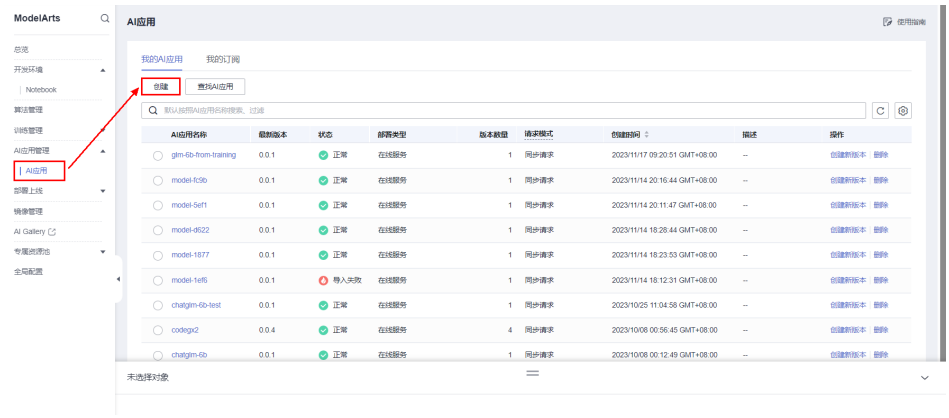
以贵阳一的SWR为例：

```
docker tag sdxl-train:0.0.1 swr.cn-southwest-2.myhuaweicloud.com/ma-group/sdxl-train:0.0.1
docker push swr.cn-southwest-2.myhuaweicloud.com/ma-group/sdxl-train:0.0.1
```

Step4 创建 AI 应用

在ModelArts的AI应用页面，进行AI应用创建。

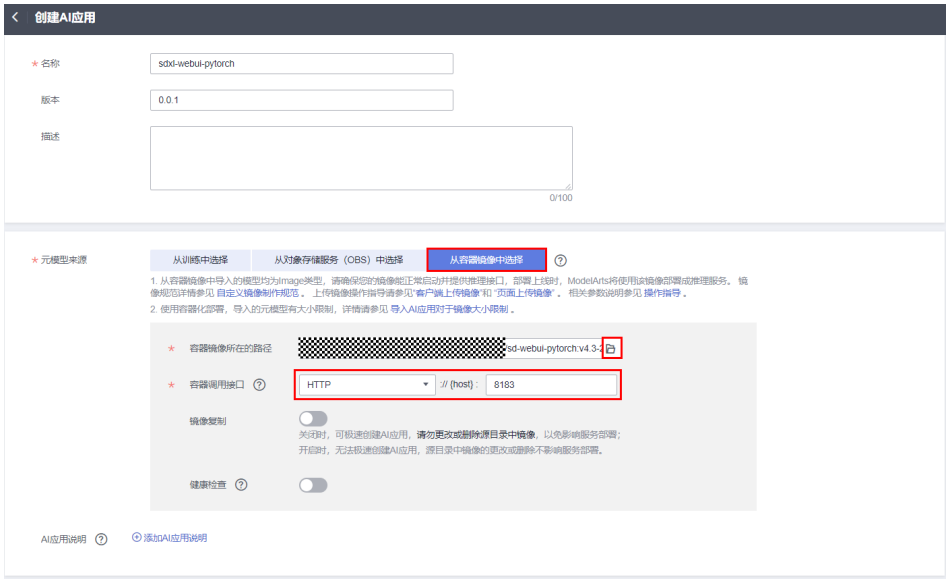
图 4-27 创建 AI 应用



填写如下参数信息。

- 名称：AI应用的名称，请按照实际应用名填写。
- 版本：版本描述，请按照实际填写。
- 元模型来源：注意此处选择“从容器镜像选择”。
- 容器镜像所在路径：点击文件夹标签，选择已经制作好的镜像。
- 容器调用接口参数：根据镜像实际提供的协议和端口填写，本案例中的SDXL镜像提供HTTP服务和8183端口。

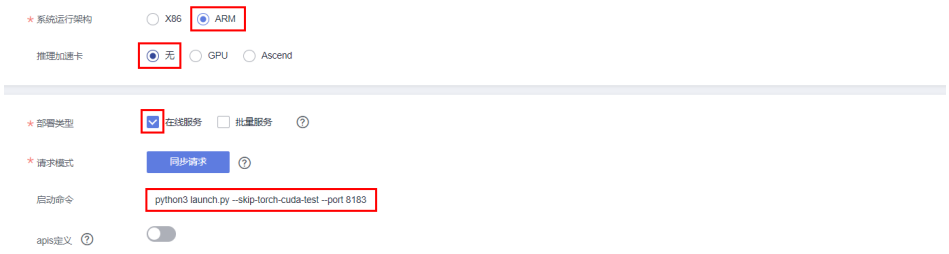
图 4-28 填写参数（1）



- 系统运行架构： 选择ARM.
- 推理加速卡：无。
- 部署类型： 在线服务。
- 请求模式：同步请求。
- 启动命令：
  source /etc/bashrc && python3 launch.py --skip-torch-cuda-test --port 8183 --enable-insecure-extension-access --listen --log-startup --disable-safe-unpickle --skip-prepare-environment --api

按照上述配置完参数后，点击右下角的立即创建， 完成AI应用的创建。

图 4-29 填写参数（2）



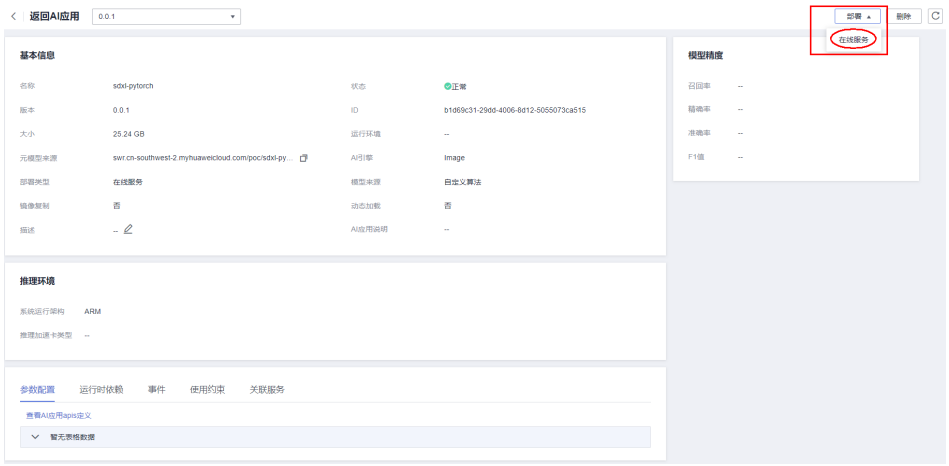
当AI应用状态变为正常时，表示创建完成。

图 4-30 AI 应用创建完成

Step5 部署服务

单击AI应用名称，进入AI应用详情页，点击部署在线服务。

图 4-31 部署在线服务

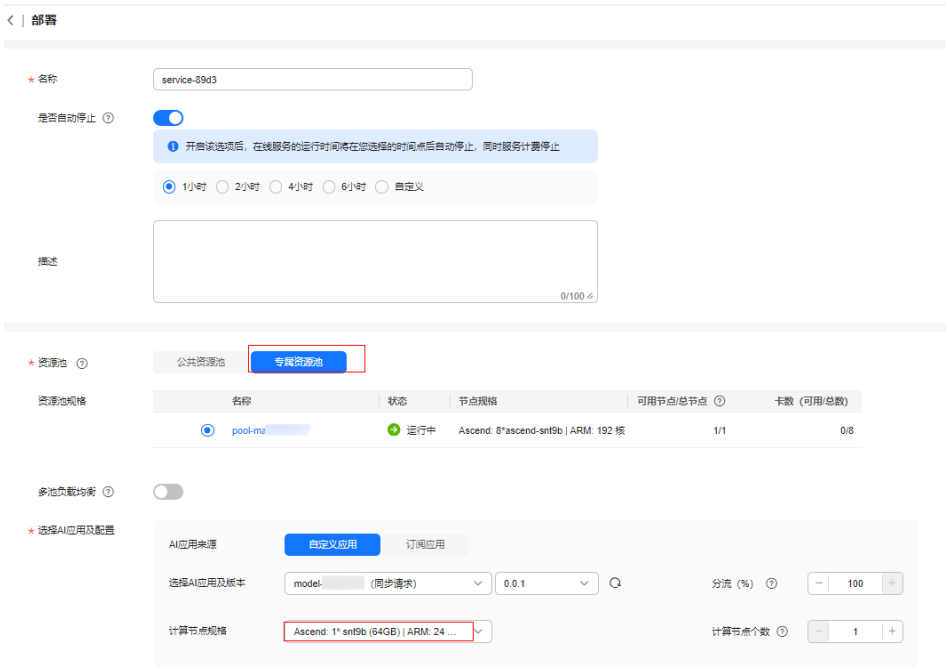


填写如下服务部署参数。

- 名称： 服务的名称，按照实际需要填写
- 是否自动停止：如果配置自动停止，服务会按照配置的时间自动停止。如果需要常驻的服务，建议关掉该按钮。
- 描述：按照需要填写。
- 资源池：选择专属资源池。若之前未购买专属资源池，具体步骤请参考**创建资源池**。

  资源规格要求：
  - 硬盘空间：至少200GB。
  - 昇腾资源规格：可以申请Ascend: 1* ascend-snt9b(32GB)或Ascend: 1* ascend-snt9b(64GB)规格。请按需选择需要的规格，64GB规格的推理耗时更短。
  - 推荐使用“西南-贵阳一”Region上的昇腾资源。
- AI应用来源： 我的AI应用。
- 选择AI应用及其版本：此处选择上一步中创建的sdxl-webui-pytorch:0.0.1应用。
- 计算节点规格： 按需选择Ascend: 1* ascend-snt9b(32GB)或Ascend: 1* ascend-snt9b(64GB)。

图 4-32 填写服务部署参数



选择开启APP认证并选择应用。

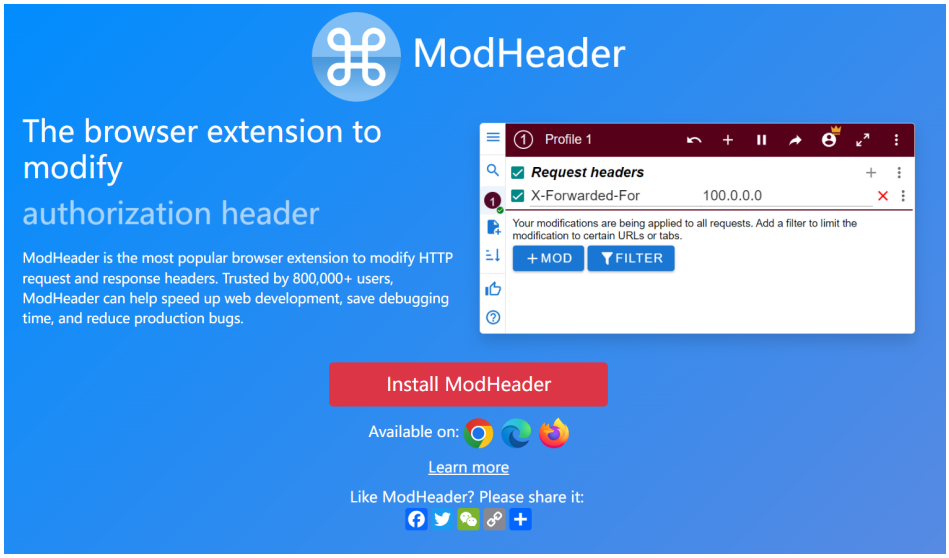
图 4-33 开启 APP 认证

按照上述配置完参数后，单击“下一步”， 确认信息无误后，单击“提交”，完成服务的部署。

Step6 访问在线服务

在Chrome浏览器中安装**ModHeader插件**。

图 4-34 安装 ModHeader 插件

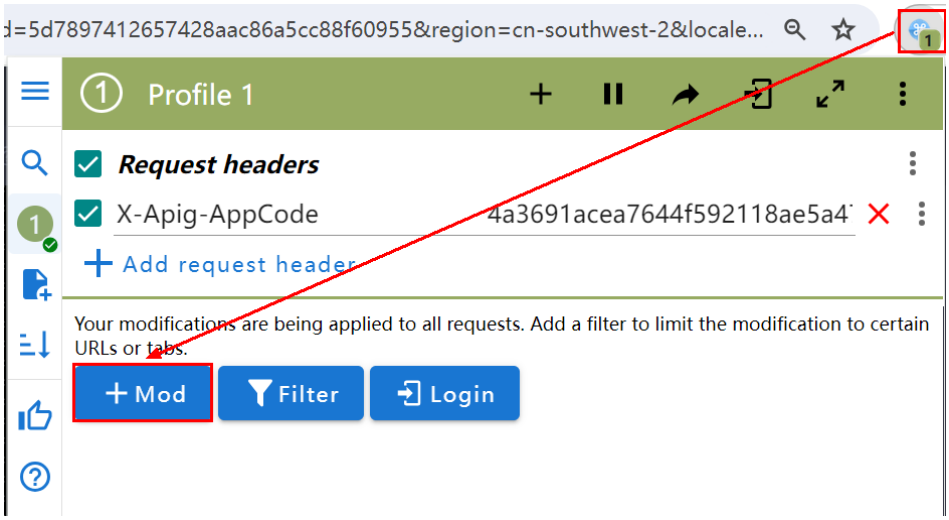


说明

Chrome浏览器安装ModHeader插件后，可能会导致访问不了Modelarts平台，访问Modelarts时需要临时禁用ModHeader插件。或者使用Edge登录Modelarts，使用Chrome安装插件访问页面。

打开ModHeader，点击添加MOD。

图 4-35 添加 MOD



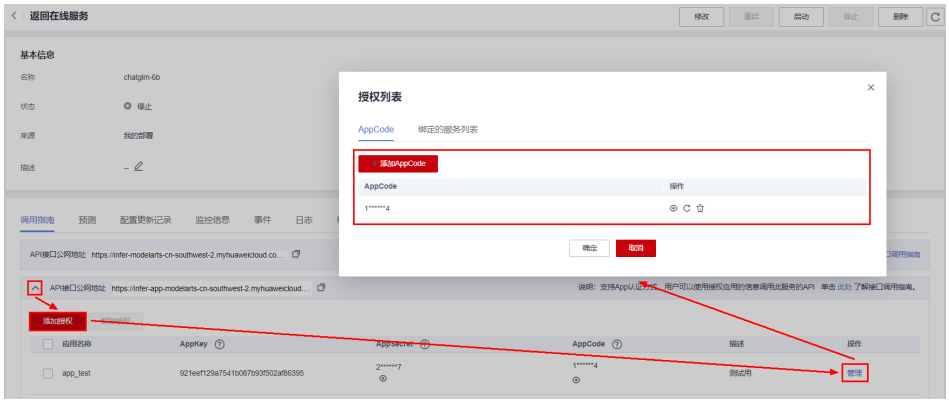
选择添加Request header。

图 4-36 添加 Request header

进入在线服务详情，查看Key值和Value值。

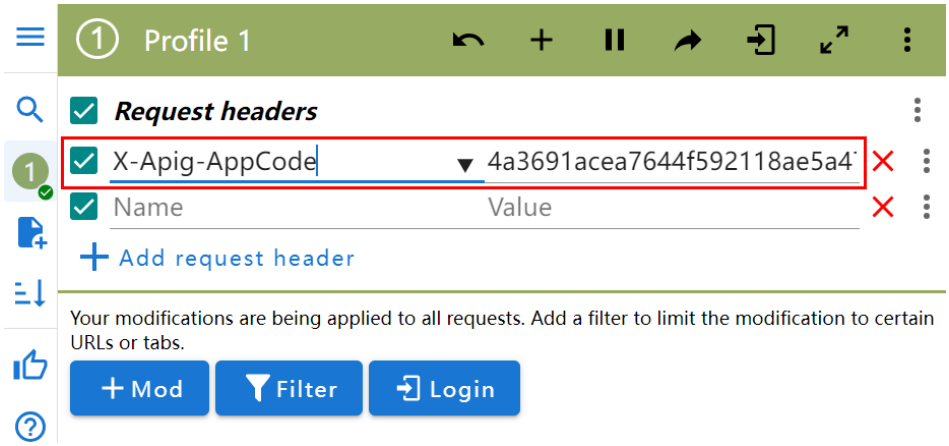
Key值固定为X-Apig-AppCode，Value值为APP认证的app_code值，在服务调用指南tab的APP认证API处展开，进行AppCode管理设置。

图 4-37 获取 Key 值和 Value 值



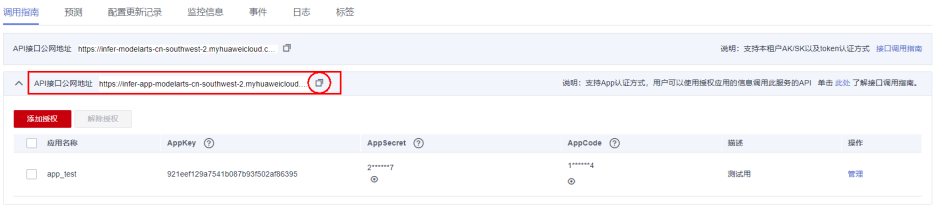
将在ModHeader插件中添加Key值和Value值。

图 4-38 添加 Key 和 value



进入在线服务详情页，查看APP认证方式的服务API。

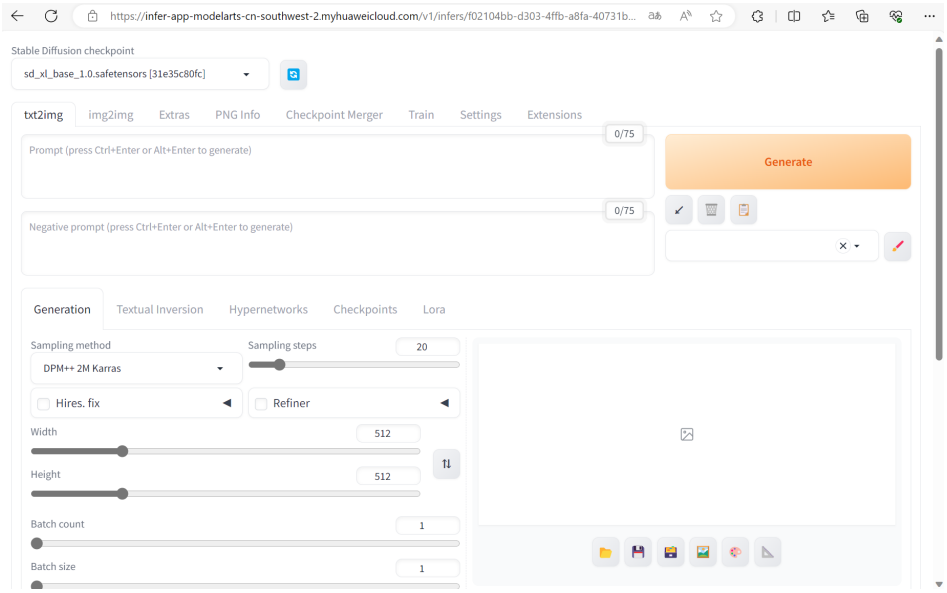
图 4-39 API 接口公网地址



复制API接口公网地址，并在地址后添加"/"，进行页面访问，例如：

```
https://infer-app-modelarts-cn-southwest-2.myhuaweicloud.com/v1/infers/abc104bb-d303-4ffb-a8fa-XXXXXXXXX/
```

图 4-40 访问在线服务



输入Promt，修改所需要的请求参数（如Width、Height），进行Promt请求。

图 4-41 填写请求参数

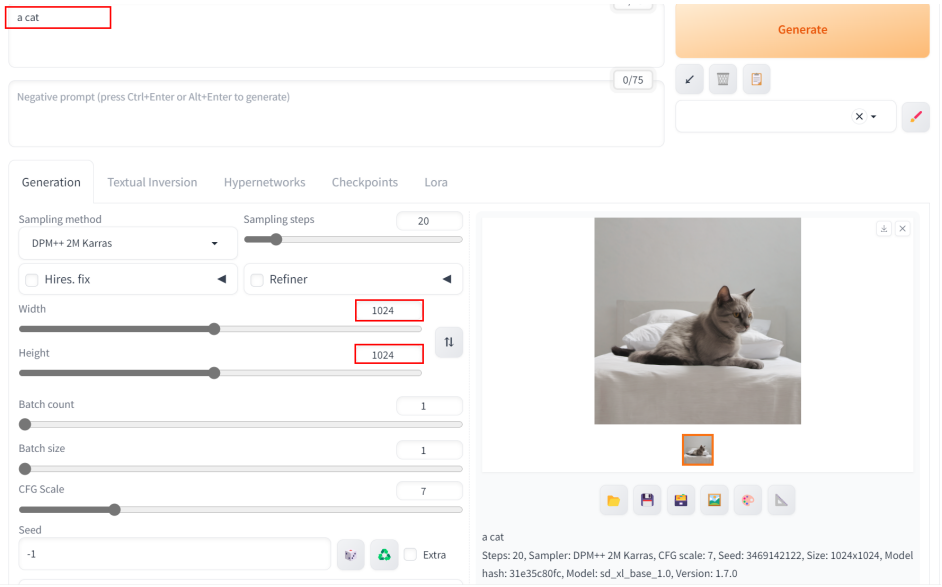
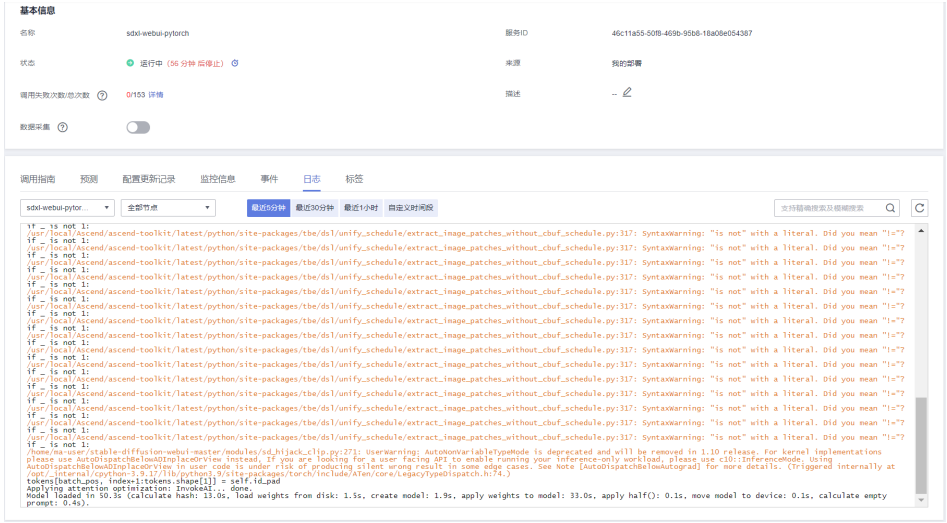


表 4-11 SDXL 模型参数及其含义

参数名称	说明	是否必选	默认值
prompt	提示词，根据提示词生成含有对应内容的图像	是	无
negative_prompt	反向提示词，图像生成过程中应避免的提示	否	无
num_inference_steps	推理步骤数，控制推理的步数	否	40
height	生成图像的纵向分辨率	否	1024
width	生成图像的横向分辨率	否	1024
high_noise_frac	高噪声比例，即基础模型跑的步数占总步数的比例	否	0.8
refiner_switch	是否使用细化模型refiner	否	true（使用）
seed	随机种子，控制生成图像的多样性	否	无

您可在ModelArts控制台查看相关日志。

图 4-42 查看相关日志



说明

首次请求时会进行模型加载，耗时较长，因此第一个请求可能超时，第二个请求将会正常，请耐心等待。

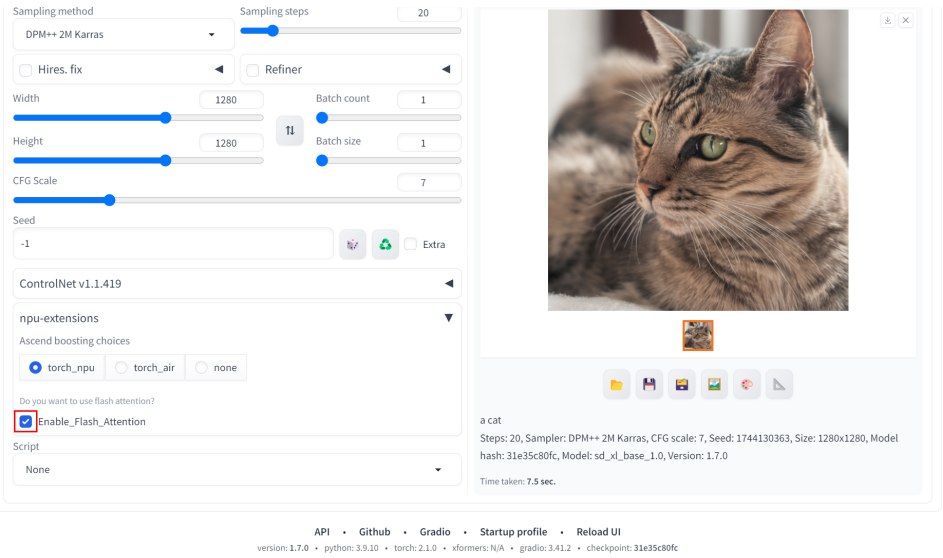
4.5.4 SD WebUI 推理性能测试

以下性能测试数据仅供参考。

- 开启Flash Attention

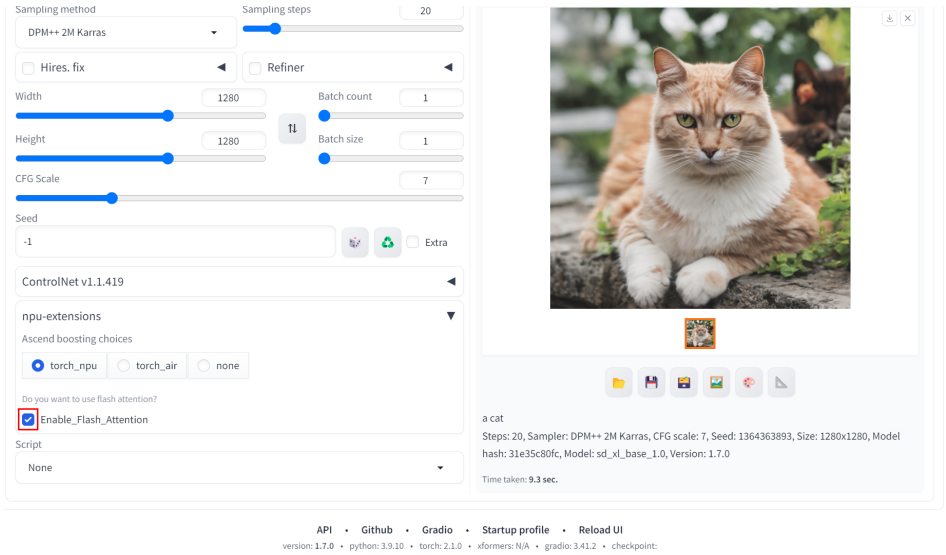
  生成1280x1280图片，使用Ascend: 1* ascend-snt9b(64GB)，约耗时7.5秒。

  图 4-43 生成图片耗时（1）



  生成1280x1280图片，使用Ascend: 1* ascend-snt9b(32GB)，约耗时9.3秒。

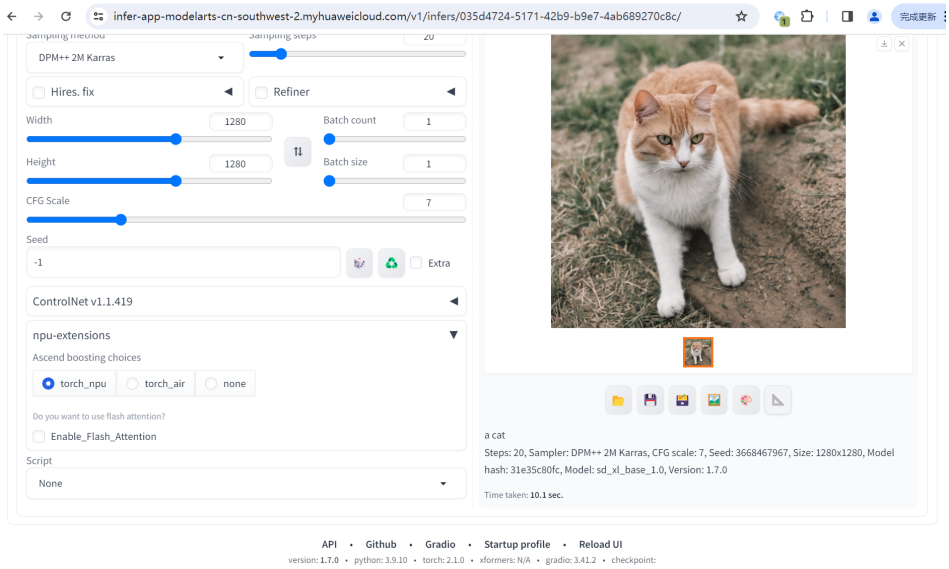
图 4-44 生成图片耗时（2）



- 不开启Flash Attention

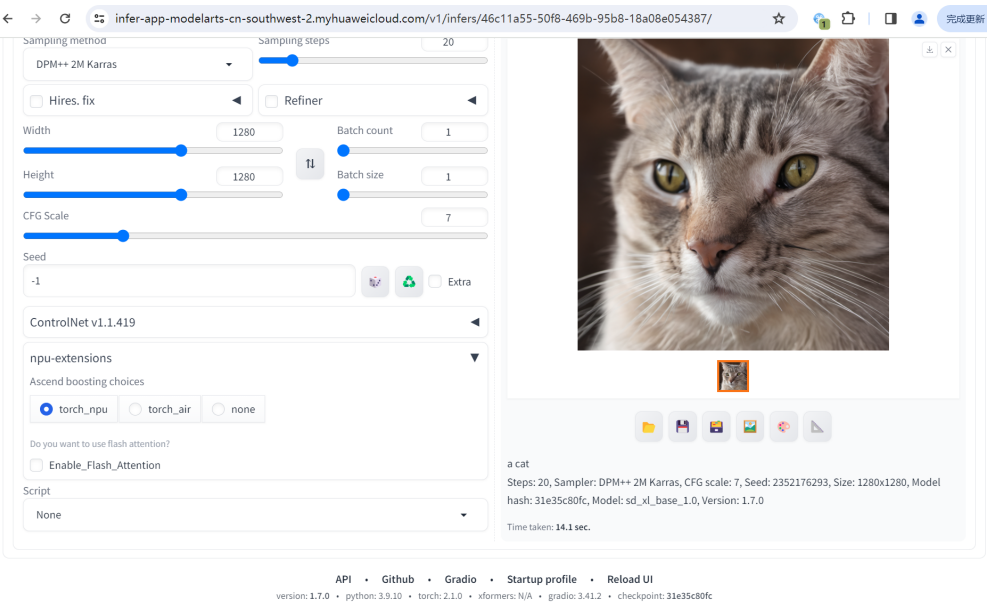
  生成1280x1280图片，使用Ascend: 1* ascend-snt9b(64GB)，约耗时10.1秒。

图 4-45 生成图片耗时（3）



生成1280x1280图片，使用Ascend: 1* ascend-snt9b(32GB)，约耗时14.1秒。

图 4-46 生成图片耗时（4）



4.6 SD1.5&SDXL Koyha 框架基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907）

4.6.1 训练场景和方案介绍

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。

方案概览

本方案介绍了在ModelArts Lite DevServer上使用昇腾计算资源Ascend Snt9B开展SDXL和SD1.5模型的训练过程，包括Finetune训练、LoRA训练和Controlnet训练。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考表4-12获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Finutune训练使用单机8卡资源。
- Lora训练使用单机单卡资源。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

软件配套版本

表 4-12 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-13 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

4.6.2 准备镜像环境

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载模型包、依赖代码包和数据集并上传到宿主机

1. 下载**stable-diffusion-v1-5模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/runwayml/stable-diffusion-v1-5/tree/main**
2. 下载**stable-diffusion-xl-base-1.0模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
3. 下载**sdxl-vae-fp16-fix模型包**并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
4. 下载开源数据集**pokemon-dataset**并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/sayannath/pokemon-dataset/tree/main**。用户也可以使用自己的数据集。
5. 下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表4-12**。本案例使用的是解压到子目录aigc_train->torch_npu->koyha_ss的所有文件，将koyha_ss整个目录上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。
```
[root@devserver docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 koyha_ss
drwx------  3 root root   4096 May  7 10:50 datasets
  drwx------  3 root root   4096 May  7 10:50 pokemon-dataset
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3 构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-v1-5
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
drwx------  2 root root   4096 Apr 30 15:17 sdxl-vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像koyha_ss-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-13**。

```
FROM {image_url}

COPY --chown=ma-user:ma-group koyha_ss /home/ma-user/koyha_ss
RUN cd /home/ma-user/koyha_ss && sh prepare.sh
COPY --chown=ma-user:ma-group stable-diffusion-v1-5 /home/ma-user/stable-diffusion-v1-5
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group sdxl-vae-fp16-fix /home/ma-user/sdxl-vae-fp16-fix
```

```
COPY --chown=ma-user:ma-group datasets /home/ma-user/datasets
WORKDIR /home/ma-user/koyha_ss
```

构建自定义镜像diffusers-train:0.0.1。

```
docker build -t koyha_ss-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数，Lora微调启动单卡，finetune微调启动八卡。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge koyha_ss-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step4 进入容器

1. 通过容器名称进入容器中。默认使用ma-user用户执行后续命令。
   ```
   docker exec -it ${container_name} bash
   ```
2. 上传代码文件到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。
   ```
   #统一文件属主为ma-user用户
   sudo chown -R ma-user:ma-group  ${container_work_dir}
   # ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
   #例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
   ```

4.6.3 Finetune 训练

本章节介绍SDXL&SD 1.5模型的Finetune训练过程。Finetune是指在已经训练好的模型基础上，使用新的数据集进行微调（ fine-tuning ）以优化模型性能。修改数据集路径、模型路径。数据集路径格式为/datasets/pokemon-dataset/image_0.png，脚本里写到pokemon-dataset路径即可。

kohya_finetune.toml文件里数据集路径更改为pokemon-dataset路径。

```
cd koyha_ss
cp run_* sd-scripts
cp kohya_finetune.toml sd-scripts
cp meta_cap.json sd-scripts
cd sd-scripts
vim run_finetune.sh
vim kohya_finetune.toml
python finetune/make_captions.py {数据集路径pokemon-dataset路径}
```

启动 SD1.5 Finetune 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh run_finetune.sh
```

所有数据保存在auto_log/avg_step_time.txt文本中 auto_log/log/目录下存放各个shapes的数据

4.6.4 LoRA 训练

本章节介绍SDXL&SD 1.5模型的LoRA训练过程。LoRA训练是指在已经训练好的模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。修改数据集路径、模型路径。脚本里写到datasets路径即可。

run_lora_sdxl中的vae路径要准确写到sdxl_vae.safetensors文件路径。

```
vim run_lora.sh
vim run_lora_sdxl.sh
```

启动 SD1.5 LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh run_lora.sh
```

所有数据保存在auto_log/avg_step_time.txt文本中 auto_log/log/目录下存放各个shapes的数据

启动 SDXL LoRA 训练服务

使用ma-user用户执行如下命令运行训练脚本。

```
sh run_lora_sdxl.sh
```

所有数据保存在autoxl_log/avg_step_time.txt文本中 autoxl_log/log/目录下存放各个shapes的数据

4.7 Open-Sora-Plan1.0 基于 DevServer 适配 PyTorch NPU 训练推理指导（6.3.907）

本文档主要介绍如何在ModelArts Lite DevServer上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora-Plan1.0训练和推理。

方案概览

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

适配的Cann版本是cann_8.0.rc2。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表4-14**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

软件配套版本

表 4-14 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-15 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```

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

3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表4-14**。本案例使用的是解压到子目录 multimodal_algorithm/OpenSoraPlan1.0/ 目录下的所有文件，将该目录上传到宿主机上。

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像Open-Sora-Plan1.0:1.0。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-15**。

```
FROM {image_url}

COPY --chown=ma-user:ma-group OpenSoraPlan1.0/* /home/ma-user/Open-Sora-Plan1.0

RUN cd /home/ma-user/Open-Sora-Plan1.0 && bash prepare.sh

WORKDIR /home/ma-user/Open-Sora-Plan1.0
```

构建自定义镜像Open-Sora-Plan1.0:1.0。

```
docker build -t Open-Sora-Plan1.0:1.0 .
```

Step4 启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。
```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge Open-Sora-Plan1.0:1.0 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step5 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

Step6 安装 Decord

Decord是一个高性能的视频处理库，在昇腾环境中安装需要修改一些源码进行适配。

Decord建议安装在 /home/ma-user/lib中。

1. 安装x264

```
mkdir /home/ma-user/lib && cd /home/ma-user/lib
# X264 install
git clone https://github.com/mirror/x264.git
export PKG_CONFIG_PATH=/home/ma-user/lib/lib/pkgconfig
cd x264
./configure --enable-shared --prefix=/home/ma-user/lib/
make
make install
cd ..
```

2. 安装ffmpeg

```
# ffmpeg install
git clone https://github.com/FFmpeg/FFmpeg.git
export TMPDIR=../tmp-ffmpeg
cd FFmpeg
git checkout 78f55457c9be420f4109da45de42a36338d56aca
# git checkout release/4.0
./configure --enable-shared --enable-swscale --enable-gpl --enable-nonfree --enable-pic --prefix=/
home/ma-user/lib/ --enable-version3 --enable-postproc --enable-pthreads --enable-static --enable-
libx264
make
make install
cd ..
```

3. 安装decord
   a. 下载decord代码。

```
git clone --recursive https://github.com/dmlc/decord
cd decord
```

   b. 第一处修改

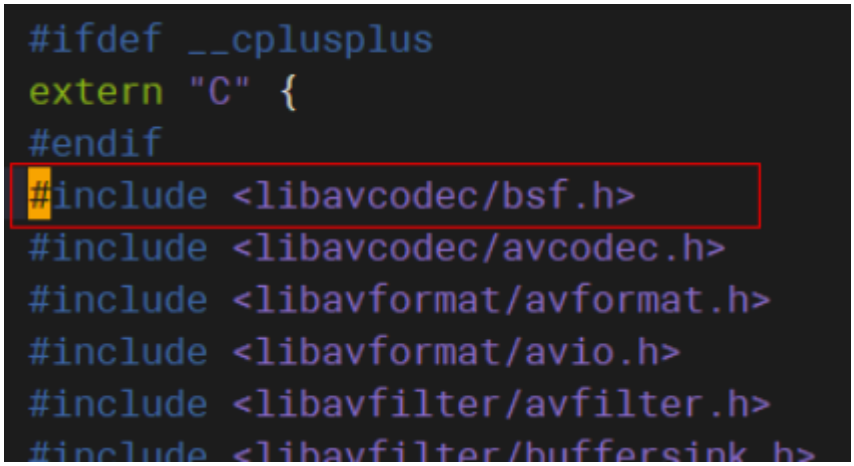
```
vim src/video/ffmpeg/ffmpeg_common.h
```

   在文件ffmpeg_common.h的23行，添加如下内容

```
#include <libavcodec/bsf.h>
```

图 4-47 文件 ffmpeg_common.h 修改前

图 4-48 文件 ffmpeg_common.h 修改后



c. 第二处修改：

```
vim src/video/video_reader.cc
```

在文件video_reader.cc的149行，进行修改：

```
    const AVCodec** dec_const = const_cast<const AVCodec**>(&dec);
    // int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1,
&dec, 0);
    int st_nb = av_find_best_stream(fmt_ctx_.get(), AVMEDIA_TYPE_VIDEO, stream_nb, -1,
dec_const, 0);
```

图 4-49 文件 video_reader.cc 修改前

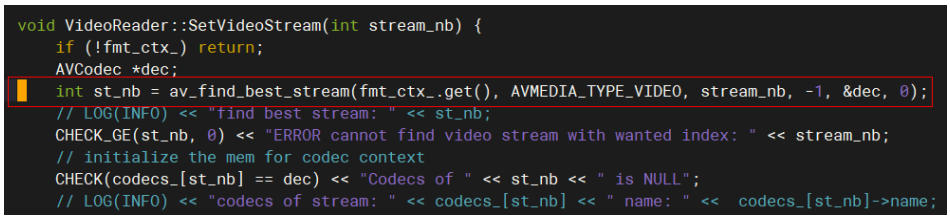
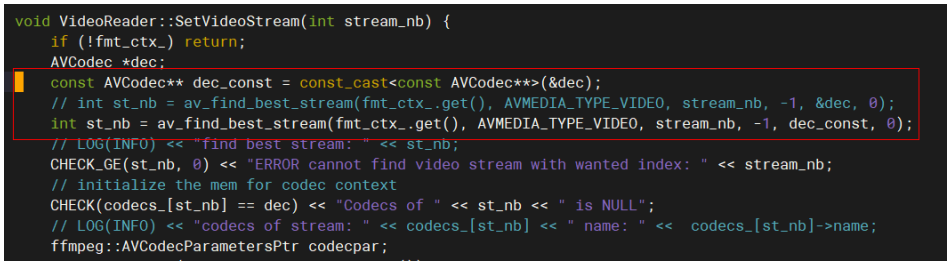


图 4-50 文件 video_reader.cc 修改后



d. 执行如下命令编译安装decord。

```
mkdir build && cd build
cmake .. -DUSE_CUDA=0 -DCMAKE_BUILD_TYPE=Release -DFFMPEG_DIR=/home/ma-user/lib/
make
cd ../python
python3 setup.py install --user
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/ma-user/lib/lib
echo "export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:/home/ma-user/lib/lib" >> ~/.bashrc
```

如果重启docker后，环境变量需要重新配置，否则会报错找不到 libavformat.so.60: cannot open shared object file: No such file or directory

配置方式如下：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/ma-user/lib/lib
```

Step7 下载数据集

先创建文件夹用来存放数据集。

```
mkdir datasets
cd datasets
```

训练使用的开源数据集链接：https://huggingface.co/datasets/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main。

由于数据集比较大，可以自行选择部分数据集手动下载解压，并放入 ./datasets文件夹下。

例如：这里下载了上述链接中mixkit.tar.gz和sharegpt4v_path_cap_64x512x512.json。

（备注：如果只下载了部分数据集，需要对应修改sharegpt4v_path_cap_64x512x512.json文件）

解压数据集：

```
tar -xzvf mixkit.tar.gz
```

解压后的数据集结果如图所示。

图 4-51 解压后的数据集文件

```
total 26607068
drwxr-x--- 10 root root         147 Jul 30 15:54 mixkit
-rw-------  1 root root 27242973015 Jul 30 15:54 mixkit.tar.gz
-rw-------  1 root root     2659687 Jul 30 15:54 sharegpt4v_path_cap_64x512x512.json
```

Step8 下载权重文件

建议手动下载所需的权重文件，在/home/ma-user/Open-Sora-Plan1.0/目录下进行操作。

1. 创建文件夹存放不同的权重文件。

   ```
   mkdir weights
   mkdir weights_t5
   mkdir cache_dir
   ```

2. 下载基础模型权重**t2v.pt**放到cache_dir文件夹下。

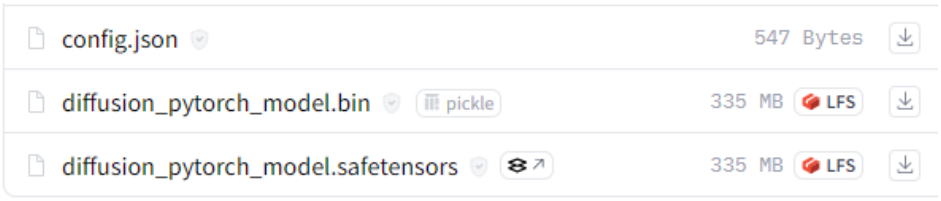
   ```
   mkdir Latte
   cd Latte
   git clone  -c http.sslVerify=false  https://huggingface.co/maxin-cn/Latte
   cd Latte
   git reset  --hard  83bdc71f7211963153464859d03d46d707e77865
   ```

   ```
   total 24
   -rw-r----- 1 root root 175 Aug 15 09:50 README.md
   -rw-r----- 1 root root 135 Aug 15 09:48 ffs.pt
   drwxr-x--- 2 root root  72 Aug 15 09:48 sd-vae-ft-ema
   drwxr-x--- 2 root root  72 Aug 15 09:48 sd-vae-ft-mse
   -rw-r----- 1 root root 135 Aug 15 09:48 skytimelapse.pt
   -rw-r----- 1 root root 135 Aug 15 09:50 t2v.pt
   -rw-r----- 1 root root 135 Aug 15 09:48 [illegible]nd.pt
   -rw-r----- 1 root root 135 Aug 15 09:48 ucf101.pt
   drwxr-x--- 2 root root  72 Aug 15 09:48 vae
   ```

然后将该目录下的t2v.pt文件复制到/home/ma-user/Open-Sora-Plan1.0/cache_dir目录下。

3. 下载VAE权重**vae-ft-mse-840000-ema-pruned.ckpt**和配置文件**config.json**，放在weights文件夹下。

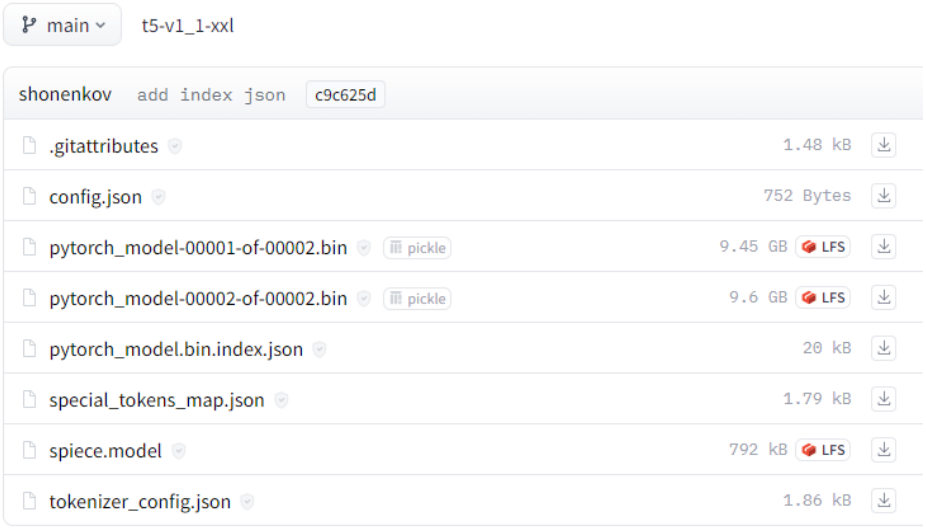
   下载链接：https://huggingface.co/stabilityai/sd-vae-ft-ema/tree/main



4. 下载**text_encoder**权重，放在weights_t5文件夹下。

   下载链接：**https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**，手动下载如**图4-52**所示文件，并放到weights_t5文件夹下

图 4-52 Huggingface 中 t5-v1_1-xxl 模型目录内容



Step9 启动训练服务

在/home/ma-user/Open-Sora-Plan1.0/目录下进行操作

训练至少需要单机8卡。

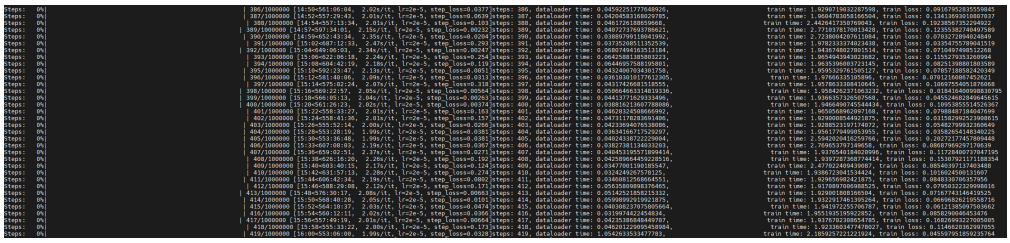
1. 命令启动训练脚本。

   例如：训练65帧的视频，拼接4张图片，则执行如下命令：

```
bash train_videoae_65x512x512.sh
```

   正常训练过程如下图所示。训练完成后，关注loss值，loss曲线收敛，记录总耗时和单步耗时。训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

图 4-53 正常训练过程



训练完成后权重保存在自动生成的目录，例如：t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5/epoch1-global_step2000/checkpoint-2000/model。

Step10 推理部署

- **使用官方权重文件推理**

  官方权重下载链接：https://huggingface.co/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main

  创建文件夹存放官方权重文件，

  ```
  mkdir weights_inference
  cd weights_inference
  mkdir vae
  mkdir 65x512x512
  ```

  以65x512x512为例：

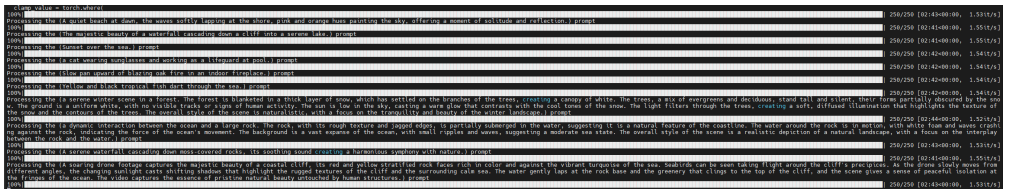
  1，进入链接https://huggingface.co/LanguageBind/Open-Sora-Plan-v1.0.0/tree/main

  2，手动下载65x512x512目录下的权重文件diffusion_pytorch_model.safetensors和配置文件config.json，并放到weights_inference/65x512x512目录下

  3，手动下载vae目录下的权重文件diffusion_pytorch_model.safetensors和配置文件config.json，并放到weights_inference/vae目录下。

  4，执行如下命令使用官方权重推理。

  ```
  bash sample_video_65.sh
  ```



- **使用训练生成的权重文件推理**

  在**Step7 启动训练服务**完成后，会在工作目录/home/ma-user/Open-Sora-Plan1.0/下自动生成一个t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5文件夹，训练后生成的权重文件存放在t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5文件夹中，例如t2v-f17-256-img4-videovae488-bf16-ckpt-xformers-bs4-lr2e-5-t5/010-F16S3-STDiT-XL-2/checkpoint-2000/model。

  修改推理配置文件中的权重文件sample_video_65.sh中的路径参数：--model_path，并执行如下推理命令。

  ```
  bash sample_video_65.sh
  ```

4.8 Open-Sora1.2 基于 DevServer 适配 PyTorch NPU 训练推理指导（6.3.907）

本文档主要介绍如何在ModelArts Lite DevServer上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora 1.2 训练和推理。

方案概览

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

适配的Cann版本是cann_8.0.rc2。

约束限制

- 本方案目前仅适用于企业客户。
- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表4-16**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

软件配套版本

表 4-16 获取软件

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907软件包中的AscendCloud-AIGC-6.3.907-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。

镜像版本

本教程中用到基础镜像地址和配套版本关系如下表所示，请提前了解。

表 4-17 基础容器镜像地址

配套软件版本	镜像用途	镜像地址	配套	获取方式
6.3.907版本	基础镜像	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	cann_8.0.rc2 pytorch_2.1.0 驱动23.0.6	从SWR拉取

说明

不同软件版本对应的基础镜像地址不同，请严格按照软件版本和镜像配套关系获取基础镜像。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 下载依赖代码包并上传到宿主机

下载华为侧插件代码包AscendCloud-AIGC-6.3.907-xxx.zip文件，获取路径参见**表4-16**。本案例使用的是解压到子目录 multimodal_algorithm/OpenSora1.2/ 目录下的所有文件，将该目录上传到宿主机上。

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像Open-Sora 1.2:1.0。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见表4-17。

```
FROM {image_url}

COPY --chown=ma-user:ma-group OpenSora1.2/* /home/ma-user/OpenSora1.2/

RUN cd /home/ma-user/OpenSora1.2 && bash prepare.sh

WORKDIR /home/ma-user/OpenSora1.2
```

构建自定义镜像OpenSora1.2:1.0。

```
docker build -t OpenSora1.2:1.0 .
```

Step4 启动镜像

启动容器镜像，推理只需要启动单卡，启动前可以根据实际需要增加修改参数。

```
docker run -itd --name ${container_name} -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 300g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge OpenSora1.2:1.0 bash
```

参数说明：

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- --device=/dev/davinci0：挂载NPU设备，该推理示例中挂载了1张卡davinci0。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

Step5 进入容器

通过容器名称进入容器中。默认使用ma-user用户执行后续命令。

```
docker exec -it ${container_name} bash
```

Step6 下载权重文件

建议手动下载所需的权重文件，在/home/ma-user/Open-Sora-Plan1.0/目录下进行操作。

1. 创建文件夹存放不同的权重文件。
   ```
   mkdir weights
   ```
2. 下载 OpenSora-VAE-v1.2权重，将下载好的权重放在 ./weights 目录下。

   OpenSora-VAE-v1.2 官网下载地址：**https://huggingface.co/hpcai-tech/OpenSora-VAE-v1.2/tree/main**
3. 下载 OpenSora-STDiT-v3权重，将下载好的权重放在 ./weights 目录下。

   OpenSora-STDiT-v3 官网下载地址：**https://huggingface.co/hpcai-tech/OpenSora-STDiT-v3/tree/main**
4. 下载 t5-v1_1-xxl 权重，将下载好的权重放在 ./weights 目录下。

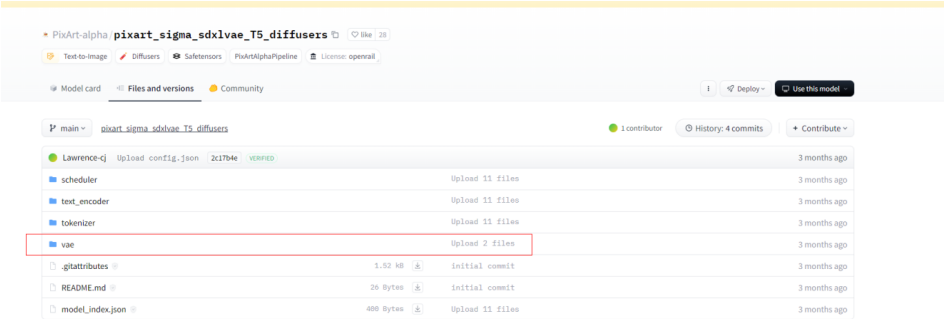
   t5-v1_1-xxl 官网下载地址： **https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**

5. 下载 pixart_sigma_sdxlvae_T5_diffusers的vae权重，将下载好的权重放在 ./weights 目录下。

   pixart_sigma_sdxlvae_T5_diffusers 官网下载地址：**https://huggingface.co/PixArt-alpha/pixart_sigma_sdxlvae_T5_diffusers/tree/main**

   下载下图中vae文件夹的内容。

   图 4-54 下载 vae 文件夹的内容

6. 下载vgg权重，将下载好的权重放在 ./weights 目录下。

   vgg16-397923af.pth 官网下载地址： **https://download.pytorch.org/models/vgg16-397923af.pth**

   vgg.pth 官网下载地址：**https://heibox.uni-heidelberg.de/f/607503859c864bc1b30b/?dl=1**

   将权重 vgg16-397923af.pth 放在 /home/ma-user/.cache/torch/hub/checkpoints/ 下，这个文件夹需要自己创建。

   ```
   cp weights/vgg16-397923af.pth /home/ma-user/.cache/torch/hub/checkpoints/vgg16-397923af.pth
   ```

   下载完成的权重文件如下图所示：

   图 4-55 下载完成的权重文件

   ```
   [root@devserver-bms-1ba73d0f-100572774 open-sora-1.2]# ls weights/
   OpenSora-STDiT-v3  OpenSora-VAE-v1.2  pixart_sigma_sdxlvae_T5_diffusers  t5-v1_1-xxl  vgg16-397923af.pth  vgg.pth
   ```

Step7 下载数据集

下载原始数据集。

```
mkdir datasets && cd datasets && mkdir webvid
wget https://anon-datasets.s3.amazonaws.com/results_2M_val.csv
python download_datasets.py
cd ..
```

download_datasets.py的内容：

```
import os
import pandas as pd

for idx, row in pd.read_csv('results_2M_val.csv').iterrows():
    os.system(f"wget -O './datasets/webvid/{row['videoid']}.mp4' --no-check-certificate {row['contentUrl']}")
```

预处理数据。

```
python -m tools.datasets.convert video ./datasets/webvid --output ./datasets/meta.csv
python -m tools.datasets.datautil ./datasets/meta.csv --info --fmin 1
python merge_data.py
```

merge_data.py的内容。

```
import pandas as pd

merged_df = pd.merge(pd.read_csv('./datasets/webvid_meta_info_fmin1.csv'), pd.read_csv('./datasets/
results_2M_val.csv')[['videoid', 'name']], left_on='id', right_on='videoid', how='left')
merged_df.to_csv('./datasets/merged.csv', index=False)
```

Step8 VAE 训练

vae训练分为3个阶段，后两次训练根据其前一次训练的结果继续训练。

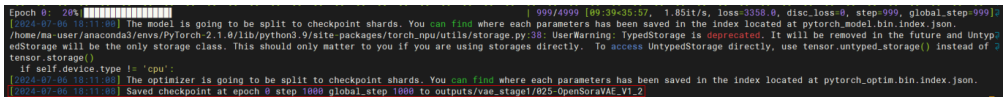
1. 第一阶段训练

   ```
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage1.py --data-path ./
   datasets/webvid_meta_data.csv
   ```

   训练完后的权重文件保存在 ./output/vae_stage1 文件夹下（例如 ./outputs/vae_stage1/000-OpenSoraVAE_V1_2/epoch0-global_step1000/model）

   具体位置打印在日志中：

   图 4-56 VAE 第一阶段训练日志

2. 第二阶段训练

   ```
   export pretrain_path = "上阶段训练的权重，例如./outputs/vae_stage1/000-OpenSoraVAE_V1_2/epoch0-
   global_step1000/model"
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage2.py --data-path ./
   datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
   ```

   训练完后的权重文件保存在 ./output/vae_stage2 文件夹下（例如 ./outputs/vae_stage2/000-OpenSoraVAE_V1_2/epoch0-global_step1000/model）

3. 第三阶段训练

   ```
   export pretrain_path = "上阶段训练的权重，例如./outputs/vae_stage2/000-OpenSoraVAE_V1_2/epoch0-
   global_step1000/model"
   torchrun --nnodes=1 --nproc_per_node=8 train_vae.py configs/vae/train/stage3.py --data-path ./
   datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
   ```

   训练完后的权重文件保存在 ./output/vae_stage3 文件夹下（例如 ./outputs/vae_stage3/000-OpenSoraVAE_V1_2/epoch0-global_step1000/model）

Step9 DIT 训练

dit训练分为3个阶段，后两次训练根据其前一次训练的结果继续训练。

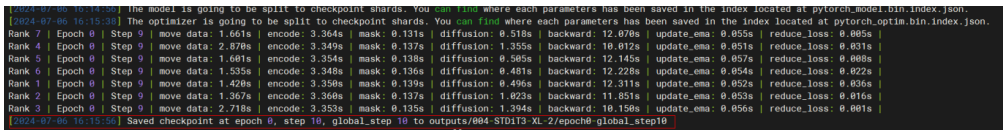
1. 第一阶段训练

   ```
   torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage1.py --data-
   path ./datasets/webvid_meta_data.csv
   ```

   训练完后的权重文件保存在 ./output 文件夹下（例如 ./outputs/000-STDiT3-XL-2/epoch0-global_step200/model）。

   具体位置打印在日志中：

   图 4-57 DIT 第一阶段训练日志

训练完成后在目录底下生成loss.txt文件(例如./outputs/000-STDiT3-XL-2/epoch0-global_step200/model)记录每一步的loss。

2. 第二阶段训练

```
export pretrain_path = "上阶段训练的权重，例如./outputs/000-STDiT3-XL-2/epoch0-global_step200/model"
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage2.py --data-path ./datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
```

训练完后的权重文件保存在 ./output 文件夹下（例如 ./outputs/001-STDiT3-XL-2/epoch0-global_step200/model），具体位置打印在日志中。

3. 第三阶段训练

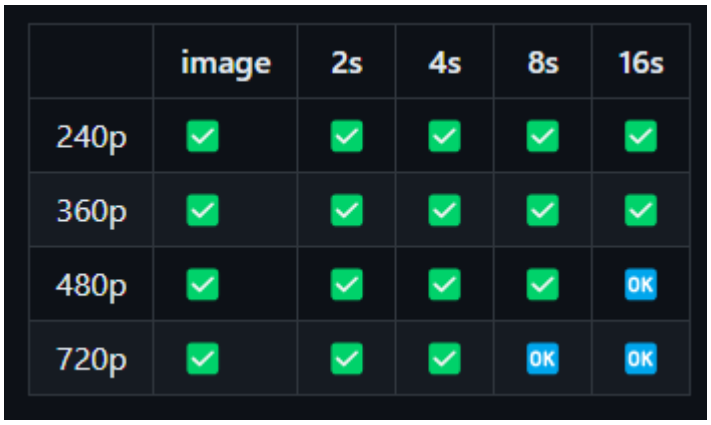
```
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage1.py --data-path ./datasets/webvid_meta_data.csv
export pretrain_path = "上阶段训练的权重，例如 ./outputs/001-STDiT3-XL-2/epoch0-global_step200/model"
torchrun --standalone --nproc_per_node 8 train.py configs/opensora-v1-2/train/stage3.py --data-path ./datasets/webvid_meta_data.csv --ckpt-path $pretrain_path
```

训练完后的权重文件保存在 ./output 文件夹下（例如 ./outputs/002-STDiT3-XL-2/epoch0-global_step200/model），具体位置打印在日志中。

Step10 推理

对与大尺寸、长时间的视频强制需要多卡推理，具体要求见下图，绿色允许只用单卡推理，蓝色至少双卡推理。

图 4-58 推理视频要求



单卡推理

```
python inference.py configs/opensora-v1-2/inference/sample.py --num-frames 4s --resolution 720p --aspect-ratio 9:16 --prompt "a beautiful waterfall"
```

多卡推理

```
torchrun --nproc_per_node 2 scripts/inference.py configs/opensora-v1-2/inference/sample.py --num-frames 16s --resolution 720p --aspect-ratio 9:16 --prompt "a beautiful waterfall"
```

最终结果保存在samples/samples/sample_0000.mp4。

4.9 SDXL&SD1.5 ComfyUI 插件基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906）

ComfyUI是一款基于节点工作流的Stable Diffusion操作界面。通过将Stable Diffusion的流程巧妙分解成各个节点，成功实现了工作流的精确定制和可靠复现。每一个节点

都有特定的功能，可以通过调整节点连接达到不同的出图效果。在图像生成方面，它不仅比传统的WebUI更迅速，而且显存占用更为经济。

本文档主要介绍如何在ModelArts Lite的DevServer环境中部署ComfyUI，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署**ComfyUI**用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 4-18 环境要求

名称	版本
driver	23.0.5
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-19 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**表4-19**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

- 确保容器可以访问公网。

Step1 准备环境

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

   说明

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

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表4-19**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

   ```
   export work_dir="自定义挂载的工作目录"
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="镜像名称或ID"
   // 启动一个容器去运行镜像
   docker run  -itd --net=bridge \
      -p 8080:8080 \
      --device=/dev/davinci6 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      --shm-size=32g \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      -v /var/log/npu/:/usr/slog \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} \
      /bin/bash
   ```

   参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -p 8080:8080：开启一个端口，可以web访问（如冲突，可自行更换其他端口）。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step4 下载并安装软件

1. 从github下载ComfyUI代码并安装依赖。

```
cd /home/ma-user
git clone https://github.com/comfyanonymous/ComfyUI.git
cd ComfyUI
git reset --hard 831511a1eecbe271e302f2f2053f285f00614180
pip install -r requirements.txt
```

如果出现报错SSL certificate problem: self signed certificate in certificate chain

图 4-59 报错 SSL certificate problem

可采取忽略SSL证书验证：使用以下命令来克隆仓库，它将忽略SSL证书验证。

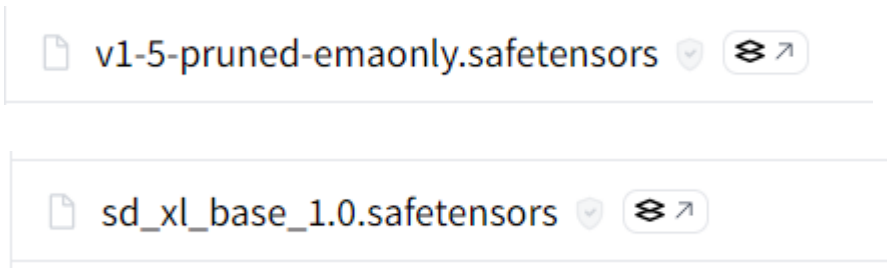
```
git clone -c http.sslVerify=false https://github.com/comfyanonymous/ComfyUI.git
```

说明

此处根据ComfyUI官网描述进行配置。

2. 下载SD模型并安装。部署好ComfyUI环境和依赖后，还需要将模型放到对应位置。
   a. 下载模型，模型下载地址：**SD1.5模型地址** ，**SDXL模型下载地址**。根据需要下载对应的模型，如下图，并将模型上传到容器内自定义挂载的工作目录，容器默认使用ma-user用户，请注意修改文件访问权限。

图 4-60 模型列表



b. 将模型复制到/home/ma-user/ComfyUI/models/checkpoints目录下。

3. 将获取到的ComfyUI插件AscendCloud-AIGC-6.3.906-xxx.zip文件上传到容器的/home/ma-user/ComfyUI/custom_nodes目录下，并解压。获取路径参见表4-19。

```
cd /home/ma-user/ComfyUI/custom_nodes/
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
mv AscendCloud/aigc_inference/torch_npu/comfyui/831511a1eecbe271/comfyui_ascend_node ./
rm -rf AscendCloud*
```

4. 使用容器IP启动服务。

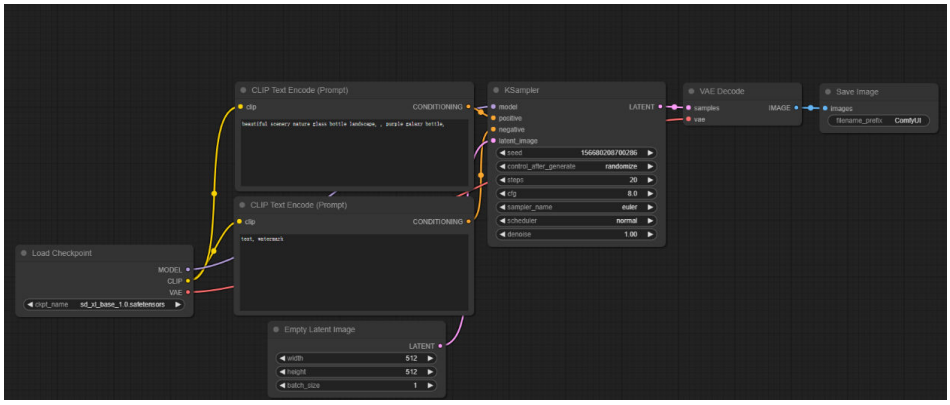
```
cd /home/ma-user/ComfyUI
python main.py --port 8080 --listen ${docker_ip} --force-fp16
```

${docker_ip}替换为容器实际的IP地址。可以在宿主机上通过docker inspect容器ID |grep IPAddress命令查询。

Step5 服务调用

1. 在浏览器中输入http://ip:8080访问界面，页面如下图。

图 4-61 访问界面

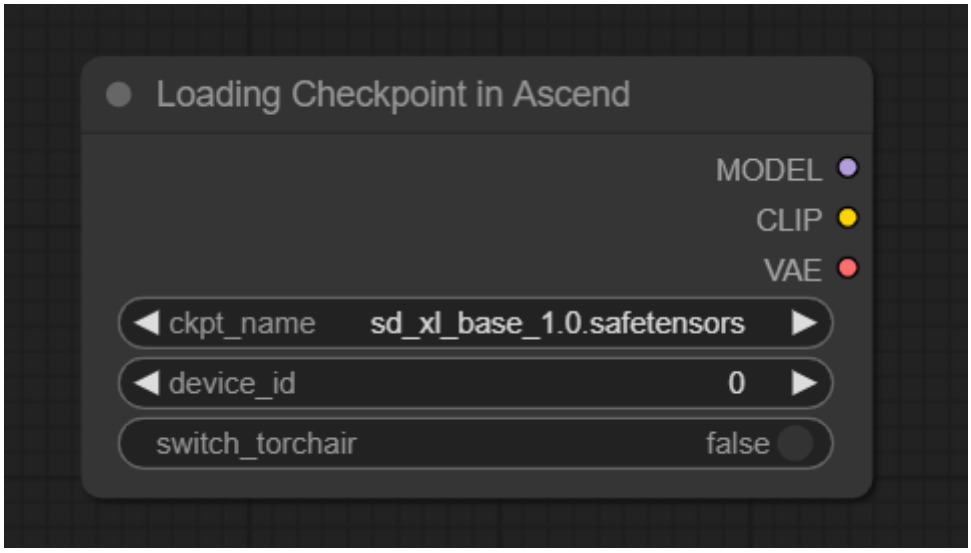


2. 加载Ascend插件节点来替换原节点。双击访问页面，并搜索“Ascend”，单击“AscendNode”，如下图。

图 4-62 搜索 Ascend

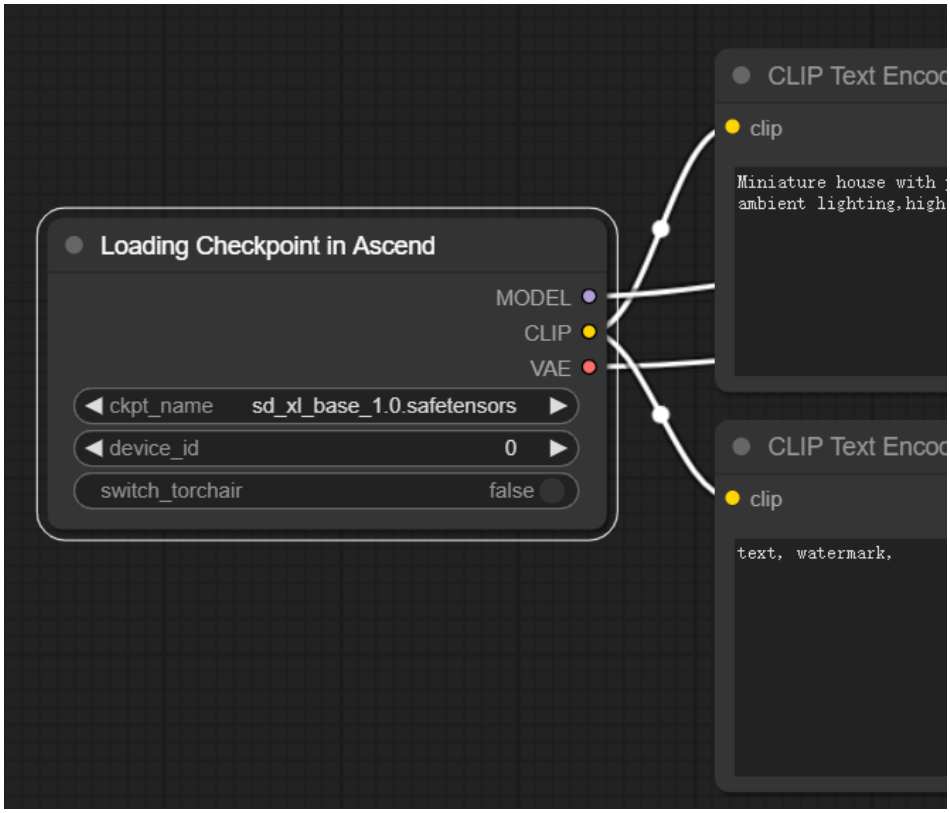
会得到一个新的关于NPU的checkpoint，如下图。

图 4-63 NPU 的 checkpoint



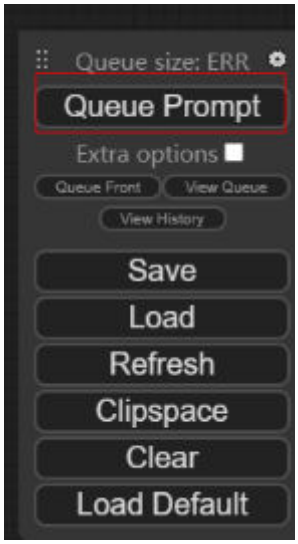
3. 根据上面checkpoint的箭头，对新的NPU的checkpoint进行规划，如下图所示。

图 4-64 规划 checkpoint



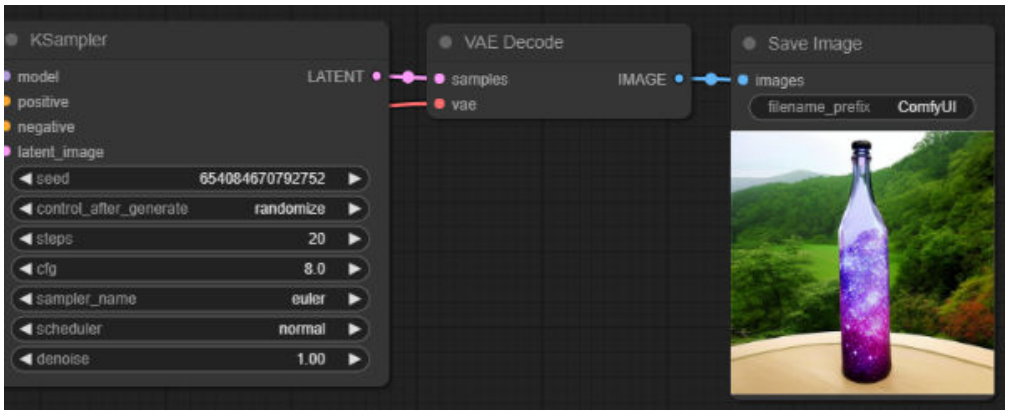
4. 在ckpt_name中选择要使用的权重文件，device_id为要使用的NPU卡号，单击“Queue Prompt”加入推理队列进行推理，如下图。

图 4-65 加入推理队列



成功之后结果如下图。

图 4-66 推理成功

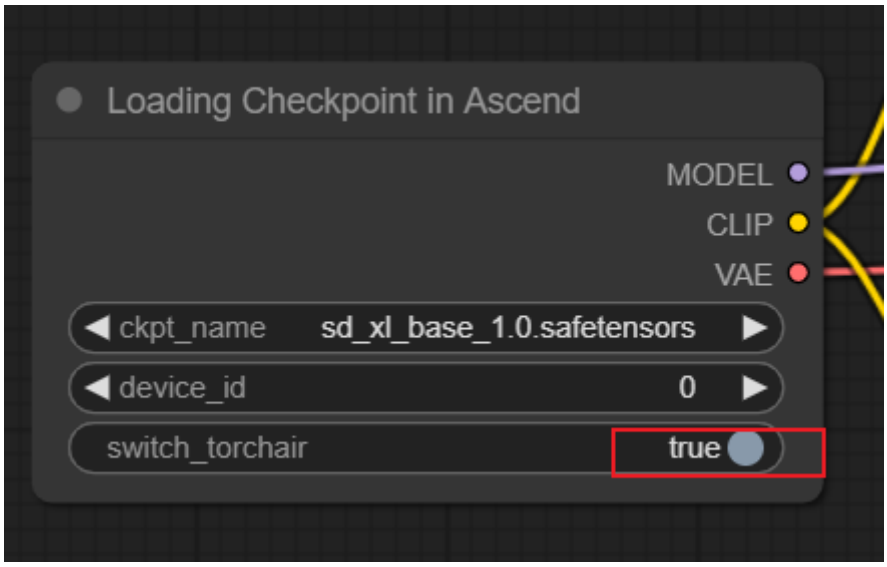


首次加载或切换模型进行推理时，需要加载模型并进行相关的初始化工作，首次推理时间较长，请耐心等待。

Step6 使用图模式功能（可选）

1. 将Ascend节点开启switch_torchair，设置值为true。

图 4-67 图模式开关



2. 按**Step5 服务调用**中步骤4正常推理即可，由于图模式编译过程耗时久，请耐心等待。SD1.5预估编译约10分钟，SDXL预估编译约30分钟。

   图模式编译过程会固定图尺寸，因此不同尺寸都需要进行一次编译，切换模型会重新进行编译。

4.10 SDXL&SD1.5 ComfyUI 基于 Lite Cluster 适配 NPU 推理指导（6.3.906）

ComfyUI是一款基于节点工作流的Stable Diffusion操作界面。通过将Stable Diffusion的流程巧妙分解成各个节点，成功实现了工作流的精确定制和可靠复现。每一个节点都有特定的功能，可以通过调整节点连接达到不同的出图效果。在图像生成方面，它不仅比传统的WebUI更迅速，而且显存占用更为经济。

本文档主要介绍如何在ModelArts Lite的Cluster环境中部署ComfyUI，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的Lite Cluster上使用昇腾计算资源部署**ComfyUI**用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Cluster资源。

本方案目前仅适用于企业客户，并且需要用户具备k8s集群相关技能。

资源规格要求

推荐使用“西南-贵阳一”Region上的Cluster资源

表 4-20 环境要求

名称	版本
CANN	cann_8.0.rc2

名称	版本
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-21 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

请参考**表4-21**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

本方案使用需要用户具备k8s集群相关技能。

Step1 准备环境

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

   说明

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

2. 配置Cluster资源，确保可以通过公网访问Cluster机器，具体配置请参见**配置Lite Cluster网络**。

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

4. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表4-21**。

```
docker pull {image_url}
```

Step3 下载并安装软件

1. 在宿主机上创建目录/root/comfyui，将下面步骤中所有的文件放到/root/comfyui目录下。
2. 下载模型，模型下载地址：**SD1.5模型地址** ，**SDXL下载地址**。根据自己的需要下载对应的模型。
3. 将获取到的ComfyUI插件AscendCloud-AIGC-6.3.906-xxx.zip文件上传到/root/comfyui，并解压。获取路径参见**表4-21**。
   ```
   unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
   mv AscendCloud/aigc_inference/torch_npu/comfyui/831511a1eecbe271/comfyui_ascend_node ./
   rm -rf AscendCloud*
   ```
4. 编写dockerfile
   ```
   FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

   RUN cd /home/ma-user && git clone https://github.com/comfyanonymous/ComfyUI.git -c http.sslVerify=false && cd ComfyUI/ && git reset --hard 831511a1eecbe271e302f2f2053f285f00614180 && pip install -r requirements.txt

   COPY --chown=ma-user:ma-group v1-5-pruned-emaonly.safetensors /home/ma-user/ComfyUI/models/checkpoints
   COPY --chown=ma-user:ma-group sd_xl_base_1.0.safetensors /home/ma-user/ComfyUI/models/checkpoints
   COPY --chown=ma-user:ma-group comfyui_ascend_node /home/ma-user/ComfyUI/custom_nodes/comfyui_ascend_node

   ENTRYPOINT cd /home/ma-user/ComfyUI && source /usr/local/Ascend/ascend-toolkit/set_env.sh && python main.py --port 30027 --listen 0.0.0.0 --force-fp16
   ```
5. 基于dockerfile进行build
   ```
   docker build -t comfyui:v1 .
   ```

Step4 上传镜像到容器镜像服务

参考**pull/push 镜像体验**章节，将上一步build的镜像上传到容器镜像服务上。

Step5 使用 CCE 进行部署

在CCE上**创建工作负载**，创建工作负载时所需的yaml文件可参考**在Lite Cluster资源池上使用Snt9B完成推理任务**。

在CCE上**创建服务**。

4.11 SDXL&SD1.5 WebUI 基于 Lite Cluster 适配 NPU 推理指导（6.3.906）

本文档主要介绍如何在ModelArts Lite的Cluster环境中部署Stable Diffusion的WebUI套件，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的Lite Cluster上使用昇腾计算资源部署Stable Diffusion WebUI套件用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买Cluster资源。

本方案目前仅适用于企业客户，并且需要用户具备k8s集群相关技能。

资源规格要求

推理部署推荐使用“西南-贵阳一”Region上的Cluster资源。

获取软件

获取插件代码包AscendCloud-6.3.906-xxx.zip中的AscendCloud-AIGC-6.3.906-xxx.zip文件。获取路径：**Support-E**。

说明

如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。

代码包文件名中的xxx表示具体的时间戳，以包名的实际时间为准。

Step1 准备环境

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

   说明

   购买Cluster资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 配置Cluster资源，确保可以通过公网访问Cluster机器，具体配置请参见**配置Lite Cluster网络**。

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

4. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

5. 获取基础镜像。建议使用官方提供的镜像部署推理服务。

   镜像地址{image_url}为：

   西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580

   ```
   docker pull {image_url}
   ```

Step2 下载软件包

1. 在宿主机上创建目录/root/webui，将下面步骤中所有的文件放到/root/webui目录下。
2. 下载SD基础模型，SD基础模型的官网下载地址。

   https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors

   https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.safetensors
3. 根据需要下载controlnet模型。

   controlnet模型官网下载地址：

   https://huggingface.co/lllyasviel/ControlNet-v1-1/tree/main

   https://huggingface.co/lllyasviel/sd_control_collection/tree/main

   例：选择下载sd1.5 canny：

   https://huggingface.co/lllyasviel/ControlNet-v1-1/blob/main/control_v11p_sd15_canny.pth

   https://huggingface.co/lllyasviel/ControlNet-v1-1/blob/main/control_v11p_sd15_canny.yaml

   例：选择下载sdxl canny：

   https://huggingface.co/lllyasviel/sd_control_collection/blob/main/diffusers_xl_canny_mid.safetensors
4. 下载safety-checker模型包。

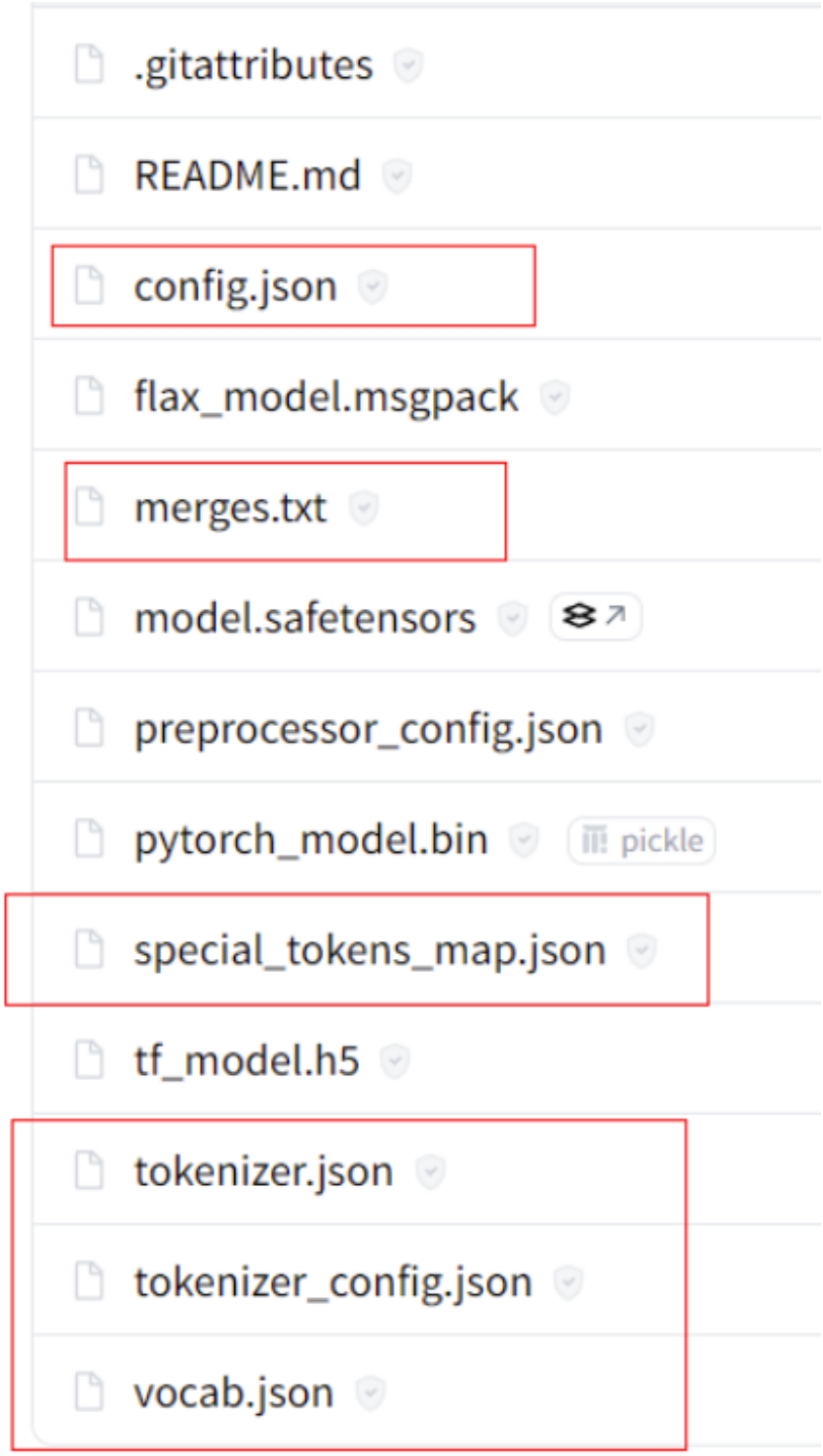
   safety-checker的官网下载地址：**https://huggingface.co/CompVis/stable-diffusion-safety-checker/tree/main**

   在/root/webui目录下创建CompVis目录，然后下载所有文件
5. 下载vaeapprox-sdxl.pt。

   vaeapprox-sdxl.pt的官网下载地址：**https://github.com/AUTOMATIC1111/stable-diffusion-webui/releases/tag/v1.0.0-pre**。
6. 下载clip-vit-large-patch14。

   在/root/webui目录下创建clip-vit-large-patch14目录，然后下载下图红框中的文件。clip-vit-large-patch14官网下载地址：**openai/clip-vit-large-patch14 at main (huggingface.co)**。

图 4-68 下载 clip-vit-large-patch14 文件

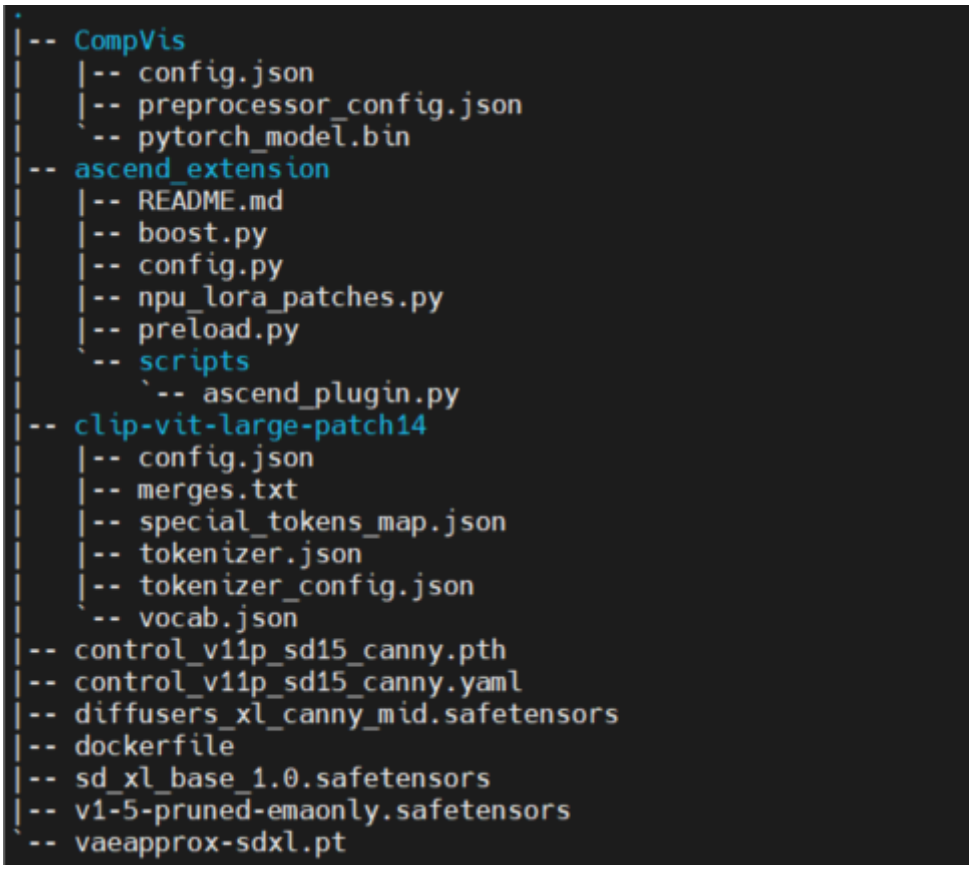


7. 将获取到的WebUI插件AscendCloud-AIGC-6.3.906-xxx.zip文件上传到/root/webui，并解压。获取路径参见**获取软件**。

```
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
mv AscendCloud/aigc_inference/torch_npu/webui/v1_9_0_RC/ascend_extension ./
rm -rf AscendCloud*
```

8. 最终/root/webui下的目录应该如下。

图 4-69 /root/webui 下的目录文件



Step3 构建 dockerfile

1. 编写dockerfile。

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580
RUN git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui.git -c http.sslVerify=false \
  && cd stable-diffusion-webui && git checkout -b v1.9.0-RC \
  && cd /home/ma-user/stable-diffusion-webui/extensions/ && git clone https://github.com/Mikubill/sd-webui-controlnet.git -c http.sslVerify=false \
  && cd /home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/ && git reset --hard 3b4eedd90fe8ebcac5363f586157d36dcd9a513f \
  && mkdir -p /home/ma-user/.cache/huggingface/hub/models--openai--clip-vit-large-patch14/refs \
  && mkdir -p /home/ma-user/.cache/huggingface/hub/models--openai--clip-vit-large-patch14/snapshots \
  && printf "%s" "32bd64288804d66eefd0ccbe215aa642df71cc41" > /home/ma-user/.cache/huggingface/hub/models--openai--clip-vit-large-patch14/refs/main
COPY --chown=ma-user:ma-group v1-5-pruned-emaonly.safetensors /home/ma-user/stable-diffusion-webui/models/Stable-diffusion/v1-5-pruned-emaonly.safetensors
COPY --chown=ma-user:ma-group sd_xl_base_1.0.safetensors /home/ma-user/stable-diffusion-webui/models/Stable-diffusion/sd_xl_base_1.0.safetensors
COPY --chown=ma-user:ma-group control_v11p_sd15_canny.pth /home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/control_v11p_sd15_canny.pth
COPY --chown=ma-user:ma-group control_v11p_sd15_canny.yaml /home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/control_v11p_sd15_canny.yaml
COPY --chown=ma-user:ma-group diffusers_xl_canny_mid.safetensors /home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/diffusers_xl_canny_mid.safetensors
COPY --chown=ma-user:ma-group ascend_extension  /home/ma-user/stable-diffusion-webui/extensions/ascend_extension
COPY --chown=ma-user:ma-group vaeapprox-sdxl.pt /home/ma-user/stable-diffusion-webui/models/VAE-approx/vaeapprox-sdxl.pt
COPY --chown=ma-user:ma-group CompVis /home/ma-user/stable-diffusion-webui/CompVis
COPY --chown=ma-user:ma-group clip-vit-large-patch14 /home/ma-user/.cache/huggingface/hub/
```

```
models--openai--clip-vit-large-patch14/snapshots/32bd64288804d66eefd0ccbe215aa642df71cc41
RUN cd /home/ma-user/stable-diffusion-webui && pip install --upgrade pip && pip install -r
requirements.txt --no-deps \
  && pip install lightning_utilities torchmetrics gradio_client matplotlib pydantic aiofiles starlette
ffmpy pydub uvicorn orjson semantic_version pydantic aiofiles starlette ffmpy pydub uvicorn orjson
semantic_version gitdb trampoline clip aenum facexlib torch==2.1.0 python-multipart gdown \
  && pip install -r requirements_versions.txt && pip install httpx==0.24.1 && pip install diffusers \
  && mkdir repositories && cd /home/ma-user/stable-diffusion-webui/repositories/ \
  && git clone https://github.com/Stability-AI/stablediffusion.git -c http.sslVerify=false && mv
stablediffusion/ stable-diffusion-stability-ai \
  && git clone https://github.com/Stability-AI/generative-models.git -c http.sslVerify=false \
  && git clone https://github.com/Stability-AI/k-diffusion.git -c http.sslVerify=false \
  && git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui-assets.git -c
http.sslVerify=false
ENTRYPOINT cd /home/ma-user/stable-diffusion-webui && python3 launch.py --skip-torch-cuda-test
--port 30028 --enable-insecure-extension-access --listen --log-startup --disable-safe-unpickle --skip-
prepare-environment --api
```

2. 基于dockerfile进行build

```
docker build -t webui:v1 .
```

Step4 上传镜像到容器镜像服务

参考**pull/push 镜像体验**章节，将上一步build的镜像上传到容器镜像服务上。

Step5 使用 CCE 进行部署

在CCE上**创建工作负载**，创建工作负载时所需的yaml文件可参考**在Lite Cluster资源池上使用Snt9B完成推理任务**。

在CCE上**创建服务**。

4.12 LLaVA 模型基于 DevServer 适配 PyTorch NPU 预训练指导（6.3.906）

LLaVA是一种新颖的端到端训练的大型多模态模型，它结合了视觉编码器和Vicuna，用于通用的视觉和语言理解，实现了令人印象深刻的聊天能力，在科学问答（Science QA）上达到了新的高度。

本文档主要介绍如何利用ModelArts Lite DevServer，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成LLaVA模型训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。训练至少需要单机8卡，推理需要单机单卡。

表 4-22 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0
驱动	23.0.5

获取软件和镜像

表 4-23 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练至少需要单机8卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取软件和镜像**。
   ```
   docker pull {image_url}
   ```
2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练默认使用单机8卡。
   ```
   docker run -itd --net=host \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_id}  \
   /bin/bash
   ```

   参数说明：

   - device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
   - ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
   - shm-size：共享内存大小。
   - ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
   - ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

   说明

   - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
   - driver及npu-smi需同时挂载至容器。
   - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。
3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。
   ```
   docker exec -it ${container_name} bash
   ```

Step3 获取代码并上传

上传代码AscendCloud-AIGC-6.3.906-xxx.zip到容器的工作目录中，包获取路径请参见**获取软件和镜像**。

上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

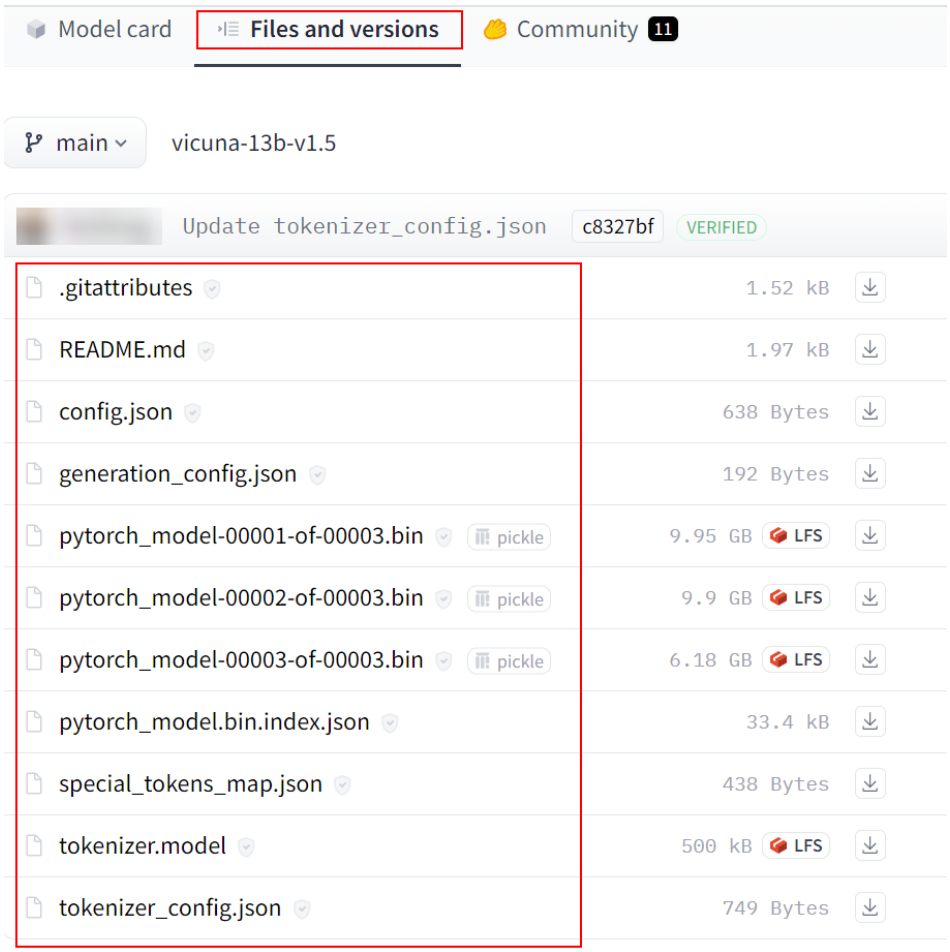
Step4 准备训练环境

1. 获取LLaVA模型代码。

```
cd ${container_work_dir}
unzip AscendCloud-6.3.906-xxx.zip
unzip AscendCloud-AIGC-6.3.906-xxx.zip
cd multimodal_algorithm/LLAVA/llava-train/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd
bash build.sh
cd LLaVA
mkdir ./playground/data/LLaVA-Pretrain
```

2. 下载vicuna-13b-v1.5模型。下载地址：**lmsys/vicuna-13b-v1.5 · Hugging Face**

图 4-70 下载 vicuna-13b-v1.5 模型



Step4 下载数据集

请用户自行下载GQA数据集，下载地址：**images**。

将GQA数据集放于${container_work_dir}/multimodal_algorithm/LLAVA/llava-train/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA/playground/data/LLaVA-Pretrain目录下。

下载**blip_laion_cc_sbu_558k.json**文件，并放于${container_work_dir}/multimodal_algorithm/LLAVA/llava-train/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA/playground/data/LLaVA-Pretrain目录下。

Step5 开始训练

进入解压后的源码包根目录。

```
cd ${container_work_dir}/multimodal_algorithm/LLAVA/llava-train/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA
```

修改训练脚本模型路径(--model_name_or_path 模型路径)。

```
vim ./scripts/v1_5/pretrain_new.sh
```

运行训练脚本，默认是单机8卡。

```
bash ./scripts/v1_5/pretrain_new.sh
```

训练完成后，权重文件保存checkpoints/llava-v1.5-13b-pretrain路径下，并输出模型训练精度和性能信息。

训练过程中，训练日志会在最后的Rank节点打印。

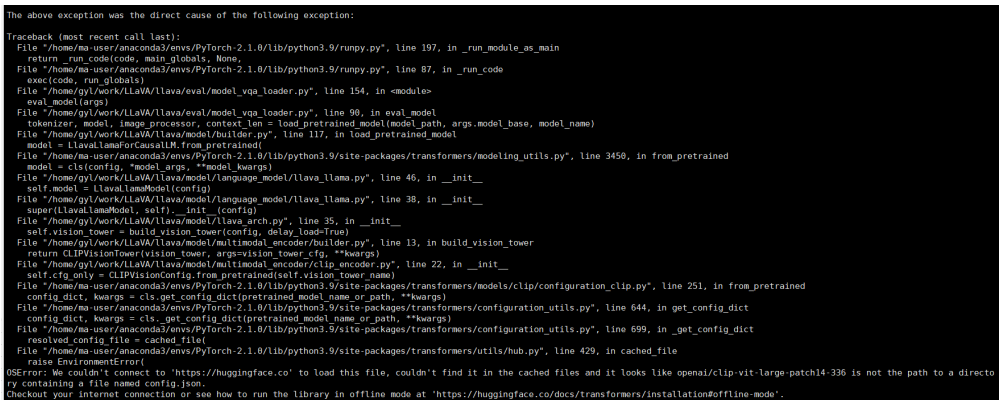
日志里存在lm loss参数 ，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

FAQ

如果clip-vit-large-patch14-336模型不能自动下载。

请手动下载（**openai/clip-vit-large-patch14-336 at main (huggingface.co)**），并在pretrain_new.sh脚本中修改--vision_tower参数。

图 4-71 提示 clip-vit-large-patch14-336 模型不能自动下载



4.13 LLaVA 模型基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906）

LLaVA是一种新颖的端到端训练的大型多模态模型，它结合了视觉编码器和Vicuna，用于通用的视觉和语言理解，实现了令人印象深刻的聊天能力，在科学问答（Science QA）上达到了新的高度。

本文档主要介绍如何利用ModelArts Lite DevServer，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成LLaVA模型推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。推理需要单机单卡。

表 4-24 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0
驱动	23.0.5

获取软件和镜像

表 4-25 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。

- 推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取软件和镜像**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

   ```
   docker run -it --net=host \
   --device=/dev/davinci0 \
   --device=/dev/davinci1 \
   --device=/dev/davinci2 \
   --device=/dev/davinci3 \
   --device=/dev/davinci4 \
   --device=/dev/davinci5 \
   --device=/dev/davinci6 \
   --device=/dev/davinci7 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_id}  \
   /bin/bash
   ```


参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

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

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step3 获取代码并上传

上传代码AscendCloud-AIGC-6.3.906-xxx.zip到容器的工作目录中，包获取路径请参见**获取软件和镜像**。

上传代码和权重到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

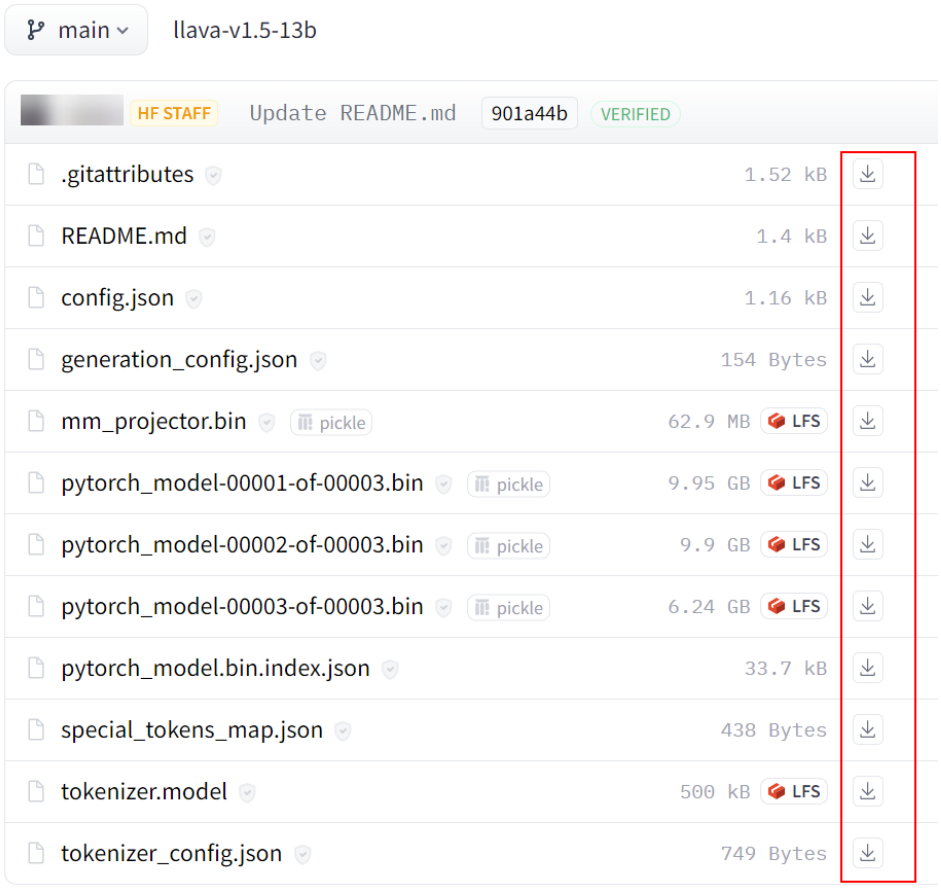
Step4 准备推理环境

1. 获取LLaVA模型代码。

```
cd ${container_work_dir}
unzip AscendCloud-6.3.906-xxx.zip
unzip AscendCloud-AIGC-6.3.906-xxx.zip
cd multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd
bash build.sh
cd LLaVA
mkdir ./playground/data/eval
```

2. 下载llava-v1.5-13b模型。下载地址：**liuhaotian/llava-v1.5-13b at main (huggingface.co)**

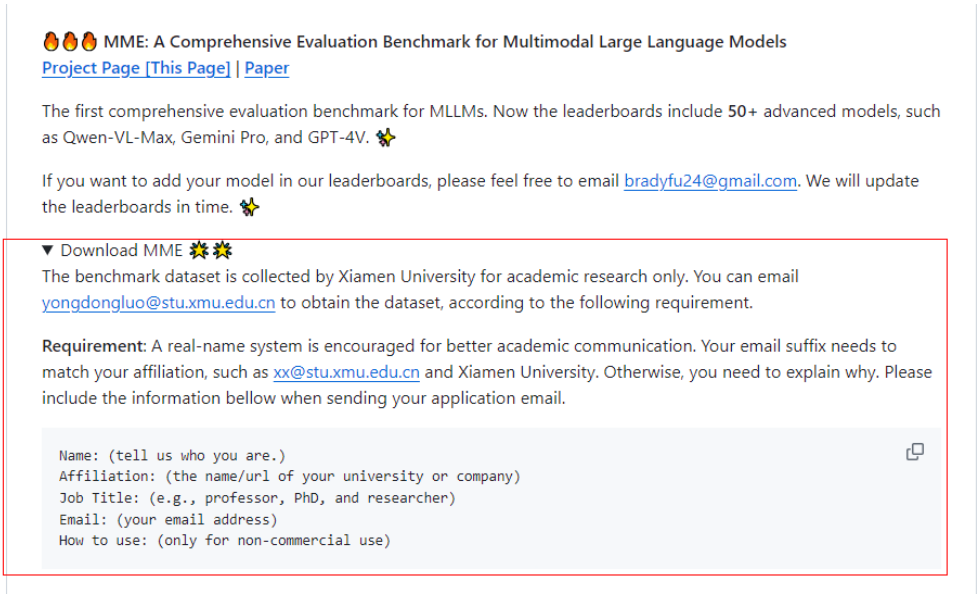
图 4-72 下载 llava-v1.5-13b 模型



Step5 下载数据集

请用户自行获取**MME评估集**，将MME评估集放于${container_work_dir}/multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA/playground/data/eval目录下。

图 4-73 MME 评估集



Step6 开始推理

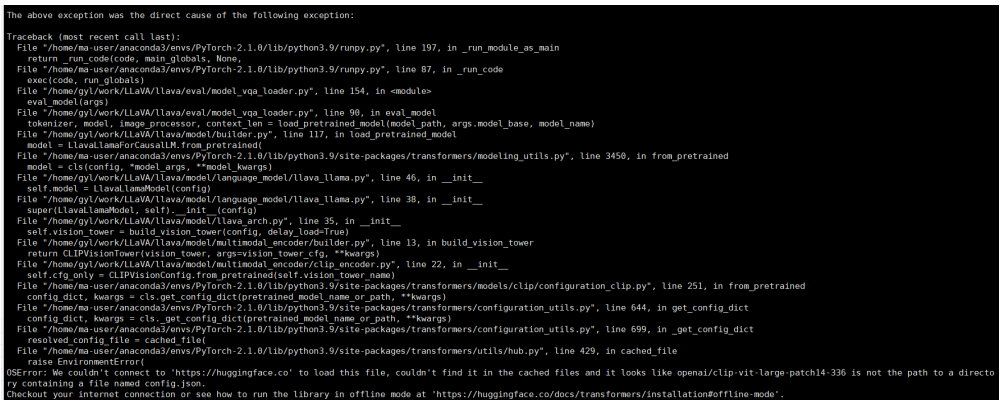
1. 进入解压后的源码包根目录。
   ```
   cd ${container_work_dir}/multimodal_algorithm/LLAVA/llava-inference/5d8f1760c08b7dfba3ae97b71cbd4c6f17d12dbd/LLaVA
   ```
2. 修改mme_8p.sh。需要将脚本里模型的路径更改为实际存放模型的路径(--model-path 模型路径)，同时检查数据集路径与实际保持一致(--question-file --image-folder --answers-file)。
   ```
   vim ./scripts/v1_5/eval/mme_8p.sh
   ```
3. 运行评估脚本。启动单卡。
   ```
   ASCEND_RT_VISIBLE_DEVICES=0 bash ./scripts/v1_5/eval/mme_8p.sh
   ```
4. 启动8卡。可支持单机八卡推理，可以减短耗时。
   ```
   ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 bash ./scripts/v1_5/eval/mme_8p.sh
   ```

FAQ

如果clip-vit-large-patch14-336模型不能自动下载。

请手动下载（**openai/clip-vit-large-patch14-336 at main (huggingface.co)**），并在llava-v1.5-13b模型下的config.json文件中修改mm_vision_tower参数中的模型路径。

图 4-74 提示 clip-vit-large-patch14-336 模型不能自动下载



4.14 Qwen-VL 基于 DevServer 适配 Pytorch NPU 的 Finetune 训练指导(6.3.906)

Qwen-VL是规模视觉语言模型，可以以图像、文本、检测框作为输入，并以文本和检测框作为输出。具有强大的性能、多语言对话、多图交错对话、支持中文开放域定位、细粒度识别和理解等特点。

本文档主要介绍如何利用训练框架PyTorch_npu + 华为自研Ascend Snt9B硬件，完成Qwen-VL Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 4-26 环境要求

名称	版本
PyTorch	pytorch_2.1.0
驱动	23.0.5

获取软件和镜像

表 4-27 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.906版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练至少需要单机8卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取软件和镜像**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练默认使用单机8卡。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci1 \
--device=/dev/davinci2 \
--device=/dev/davinci3 \
--device=/dev/davinci4 \
--device=/dev/davinci5 \
--device=/dev/davinci6 \
--device=/dev/davinci7 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=64g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

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

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户。

```
docker exec -it ${container_name} bash
```

Step3 获取代码并上传

上传代码AscendCloud-AIGC-6.3.906-xxx.zip到容器的工作目录${container_work_dir}中，包获取路径请参见**表4-27**。

Step4 准备训练环境

1. 下载权重。从HuggingFace下载**Qwen-VL-Chat**，或将您已下载的权重文件上传到容器工作目录${container_work_dir}中。

```
# 模型结构如下：
Qwen-VL-Chat/
├── config.json
├── configuration_qwen.py
├── generation_config.jsons
├── modeling_qwen.py
```



```
├── pytorch_model-00001-of-00010.bin
├── pytorch_model-00002-of-00010.bin
├── pytorch_model-00003-of-00010.bin
├── pytorch_model-00004-of-00010.bin
├── pytorch_model-00005-of-00010.bin
├── pytorch_model-00006-of-00010.bin
├── pytorch_model-00007-of-00010.bin
├── pytorch_model-00008-of-00010.bin
├── pytorch_model-00009-of-00010.bin
├── pytorch_model-00010-of-00010.bin
├── pytorch_model.bin.index.json
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
├── SimSun.ttf
├── tokenization_qwen.py
├── tokenizer_config.json
└── visual.py
```

2. 赋予容器访问权重文件的权限。上传文件到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

3. 在容器中解压代码包并执行Qwen-VL安装脚本。

```
# 解压代码包
unzip AscendCloud-AIGC-6.3.906-*.zip
rm -rf AscendCloud-AIGC-6.3.906-*
# 执行安装脚本
# model_path 配置为Qwen-VL的权重路径，例：/home/ma-user/Qwen-VL-Chat
git config --global http.sslVerify false
bash multimodal_algorithm/QwenVL/6d0ab0efd0a/qwen_vl_install.sh {model_path}
# 执行完成后，代码路径为ModelZoo-PyTorch/PyTorch/built-in/mlm/Qwen-VL
# 安装bc命令
sudo yum install -y bc
```

Step5 准备训练数据集

用户需自行制作数据集，并将数据集上传到容器的工作目录中，再赋予容器读写数据集目录的权限。

数据集制作请参考Qwen-VL**官方指导资料**，将所有数据样本放到一个列表中并存入json文件中。每个样本对应一个字典，包含id和conversation，其中后者为一个列表。数据集的json文件示例如下所示。

```
[
 {
  "id": "identity_0",
  "conversations": [
   {
    "from": "user",
    "value": "你好"
   },
   {
    "from": "assistant",
    "value": "我是Qwen-VL,一个支持视觉输入的大模型。"
   }
  ]
 },
 {
  "id": "identity_1",
  "conversations": [
   {
    "from": "user",
    "value": "Picture 1: <img>https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/demo.jpeg</
img>\n图中的狗是什么品种？"
```

```
    },
    {
      "from": "assistant",
      "value": "图中是一只拉布拉多犬。"
    },
    {
      "from": "user",
      "value": "框出图中的格子衬衫"
    },
    {
      "from": "assistant",
      "value": "<ref>格子衬衫</ref><box>(588,499),(725,789)</box>"
    }
  ]
 },
 {
  "id": "identity_2",
  "conversations": [
    {
      "from": "user",
      "value": "Picture 1: <img>assets/mm_tutorial/Chongqing.jpeg</img>\nPicture 2: <img>assets/
mm_tutorial/Beijing.jpeg</img>\n图中都是哪"
    },
    {
      "from": "assistant",
      "value": "第一张图片是重庆的城市天际线，第二张图片是北京的天际线。"
    }
  ]
 }
]
```

- 为针对多样的VL任务，特殊tokens如下：<img> </img> <ref> </ref> <box> </box>。
- 对于带图像输入的内容可表示为Picture id: <img>img_path</img>\n{your prompt}，其中id表示对话中的第几张图片。"img_path"可以是本地的图片或网络地址。
- 对话中的检测框可以表示为<box>(x1,y1),(x2,y2)</box>，其中 (x1, y1) 和(x2, y2)分别对应左上角和右下角的坐标，并且被归一化到[0, 1000)的范围内。检测框对应的文本描述也可以通过<ref>text_caption</ref>表示。

Step6 开始训练

进入代码根目录。

```
cd ModelZoo-PyTorch/PyTorch/built-in/mlm/Qwen-VL
```

运行精度训练脚本train_full_8p.sh。运行前请先修改参数。

```
bash test/train_full_8p.sh --model_name=${预训练模型路径} --data_path=${训练数据集路径} --epochs=${训练epoch数量} # 8卡精度训练，混精bf16
例： bash test/train_full_8p.sh --model_name=path/Qwen-VL-Chat  --data_path=path/xx.json --epochs=${训练epoch数量}
```

运行性能训练脚本train_performance_8p.sh。运行前请先修改参数。

```
# 运行性能训练脚本
bash test/train_performance_8p.sh --model_name=${预训练模型路径} --data_path=${训练数据集路径}  # 8卡性能，混精bf16
```

训练后的产物路径说明如下。

```
#日志路径：
ModelZoo-PyTorch/PyTorch/built-in/mlm/Qwen-VL/test/output/8p
#训练输出权重路径：
ModelZoo-PyTorch/PyTorch/built-in/mlm/Qwen-VL/output-qwen-vl
```

训练过程中，训练日志会在最后的Rank节点打印。

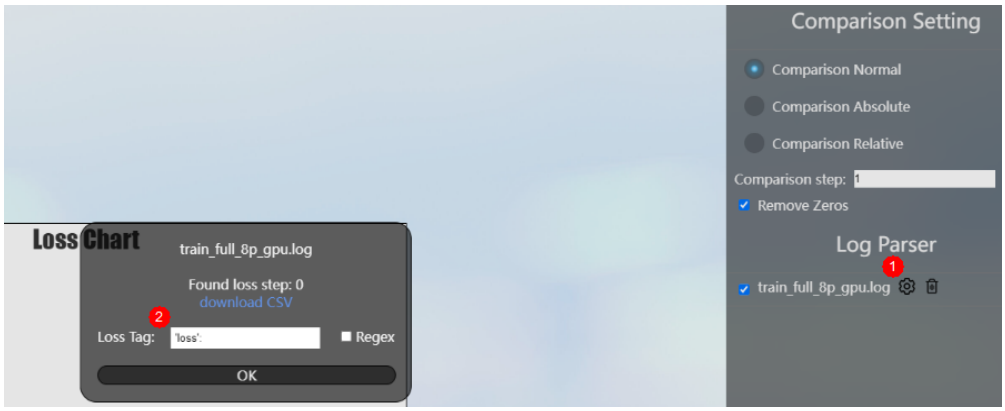
日志里存在lm loss参数，lm loss参数随着训练迭代周期持续性减小，并逐渐趋于稳定平缓。可以使用可视化工具TrainingLogParser查看loss收敛情况。

FAQ

问题：使用TrainingLogParser工具解析训练日志中loss数据，坐标栏空白，未显示数据走势曲线。

解决方法：在解析工具页面右侧，单击日志文件名右边的设置图标，在弹出的窗口中修改Loss Tag。将字符串loss加上单引号，改为'loss': ，如图4-75所示。

图 4-75 修改 Loss Tag



4.15 Qwen-VL 基于 DevServer 适配 Pytorch NPU 的推理指导(6.3.906)

Qwen-VL是规模视觉语言模型，可以以图像、文本、检测框作为输入，并以文本和检测框作为输出。具有强大的性能、多语言对话、多图交错对话、支持中文开放域定位、细粒度识别和理解等特点。

本文档主要介绍如何利用训练框架PyTorch_npu + 华为自研Ascend Snt9B硬件，完成Qwen-VL推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 4-28 环境要求

名称	版本
PyTorch	pytorch_2.1.0
驱动	23.0.5

获取镜像

表 4-29 获取镜像

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	从SWR拉取。

约束限制

- 推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**获取镜像**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。推理默认使用单机单卡。

```
docker run -itd --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=32g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- device=/dev/davinci0：挂载NPU设备，示例中挂载了1张卡davinci0。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小。
- ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

说明

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

3. 进入容器。需要将${container_name}替换为实际的容器名称。启动容器默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step3 准备推理环境

1. 在容器工作目录下载Qwen-VL模型源码。

```
git clone https://github.com/QwenLM/Qwen-VL.git
cd Qwen-VL && git checkout aa00ed04091eea5fcdd32985e7915f1c53e7d599
```

2. 安装依赖。在模型源码包根目录下执行命令，安装模型需要的依赖。

```
#安装依赖包
pip install -r requirements.txt
```

3. 下载权重。从HuggingFace下载**Qwen-VL-Chat**，或将您已下载的权重文件上传到容器工作目录${container_work_dir}中。

```
# 模型结构如下：
Qwen-VL-Chat/
├── config.json
├── configuration_qwen.py
├── generation_config.json
├── modeling_qwen.py
├── pytorch_model-00001-of-00010.bin
├── pytorch_model-00002-of-00010.bin
├── pytorch_model-00003-of-00010.bin
├── pytorch_model-00004-of-00010.bin
├── pytorch_model-00005-of-00010.bin
├── pytorch_model-00006-of-00010.bin
├── pytorch_model-00007-of-00010.bin
├── pytorch_model-00008-of-00010.bin
```

```
├── pytorch_model-00009-of-00010.bin
├── pytorch_model-00010-of-00010.bin
├── pytorch_model.bin.index.json
├── qwen_generation_utils.py
├── qwen.tiktoken
├── README.md
├── SimSun.ttf
├── tokenization_qwen.py
├── tokenizer_config.json
└── visual.py
```

4. 赋予容器访问权重文件的权限。上传代码和数据到宿主机时使用的是root用户，此处需要执行如下命令统一文件属主为ma-user用户。

```
#统一文件属主为ma-user用户
sudo chown -R ma-user:ma-group  ${container_work_dir}
# ${container_work_dir}:/home/ma-user/ws 容器内挂载的目录
#例如：sudo chown -R ma-user:ma-group  /home/ma-user/ws
```

5. 修改Qwen-VL-Chat/modeling_qwen.py。

```
# 模型下载后，需修改Qwen-VL-Chat/modeling_qwen.py  36~37行，否则推理时会报cuda错误
sed -i "s/SUPPORT_FP16 =.*/SUPPORT_FP16 = SUPPORT_CUDA/g" modeling_qwen.py
sed -i "s/SUPPORT_BF16 =.*/SUPPORT_BF16 = SUPPORT_CUDA/g" modeling_qwen.py
```

modeling_qwen.py修改前第36~37行样例如下。

```
SUPPORT_BF16 = SUPPORT_CUDA and torch.cuda.is_bf16_supported()
SUPPORT_FP16 = SUPPORT_CUDA and torch.cuda.get_device_capability(0)[0] >= 7
```

modeling_qwen.py修改后第36~37行样例如下。

```
SUPPORT_BF16 = SUPPORT_CUDA
SUPPORT_FP16 = SUPPORT_CUDA
```

Step4 开始推理

在容器工作目录下创建推理脚本文件infer.py，文件内容如下。

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers.generation import GenerationConfig
import os
os.environ['CURL_CA_BUNDLE'] = ''
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu
torch.manual_seed(1234)

# 模型权重路径按实际填写
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen-VL-Chat", trust_remote_code=True)

model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen-VL-Chat", device_map="cuda",
trust_remote_code=True, bf16=True).eval()

# 第一轮对话
query = tokenizer.from_list_format([
    {'image': 'https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/demo.jpeg'}, # Either a local
path or an url
    {'text': '这是什么?'},
])
response, history = model.chat(tokenizer, query=query, history=None)
print(response)
# 图中是一名女子在沙滩上和狗玩耍，旁边是一只拉布拉多犬，它们处于沙滩上。

# 第二轮对话
response, history = model.chat(tokenizer, '框出图中击掌的位置', history=history)
print(response)
# <ref>击掌</ref><box>(536,509),(588,602)</box>
image = tokenizer.draw_bbox_on_latest_picture(response, history)
if image:
  image.save('1.jpg')
else:
  print("no box")
```

运行推理脚本。

```
python infer.py
```

推理结果如下所示。

图 4-76 推理结果（1）

```
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/urllib3/connectionpool.py:1013: InsecureRequestWarning: Unverified HTTPS request is being made
. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/1.26.x/advanced-usage.html#ssl-warnings
  warnings.warn(
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/transformers/generation/logits_process.py:425: UserWarning: AutoNonVariableTypeMode is depreca
 in 1.10 release. For kernel implementations please use AutoDispatchBelowADInplaceOrView instead, If you are looking for a user facing API to enable running your inf
lease use c10::InferenceMode. Using AutoDispatchBelowADInplaceOrView in user code is under risk of producing silent wrong result in some edge cases. See Note [AutoDi
r more details. (Triggered internally at build/CMakeFiles/torch_npu.dir/compiler_depend.ts:74.)
  sorted_indices_to_remove[..., -self.min_tokens_to_keep :] = 0
图中是一名女子在沙滩上和狗玩耍，身边的狗是一只拉布拉多犬，它们处于沙滩上。
/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/urllib3/connectionpool.py:1013: InsecureRequestWarning: Unverified HTTPS request is being made
. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/1.26.x/advanced-usage.html#ssl-warnings
  warnings.warn(
<ref>击掌</ref><box>(523,513),(588,608)</box>
```

图 4-77 推理结果（2）

Step5 调用 API 接口推理

进入源码根目录，安装依赖。

```
cd Qwen-VL
pip install -r  requirements_openai_api.txt
```

修改openai_api.py脚本，适配NPU。

```
# 在openai_api.py脚本的import torch下新增两行
import torch_npu
from torch_npu.contrib import transfer_to_npu
```

启动API Server。执行命令前请先修改参数。

```
python openai_api.py -c ${model-path} --server-name=${server-name} --server-port=${server-port}
```

参数说明

- model-path: 模型权重路径，例：/home/ma-user/Qwen-VL-Chat
- server-name：进程监听IP或者域名
- server-port：进程监听端口，如果不配置该参数，启动端口默认是8000。

本地调用

本地调用命令如下。

```
# 调用地址以实际为准
curl -kv -X POST localhost:{server_port}/v1/chat/completions -H "Content-Type: application/json" -d
'{"model": "qwen","messages": [{"role":"user","content":"Picture id: <img>img_path/</img>\n{your
prompt}"}]}'
```

参数说明：

- {server_port}：启动服务端时配置的监听端口
- content：对于带图像输入的内容可表示为 Picture id: <img>img_path</img>\n{your prompt}，其中id表示对话中的第几张图片。"img_path"可以是本地的图片或网络地址。如果是本地图片，容器需要有权限读取图片。网络图片服务端会自动下载。

messages的样例如下：

```
# body参考
# 图片存放本地示例
{
  "messages": [
    {
      "role": "user",
      "content": "Picture 1: <img>/tmp/upload/demo.jpg</img>图片上有什么"
    }
  ],
  "model": "qwen"
}
```

图 4-78 本地调用

```
[root@ei-bms-001 ~]# curl -kv -X POST localhost:8000/v1/chat/completions -H "Content-Type: application/json" -d '{"model": "qwen","messages": [{"role":"user","content":"Picture 1: <img>/ho
me/wy/code/demo.jpeg</img>图片上有什么"}]}'
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying ::1:8000...
* connect to ::1 port 8000 failed: Connection refused
*   Trying 127.0.0.1:8000...
* Connected to localhost (127.0.0.1) port 8000 (#0)
> POST /v1/chat/completions HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.71.1
> Accept: */*
> Content-Type: application/json
> Content-Length: 123
>
* upload completely sent off: 123 out of 123 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< date: Mon, 17 Jun 2024 13:56:50 GMT
< server: uvicorn
< content-length: 287
< content-type: application/json
<
* Connection #0 to host localhost left intact
{"model":"qwen","object":"chat.completion","choices":[{"index":0,"message":{"role":"assistant","content":"图中是一名女子在沙滩上与狗玩耍，身边的狗是一只拉布拉多犬，它们处于沙滩上。","funct
ion_call":null},"finish_reason":"stop"}],"created":1718632613}[root@ei-bms-001 ~]#
```

远程调用

远程调用命令如下：

```
# endpoint以实际为准，图片为远程网络上的url

curl -kv -X POST http://{server_name}:{server_port}/v1/chat/completions -H "Content-Type: application/
json" -d '{"model": "qwen","messages": [{"role":"user","content":"Picture 1: <img>https://qianwen-res.oss-
cn-beijing.aliyuncs.com/Qwen-VL/assets/demo.jpeg</img>图片上有什么"}]}'
```

参数说明

- {server_name}：配置为服务端启动时监听的IP地址或者域名。

- {server_port}：配置为服务端启动时监听的端口。

图 4-79 远程调用

```
[root@devserver-bms-9f91cddb ~]# curl -kv -X POST http://100.95.148.27:9999/v1/chat/completions -H "Content-Type: application/json" -d '{"model": "qwen","messages": [{"r
ole":"user","content":"Picture 1: <img>https://qianwen-res.oss-cn-beijing.aliyuncs.com/Qwen-VL/assets/demo.jpeg</img>图片上有什么"}]}'
Note: Unnecessary use of -X or --request, POST is already inferred.
* Uses proxy env variable no_proxy == '127.0.0.1,localhost,mirrors.tools.huawei.com,mirrors.myhuaweicloud.com'
*   Trying 100.95.148.27:9999...
* Connected to 100.95.148.27 (100.95.148.27) port 9999 (#0)
> POST /v1/chat/completions HTTP/1.1
> Host: 100.95.148.27:9999
> User-Agent: curl/7.71.1
> Accept: */*
> Content-Type: application/json
> Content-Length: 172
>
* upload completely sent off: 172 out of 172 bytes
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< date: Tue, 18 Jun 2024 11:57:05 GMT
< server: uvicorn
< content-length: 296
< content-type: application/json
<
* Connection #0 to host 100.95.148.27 left intact
{"model":"qwen","object":"chat.completion","choices":[{"index":0,"message":{"role":"assistant","content":"图中是一名女子在沙滩上和狗玩耍，旁边的狗是一只黄色的拉布拉多犬
。它们处于沙滩上。","function_call":null},"finish_reason":"stop"}],"created":1718711829}[root@devserver-bms-9f91cddb ~]#
```

4.16 Open-Sora 1.0 基于 DevServer 适配 PyTorch NPU 训练指导（6.3.905）

本文档主要介绍如何在ModelArts Lite DevServer上，使用PyTorch_npu+华为自研Ascend Snt9B硬件，完成Open-Sora训练和推理。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。训练至少需要单机8卡，推理需要单机单卡。

表 4-30 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-31 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。

分类	名称	获取路径
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表4-31**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 本文档适配的是
- 训练至少需要单机8卡，推理需要单机单卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参见**表4-31**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。训练至少需要单机8卡，推理需要单机单卡。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
// 启动一个容器去运行镜像
docker run -itd \
      --device=/dev/davinci0 \
      --device=/dev/davinci1 \
      --device=/dev/davinci2 \
      --device=/dev/davinci3 \
      --device=/dev/davinci4 \
      --device=/dev/davinci5 \
      --device=/dev/davinci6 \
      --device=/dev/davinci7 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 80g \
      --net=bridge \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。
- ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的大文件系统，work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- shm-size：共享内存大小，建议不低于80GB。
- name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- ${image_name}：代表镜像地址。

说明

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

3. 进入容器。需要将${container_name}替换为实际的容器名称。

```
docker exec -it ${container_name} bash
```

说明

启动容器默认使用ma-user用户。后续所有命令执行也建议使用ma-user用户。

Step3 获取代码包并安装依赖

1. 下载插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，上传到容器的/home/ma-user/目录下，解压并安装相关依赖。获取路径参见**获取软件和镜像**。

```
mkdir -p /home/ma-user/ascendcloud-aigc-algorithm-open_sora    #创建目录
cd /home/ma-user/ascendcloud-aigc-algorithm-open_sora/         #进入目录

unzip -zxvf AscendCloud-3rdAIGC-6.3.905-*.zip
tar -zxvf ascendcloud-aigc-algorithm-open_sora.tar.gz
rm -rf AscendCloud-3rdAIGC-6.3.905-*
```

2. 安装Python环境。

```
pip install -r requirements.txt
cp attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
cp low_level_optim.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/colossalai/zero/low_level/low_level_optim.py
```

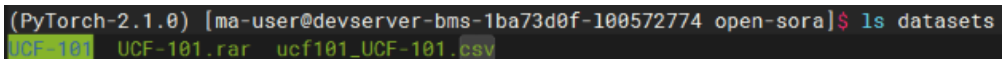
Step4 下载数据集

训练使用的开源数据集UCF101.rar，执行如下命令下载数据集并处理。数据集相关介绍参见**https://www.crcv.ucf.edu/data/UCF101.php**。

```
mkdir datasets
cd datasets
wget https://www.crcv.ucf.edu/data/UCF101/UCF101.rar
unrar x UCF101.rar
cd ..
python -m tools.datasets.convert_dataset ucf101 ./datasets/ --split UCF-101
mv ucf101_UCF-101.csv datasets/
```

处理完数据集后的结果如**图4-80**所示。

图 4-80 处理后的数据文件



Step5 启动训练服务

训练至少需要单机8卡。建议手动下载所需的权重文件，放在weights文件夹下。在/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录下进行操作。

1. 创建weights文件夹。

```
mkdir weights
```

2. 下载基础模型权重：PixArt-XL-2-512x512.pth和PixArt-XL-2-256x256.pth

```
cd weights
# 下载PixArt-XL-2-512x512.pth和PixArt-XL-2-256x256.pth
wget https://huggingface.co/PixArt-alpha/PixArt-alpha/resolve/main/PixArt-XL-2-512x512.pth
wget https://huggingface.co/PixArt-alpha/PixArt-alpha/resolve/main/PixArt-XL-2-256x256.pth
```

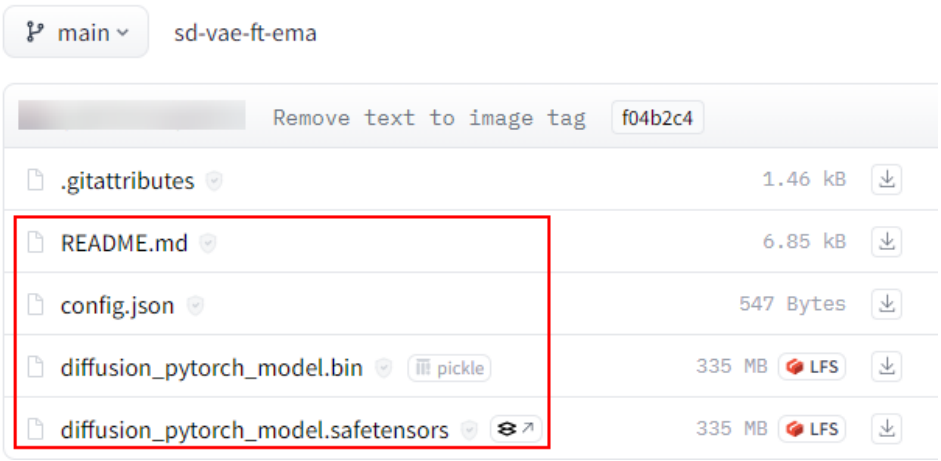
3. 下载VAE权重：sd-vae-ft-ema

   在weights文件夹下创建sd-vae-ft-ema文件夹。

```
mkdir sd-vae-ft-ema
```

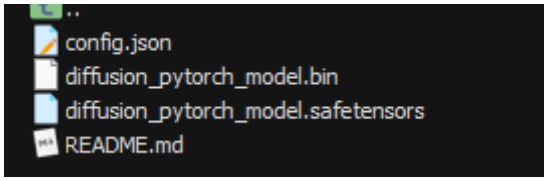
   然后进入官网地址： **https://huggingface.co/stabilityai/sd-vae-ft-ema/tree/main**，手动下载如**图4-81**所示四个文件，并上传到服务器的/home/ma-user/ascendcloud-aigc-algorithm-open_sora/weights/sd-vae-ft-ema/目录下。

图 4-81 Huggingface 中 sd-vae-ft-ema 模型目录内容



上传完成后，weights/sd-vae-ft-ema/目录内容如图4-82所示。

图 4-82 服务器 weights/sd-vae-ft-ema/目录内容



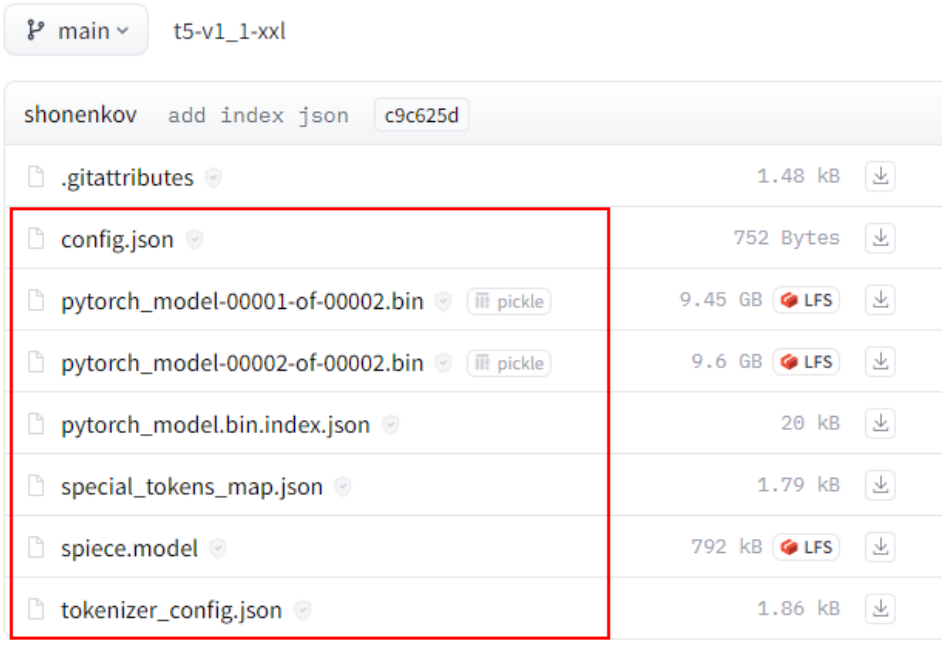
4. 下载Encoder模型权重：DeepFloyd/t5-v1_1-xxl

   在weights文件夹下创建t5-v1_1-xxl文件夹。

   ```
   mkdir t5-v1_1-xxl
   ```

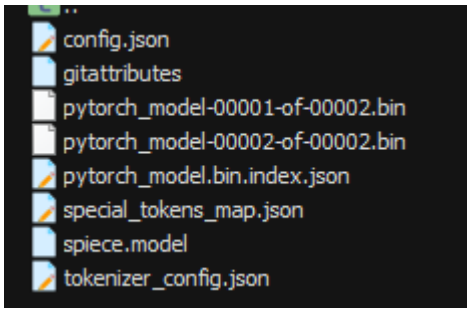
   然后进入官网地址 **https://huggingface.co/DeepFloyd/t5-v1_1-xxl/tree/main**，手动下载如图4-83所示文件，并放到 /home/ma-user/ascendcloud-aigc-algorithm-open_sora/weights/t5-v1_1-xxl 文件夹下。

图 4-83 Huggingface 中 t5-v1_1-xxl 模型目录内容



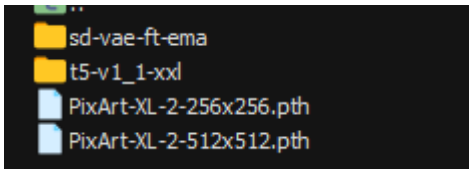
上传完成后，weights/t5-v1_1-xxl/目录下内容如图4-84所示。

图 4-84 服务器 weights/t5-v1_1-xxl/目录内容



最后weights文件夹下内容目录如图4-85所示。

图 4-85 服务器 weights 目录



从weights目录下返回到代码目录下。

```
cd ..
```

5. 在/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录下执行如下命令启动训练脚本。

```
torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
```

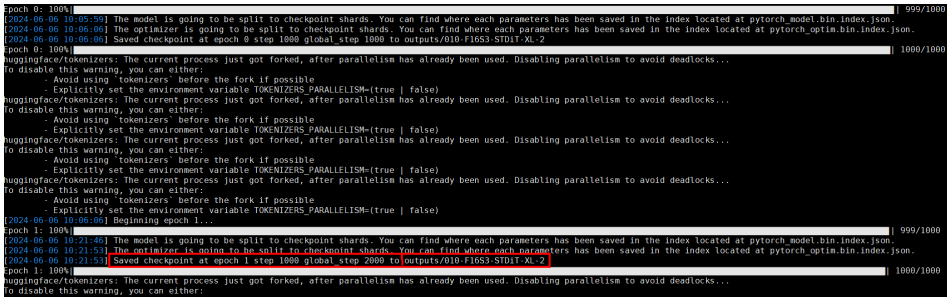

正常训练过程如下图所示。训练完成后，关注loss值，loss曲线收敛，记录总耗时和单步耗时。训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

图 4-86 正常训练过程

```
Epoch 30:   0%|                                                   | 0/25 [00:
===global_step:750,loss:0.034547336399555206

Epoch 30:   4%|███▌                                      | 1/25 [00:06<02:33
===global_step:751,loss:0.00026112061459571123

Epoch 30:   8%|███████▏                                   | 2/25 [00:13<02:28
===global_step:752,loss:0.001797467702999711

Epoch 30:  12%|██████████▋                                | 3/25 [00:19<02:23
===global_step:753,loss:0.03514224290847778

Epoch 30:  16%|██████████████▎                            | 4/25 [00:26<02:17
===global_step:754,loss:0.0446937158703804
```

训练完成后权重保存在自动生成的目录，例如：outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/。

图 4-87 训练完成后权重保存在信息



```
Epoch 0: 100%|██████████████████████████████████████████████████| 999/1000
[2024-06-06 10:05:59] The model is going to be split to checkpoint shards. You can find where each parameters has been saved in the index located at pytorch_model.bin.index.json.
[2024-06-06 10:06:06] The optimizer is going to be split to checkpoint shards. You can find where each parameters has been saved in the index located at pytorch_optim.bin.index.json.
[2024-06-06 10:06:06] Saved checkpoint at epoch 0 step 1000 global_step 1000 to outputs/010-F16S3-STDiT-XL-2
Epoch 0: 100%|█████████████████████████████████████████████████| 1000/1000
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
[2024-06-06 10:06:06] Beginning epoch 1...
Epoch 1: 100%|█████████████████████████████████████████████████| 999/1000
[2024-06-06 10:21:46] The model is going to be split to checkpoint shards. You can find where each parameters has been saved in the index located at pytorch_model.bin.index.json.
[2024-06-06 10:21:53] The optimizer is going to be split to checkpoint shards. You can find where each parameters has been saved in the index located at pytorch_optim.bin.index.json.
[2024-06-06 10:21:53] Saved checkpoint at epoch 1 step 1000 global_step 2000 to outputs/010-F16S3-STDiT-XL-2
Epoch 1: 100%|████████████████████████████████████████████████| 1000/1000
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
```

Step6 推理

执行如下命令使用官方权重推理。推理脚本inference.py 会自动下载官方权重文件。

```
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```

说明

如果自动下载官方权重文件OpenSora-v1-HQ-16x512x512.pth失败，建议手动下载权重文件并上传到容器/home/ma-user/ascendcloud-aigc-algorithm-open_sora/目录中。

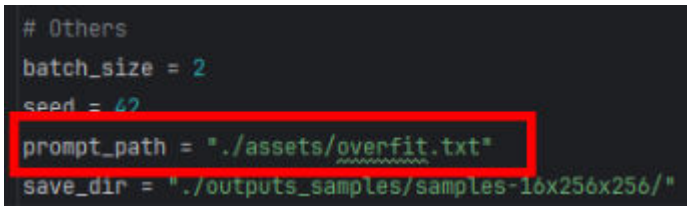
"OpenSora-v1-HQ-16x512x512.pth": "**https://huggingface.co/hpcai-tech/Open-Sora/resolve/main/OpenSora-v1-HQ-16x512x512.pth**"

执行如下命令使用训练后生成的权重推理。训练完成后会在工作目录/home/ma-user/ascendcloud-aigc-algorithm-open_sora/下自动生成一个outputs文件夹，训练后生成的权重文件存放在outputs文件夹中，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/。

```
export CKPT_PATH=./outputs/.../ #由训练日志中获得
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

如果要使用自己的prompt进行推理，可以修改用户自己推理脚本配置文件中prompt_path。例如在configs/opensora/inference/64x512x512.py配置文件中，使用了自己的prompt文件overfit.txt。

图 4-88 修改 prompt_path



Step7 精度对比

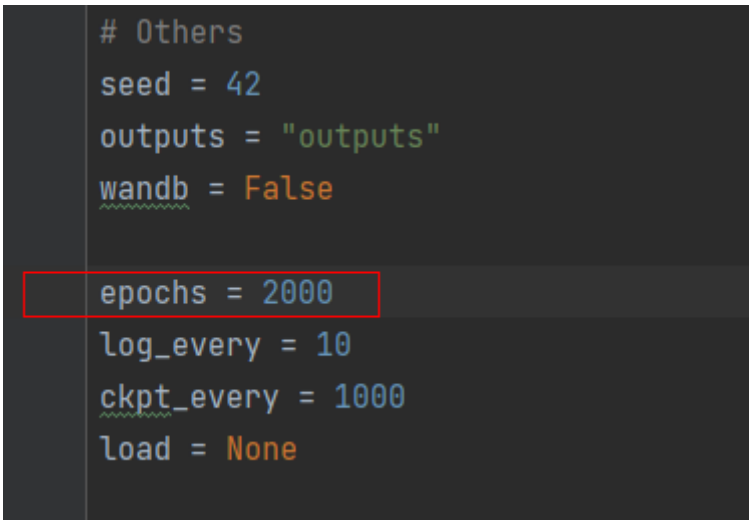
由于NPU和GPU生成的随机数不一样，需要固定二者的随机数再进行精度对比。通常的做法是先用GPU单卡跑一遍训练，生成固定下来的随机数。然后NPU和GPU都用固定的随机数进行单机8卡训练，比较精度。

1. 训练精度对齐。对齐前2000步的loss，观察loss在极小误差范围内。

   GPU环境下，使用Github中的官方代码跑训练任务。Github中的官方代码下载路径：**https://github.com/hpcaitech/Open-Sora/tree/v1.0.0**

   在NPU代码 configs/opensora/train/64x512x512.py中把 epochs = 200000 临时改成 epochs = 2000

   图 4-89 配置文件 64x512x512.py 修改训练步数

   将NPU代码中configs/opensora/train/64x512x512.py文件和configs/opensora/inference/64x512x512.py文件复制到GPU代码目录中，使用相同的参数配置文件。

   将NPU代码目录中的opensora/schedulers/iddpm/__init__.py文件和opensora/schedulers/iddpm/gaussian_diffusion.py文件复制到GPU代码目录中，添加固定随机数功能。

   进行GPU单机八卡训练，生成固定训练随机数，随机数会保存在noise文件夹中。

```
mkdir noise_train  #创建文件夹noise_train，用于存放生成的随机数
export LOCK_RAND=True  #是否固定随机数
export SAVE_RAND=True  #是否保存生成的随机数
export NOISE_PATH="./noise_train"  #将生成的随机数保存在"./noise_train"目录
torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
```

说明

正常训练时不需要增加如下命令，只有训练精度对比时需要。

```
export LOCK_RAND=True   #是否固定随机数
export SAVE_RAND=True   #是否保存生成的随机数
export NOISE_PATH="./noise_train"    #将生成的随机数保存在"./noise_train"目录
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机8卡训练，比较在相应目录下生成的loss.txt文件。在NPU训练前，需要将上面GPU单机单卡训练生成的"./noise_train"文件夹移到NPU相同目录下。NPU和GPU的训练命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_train"
torchrun --nnodes=1 --nproc_per_node=8 train.py configs/opensora/train/64x512x512.py
```

GPU和NPU训练脚本中的参数要保持一致，除了参数dtype。NPU环境下，dtype="fp16"，GPU环境下，dtype="bf16"。

2. 基于NPU训练后的权重文件和GPU训练后的权重文件，对比推理精度。推理精度对齐流程和训练精度对齐流程相同，先在GPU固定推理的随机数。

```
mkdir noise_test1  #创建文件夹noise_test1，用于存放生成的随机数
export LOCK_RAND=True   #是否固定随机数
export SAVE_RAND=True    #是否保存生成的随机数
export NOISE_PATH="./noise_test1"    #将生成的随机数保存在"./noise_test1"目录
export CKPT_PATH=./outputs/.../ #由训练日志中获得，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机单卡推理，比较生成的视频是否一致。在NPU推理前，需要将上面GPU单机单卡推理生成的"./noise_test1"文件夹移到NPU相同目录下。NPU和GPU的推理命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_test1"
export CKPT_PATH=./outputs/.../ #由训练日志中获得，例如outputs/010-F16S3-STDiT-XL-2/epoch1-global_step2000/
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path $CKPT_PATH
```

3. 基于官方权重文件分别在GPU和NPU进行推理，对比推理精度。推理精度对齐流程和训练精度对齐流程相同，先在GPU固定推理的随机数。

```
mkdir noise_test2  #创建文件夹noise_test2，用于存放生成的随机数
export LOCK_RAND=True  #是否固定随机数
export SAVE_RAND=True   #是否保存生成的随机数
export NOISE_PATH="./noise_test2"  #将生成的随机数保存在"./noise_test2"目录
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```

在NPU和GPU机器使用上面生成的固定随机数，分别跑一遍单机单卡推理，比较生成的视频是否一致。在NPU推理前，需要将上面GPU单机单卡推理生成的"./noise_test2"文件夹移到NPU相同目录下。NPU和GPU的推理命令相同，如下。

```
export LOCK_RAND=True
export SAVE_RAND=False
export NOISE_PATH="./noise_test2"
torchrun --standalone --nproc_per_node 1 inference.py configs/opensora/inference/64x512x512_npu.py --ckpt-path ./OpenSora-v1-HQ-16x512x512.pth
```

4.17 SDXL 基于 Standard 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。SDXL Finetune是指在已经训练好的SDXL模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何在ModelArts Standard上，利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL Finetune训练。

获取软件和镜像

表 4-32 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

表 4-33 模型镜像版本

模型	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**获取软件和镜像**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练作业至少需要单机8卡。
- 确保容器可以访问公网。
- 本案例仅支持在专属资源池上运行。

Step1 创建专属资源池

本文档中的模型运行环境是ModelArts Standard，用户需要购买专属资源池，具体步骤请参考**创建资源池**。

资源规格要求：

- 硬盘空间：至少200GB。
- 昇腾资源规格：Ascend: 8*ascend-snt9b表示昇腾8卡规格。
- 推荐使用“西南-贵阳一”Region上的昇腾资源。

Step2 创建 OBS 桶

ModelArts使用对象存储服务（Object Storage Service，简称OBS）进行数据存储以及模型的备份和快照，实现安全、高可靠和低成本的存储需求。因此，在使用ModelArts之前通常先创建一个OBS桶，然后在OBS桶中创建文件夹用于存放数据。

本文档需要将运行代码以及输入输出数据存放OBS，请提前创建OBS（参考**创建OBS桶**），例如桶名：sdxl-train。并在该桶下创建文件夹目录用于后续存储代码使用，例如：code。

Step3 准备代码

在**获取软件和镜像**中，下载并解压代码包。本文档主要使用ascendcloud-aigc-poc-sdxl-finetune文件夹中的文件，请利用**OBS Browser+工具**将文件夹中内容上传至OBS的代码文件夹code中。

```
obs://<bucket_name>/code
├── attention_processor.py
├── config.yaml
├── diffusers_finetune_train.sh
├── train_text_to_image_sdxl-0212.py
```

Step4 下载模型依赖包

请在如下链接中下载好模型依赖包。

- 下载stable-diffusion-xl-base-1.0，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
- 下载vae-fp16-fix，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**

Step5 下载数据集

本案例使用Huggingface提供的pokemon-blip-captions数据集，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**

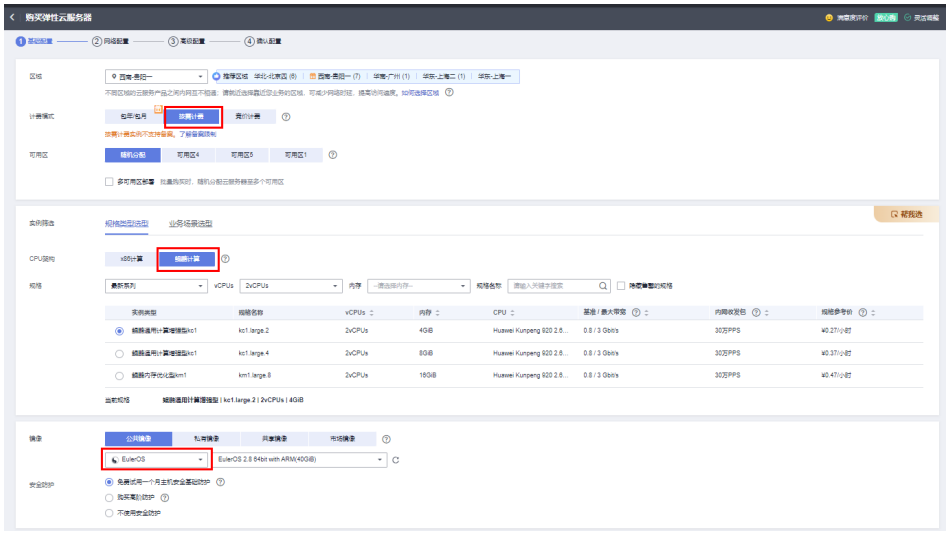
Step6 准备镜像

步骤1 创建ECS。

参考**ECS文档**购买弹性云服务器。网络配置、高级配置等后续步骤，可根据默认选择，或进行自定义。创建完成后，单击“远程登录”，并在控制台发送后续步骤中的远程命令。

注意：创建的ECS虚拟机使用ARM镜像创建。

图 4-90 购买 ECS



步骤2 安装Docker。

1. 检查docker是否安装。

```
docker -v  #检查docker是否安装
```

如尚未安装，运行以下命令安装docker。

```
yum install -y docker
```

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

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

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

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

步骤3 构建自定义镜像。

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**获取软件和镜像**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /
home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_finetune_train.sh /home/ma-user/sdxl-train/user-job-dir/code/
diffusers_finetune_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/
code/train_text_to_image_sdxl-0212.py
COPY --chown=ma-user:ma-group config.yaml /home/ma-user/sdxl-train/user-job-dir/code/config.yaml

COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-
base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
COPY --chown=ma-user:ma-group attention_processor.py /home/ma-user/anaconda3/envs/
PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
```

把ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件、模型依赖包、数据集、Dockerfile文件都上传至ECS，上传步骤可参考**本地Windows主机使用WinSCP上传文件到Linux云服务器**。

文件上传后目录如下：

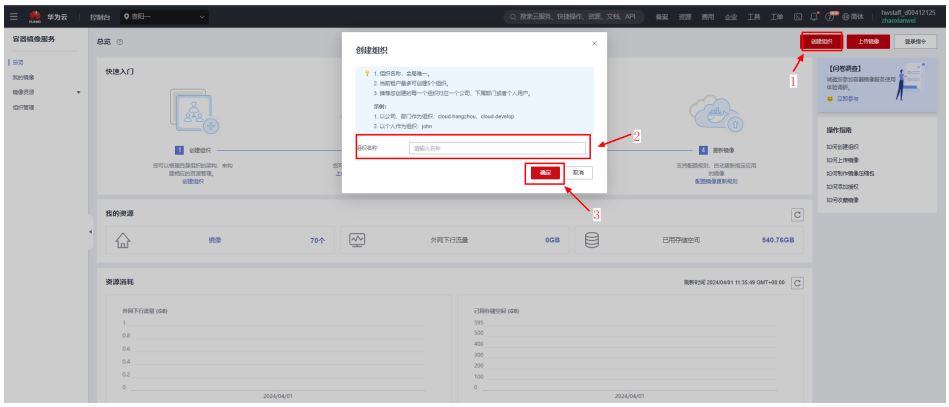
```
<ECS_folder>
├── attention_processor.py              #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── config.yaml                         #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── diffusers_finetune_train.sh         #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── train_text_to_image_sdxl-0212.py    #  ascendcloud-aigc-poc-sdxl-finetune代码文件夹文件
├── Dockerfile                          #  Dockerfile文件
├── vae-fp16-fix                        #  模型依赖包vae-fp16-fix
├── stable-diffusion-xl-base-1.0x       #  模型依赖包stable-diffusion-xl-base-1.0x
├── datasets                            #  新建datasets文件夹，pokemon-blip-captions数据集放在该目录下
    ├── lambdalabs___pokemon-blip-captions
```

在该目录下执行命令构建自定义镜像：

```
docker build -t sdxl-train:0.0.1 .
```

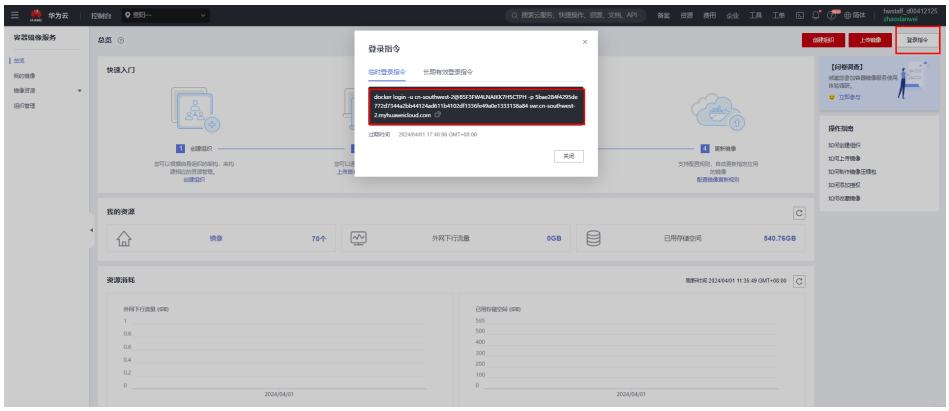
步骤4 在SWR服务页面创建镜像组织。

图 4-91 创建镜像组织



步骤5 在SWR中单击右上角的“登录指令”，然后在跳出的登录指定窗口，单击复制临时登录指令。在创建的ECS中粘贴临时登录指令，即可完成登录。

图 4-92 复制登录指令



步骤6 修改并上传镜像。

在ECS中输入上一步的登录指令后，使用下列示例命令：

```
docker tag {image_url} swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
docker push swr.myhuaweicloud.com/<组织名称>/<镜像名称>:<tag>
```

参数说明：

<组织名称>：步骤4中创建的组织名称。

<镜像名称>:<tag>：定义镜像名称。示例：sdxl-train:0.0.1。

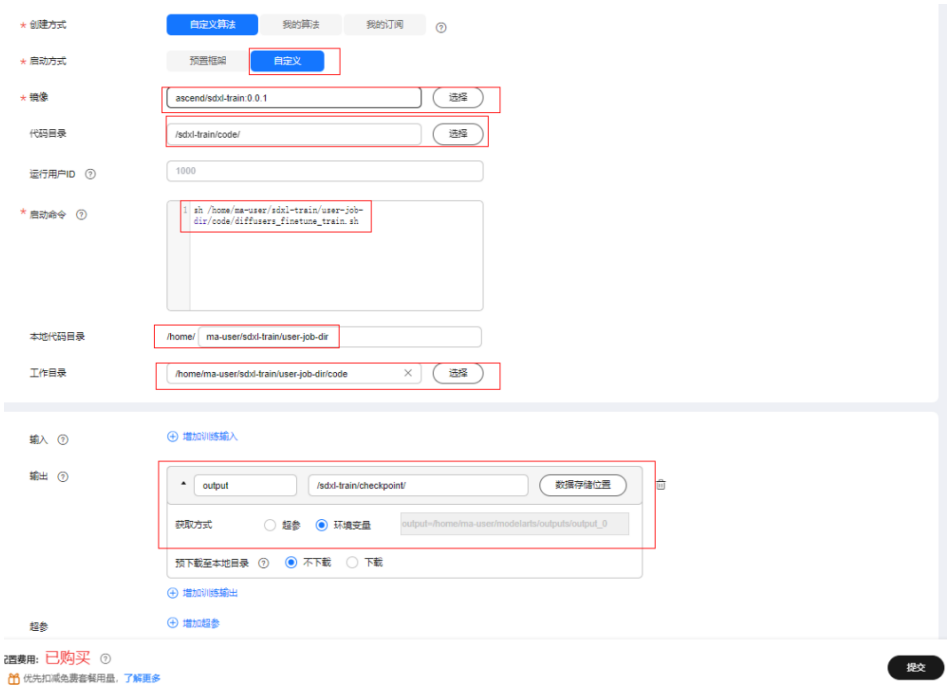
----结束

Step7 创建训练作业

创建训练作业，填下如下参数。

- 创建方式：选择自定义算法，启动方式选择自定义，然后选择上传到SWR的自定义镜像。
- 代码目录：选择上传到OBS的代码文件夹，例如/sdxl-train/code。若用户需要修改代码文件，可修改OBS桶中代码文件，创建训练作业时，会将OBS的code目录复制到训练容器的/home/ma-user/sdxl-train/user-job-dir/目录下，覆盖容器中原有的code目录。
- 启动命令：直接运行启动脚本文件diffusers_finetune_train.sh。
  ```
  sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
  ```
- 本地代码目录：保持默认即可。
- 工作目录：选择代码文件目录，例如/home/ma-user/sdxl-train/user-job-dir/code/。
- 输出：单击“增加训练输出”，将模型保存到OBS中。参数名称为output，数据存储位置选择OBS桶中制定文件夹，例如sdxl-train/checkpoint，获取方式选择环境变量，/home/ma-user/modelarts/outputs/output_0下的模型文件会保存到OBS中。

图 4-93 选择镜像



- 资源池：选择专属资源池，规格选择Ascend: 8*ascend-snt9b。如果需要多机训练，增加计算节点个数即可，启动脚本文件diffusers_finetune_train.sh支持多机训练。

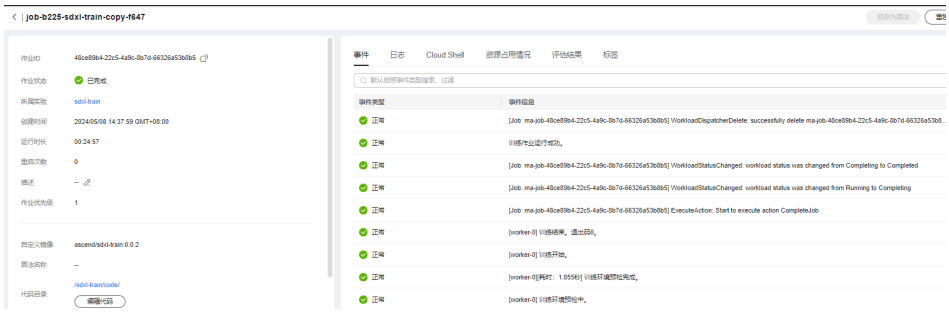
图 4-94 选择资源池规格

- 作业日志路径：选择输出日志到OBS的指定目录。

图 4-95 选择作业日志路径

填写参数完成后，提交创建训练任务，训练完成后，作业状态会显示为已完成。

图 4-96 训练完成



4.18 SDXL 基于 DevServer 适配 PyTorch NPU 的 Finetune 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。SDXL Finetune是指在已经训练好的SDXL模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 4-34 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-35 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名发布的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表4-35**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 训练资源需要使用单机8卡。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。
```
npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
```
如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。
3. 检查是否安装docker。
```
docker -v   #检查docker是否安装
```
如尚未安装，运行以下命令安装docker。
```
yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
```
4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 下载代码包、依赖模型包和数据集

1. 下载stable-diffusion-xl-base-1.0模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
2. 下载vae-fp16-fix模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
3. 下载开源数据集并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**。用户也可以使用自己的数据集。
4. 下载SDXL插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，获取路径参见**获取软件和镜像**。本案例使用的是AscendCloud-3rdAIGC-6.3.905-xxx.zip文件中的ascendcloud-aigc-poc-sdxl-finetune.tar.gz代码包。解压后上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。
```
[root@devserver-ei-cto-office-ae06cae7-tmp1216 docker_build]# ll
total 192
-rw-------  1 root root 108286 May  6 16:56 attention_processor.py
-rw-------  1 root root    430 May  8 09:31 config.yaml
drwx------  3 root root   4096 May  7 10:50 datasets
-rw-------  1 root root   1356 May  8 16:30 diffusers_finetune_train.sh
-rw-------  1 root root   1468 May  8 16:49 Dockerfile    #需要用户参考Step3构建镜像步骤写Dockerfile文件
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
-rw-------  1 root root  58048 May  8 17:48 train_text_to_image_sdxl-0212.py
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-35**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_finetune_train.sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/code/train_text_to_image_sdxl-0212.py
COPY --chown=ma-user:ma-group config.yaml /home/ma-user/sdxl-train/user-job-dir/code/config.yaml
```

```
COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
COPY --chown=ma-user:ma-group attention_processor.py /home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/diffusers/models/attention_processor.py
```

构建自定义镜像sdxl-train:0.0.1。

```
docker build -t sdxl-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0 --device=/dev/davinci1 --device=/dev/davinci2 --device=/dev/davinci3 --device=/dev/davinci4 --device=/dev/davinci5 --device=/dev/davinci6 --device=/dev/davinci7 --security-opt seccomp=unconfined --network=bridge sdxl-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0，...， --device=/dev/davinci7：挂载NPU设备，示例中挂载了8张卡davinci0~davinci7。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

进入容器。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it sdxl-train bash
```

Step5 修改算法脚本

进入容器后，修改启动脚本文件。

```
vi /home/ma-user/sdxl-train/user-job-dir/code/diffusers_finetune_train.sh
```

在第2行增加export MA_NUM_HOSTS=1 即可，如：

```
#!/bin/bash
export MA_NUM_HOSTS=1
if [[ $MA_NUM_HOSTS == 1 ]]; then
```

Step6 启动训练服务

执行如下命令运行训练脚本。

```
cd /home/ma-user/sdxl-train/user-job-dir/code
sh diffusers_finetune_train.sh
```

说明

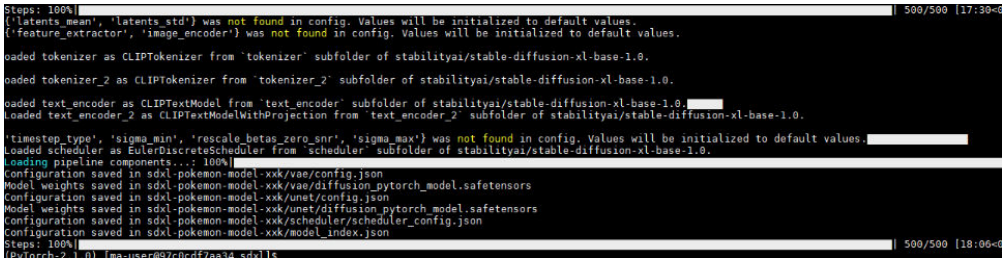
训练执行脚本中配置了保存checkpoint的频率，每500steps保存一次，如果磁盘空间较小，这个值可以改大到5000，避免磁盘空间写满，导致训练失败终止。

checkpoint保存频率的修改命令如下：

```
--checkpointing_steps=5000
```

训练执行成功如下图所示。

图 4-97 训练执行成功



4.19 SDXL 基于 DevServer 适配 PyTorch NPU 的 LoRA 训练指导（6.3.905）

Stable Diffusion（简称SD）是一种基于扩散过程的图像生成模型，应用于文生图场景，能够帮助我们生成图像。SDXL LoRA训练是指在已经训练好的SDXL模型基础上，使用新的数据集进行LoRA微调以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，完成SDXL的LoRA微调训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 4-36 环境要求

名称	版本
CANN	cann_8.0.rc2
PyTorch	2.1.0

获取软件和镜像

表 4-37 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-3rdAIGC-6.3.905-xxx.zip 文件名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像包	swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240528150158-b521cc0	SWR上拉取

约束限制

- 本文档适配昇腾云ModelArts 6.3.905版本，请参考**表4-37**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- Lora训练使用单机单卡资源。
- 确保容器可以访问公网。

Step1 检查环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                 # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 下载代码包、依赖模型包和数据集

1. 下载stable-diffusion-xl-base-1.0模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
2. 下载vae-fp16-fix模型包并上传到宿主机上，官网下载地址：**https://huggingface.co/madebyollin/sdxl-vae-fp16-fix/tree/main**
3. 下载开源数据集并上传到宿主机上，官网下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/tree/main**。用户也可以使用自己的数据集。
4. 下载SDXL插件代码包AscendCloud-3rdAIGC-6.3.905-xxx.zip文件，获取路径参见**获取软件和镜像**。本案例使用的是AscendCloud-3rdAIGC-6.3.905-xxx.zip文件中的ascendcloud-aigc-poc-sdxl-lora-train.tar.gz代码包。解压后上传到宿主机上。

   依赖的插件代码包、模型包和数据集存放在宿主机上的本地目录结构如下，供参考。

   ```
   [root@devserver-ei-cto-office-ae06cae7-tmp1216 docker_build]# ll
   total 192
   ```

```
-rw-------  1 root root 108286 May  6 16:56 npu_attention_processor.py
drwx------  3 root root   4096 May  7 10:50 datasets
-rw-------  1 root root   1356 May  8 16:30 diffusers_lora_train.sh
drwx------ 10 root root   4096 Apr 30 15:18 stable-diffusion-xl-base-1.0
-rw-------  1 root root  58048 May  8 17:48 train_text_to_image_lora_sdxl-0212.py
drwx------  2 root root   4096 Apr 30 15:17 vae-fp16-fix
```

Step3 构建镜像

基于官方提供的基础镜像构建自定义镜像sdxl-train:0.0.1。参考如下命令编写Dockerfile文件。镜像地址{image_url}请参见**表4-37**。

```
FROM {image_url}

RUN mkdir /home/ma-user/sdxl-train && mkdir /home/ma-user/sdxl-train/user-job-dir && mkdir /home/ma-user/sdxl-train/user-job-dir/code
COPY --chown=ma-user:ma-group diffusers_lora_train.sh /home/ma-user/sdxl-train/user-job-dir/code/diffusers_lora_train.sh
COPY --chown=ma-user:ma-group train_text_to_image_lora_sdxl-0212.py /home/ma-user/sdxl-train/user-job-dir/code/train_text_to_image_lora_sdxl-0212.py
COPY --chown=ma-user:ma-group npu_attention_processor.py /home/ma-user/sdxl-train/user-job-dir/code/npu_attention_processor.py

COPY --chown=ma-user:ma-group stable-diffusion-xl-base-1.0 /home/ma-user/sdxl-train/stable-diffusion-xl-base-1.0
COPY --chown=ma-user:ma-group vae-fp16-fix /home/ma-user/sdxl-train/vae-fp16-fix
COPY --chown=ma-user:ma-group datasets /home/ma-user/sdxl-train/datasets

RUN pip install  accelerate datasets transformers diffusers
RUN source /etc/bashrc && pip install  deepspeed
```

构建自定义镜像sdxl-train:0.0.1。

```
docker build -t sdxl-train:0.0.1 .
```

Step4 启动镜像

启动容器镜像。启动前可以根据实际需要增加修改参数。

```
docker run -itd --name sdxl-train -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v /etc/localtime:/etc/localtime -v /usr/local/Ascend/driver:/usr/local/Ascend/driver -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi --shm-size 60g --device=/dev/davinci_manager --device=/dev/hisi_hdc --device=/dev/devmm_svm --device=/dev/davinci0  --security-opt seccomp=unconfined --network=bridge sdxl-train:0.0.1 bash
```

参数说明：

- --device=/dev/davinci0：挂载NPU设备，单卡即可。

  说明

  - driver及npu-smi需同时挂载至容器。
  - 不要将多个容器绑到同一个NPU上，会导致后续的容器无法正常使用NPU功能。

进入容器。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it sdxl-train bash
```

Step5 安装依赖

安装pip依赖。

```
pip install diffusers==0.21.2
```

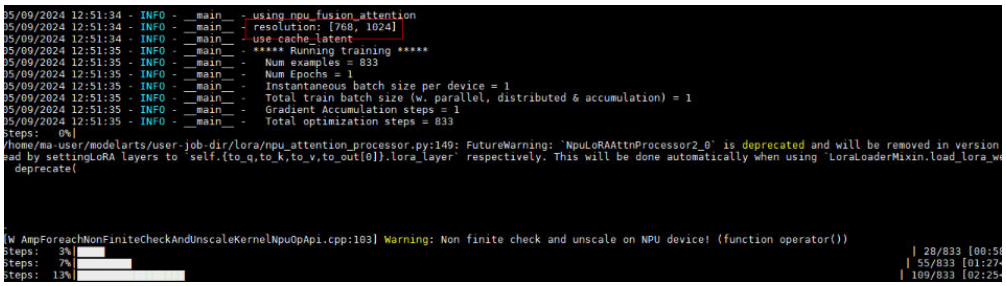
Step6 启动训练服务

执行如下命令启动训练脚本diffusers_lora_train.sh。

```
cd /home/ma-user/sdxl-train/user-job-dir/code
sh diffusers_lora_train.sh
```

训练执行成功如下图所示。

图 4-98 训练执行成功



4.20 SDXL ComfyUI 插件基于 DevServer 适配 PyTorch NPU 推理指导（6.3.904）

ComfyUI是一款基于节点工作流的Stable Diffusion操作界面。通过将Stable Diffusion的流程巧妙分解成各个节点，成功实现了工作流的精确定制和可靠复现。每一个节点都有特定的功能，可以通过调整节点连接达到不同的出图效果。在图像生成方面，它不仅比传统的WebUI更迅速，而且显存占用更为经济。

本文档主要介绍如何在ModelArts Lite的DevServer环境中部署ComfyUI，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署**ComfyUI**用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 4-38 环境要求

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-39 获取软件和镜像

分类	名称	获取路径
插件代码包	ascendcloud-aigc-6.3.904-*.tar.gz **说明** 包名中的*表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E网站**。 **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.904版本，请参考**表4-39**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

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

   说明

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

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表4-39**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称"
// 启动一个容器去运行镜像
docker run -itd \
      --device=/dev/davinci1 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 32g \
      --net=bridge \
      -p 8443:8443 \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。

```
docker exec -it ${container_name} bash
```

Step4 下载并安装软件

1. 从github下载ComfyUI代码并安装依赖。

```
cd /home/ma-user
git clone https://github.com/comfyanonymous/ComfyUI.git
cd ComfyUI
git reset --hard 831511a1eecbe271e302f2f2053f285f00614180
pip install -r requirements.txt
```

如果出现报错SSL certificate problem: self signed certificate in certificate chain

图 4-99 报错 SSL certificate problem

可采取忽略SSL证书验证：使用以下命令来克隆仓库，它将忽略SSL证书验证。

```
git clone -c http.sslVerify=false https://github.com/comfyanonymous/ComfyUI.git
```

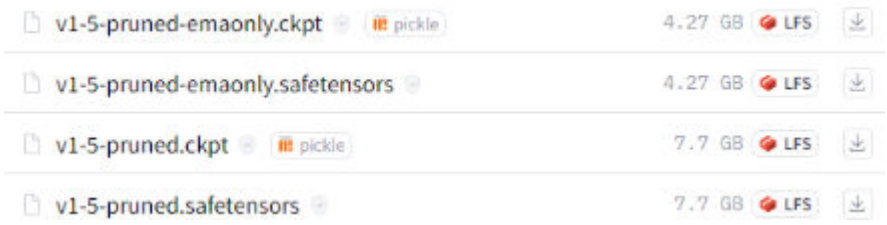
说明

此处根据ComfyUI官网描述进行配置。

2. 下载SD模型并安装。部署好ComfyUI环境和依赖后，还需要将模型放到对应位置。

   a. 下载模型，模型下载地址：**sd1.5模型地址** ，**sdxl下载地址**。根据自己的需要下载对应的模型，如下图，并将模型上传到容器内自定义挂载的工作目录。

   图 4-100 模型列表

   b. 将模型复制到/home/ma-user/ComfyUI/models/checkpoints目录下。

3. 将获取到的ComfyUI插件ascendcloud-aigc-6.3.904-*.tar.gz文件上传到容器的/home/ma-user/ComfyUI/custom_nodes目录下，并解压。获取路径参见**表4-39**。

```
cd /home/ma-user/ComfyUI/custom_nodes/
tar -zxvf ascendcloud-aigc-6.3.904-*.tar.gz
tar -zxvf ascendcloud-aigc-extensions-comfyui.tar.gz
rm -rf ascendcloud-aigc-6.3.904-*
```

说明

ascendcloud-aigc-6.3.904-*.tar.gz后面的*表示时间戳，请按照实际替换。

4. 使用容器IP启动服务。

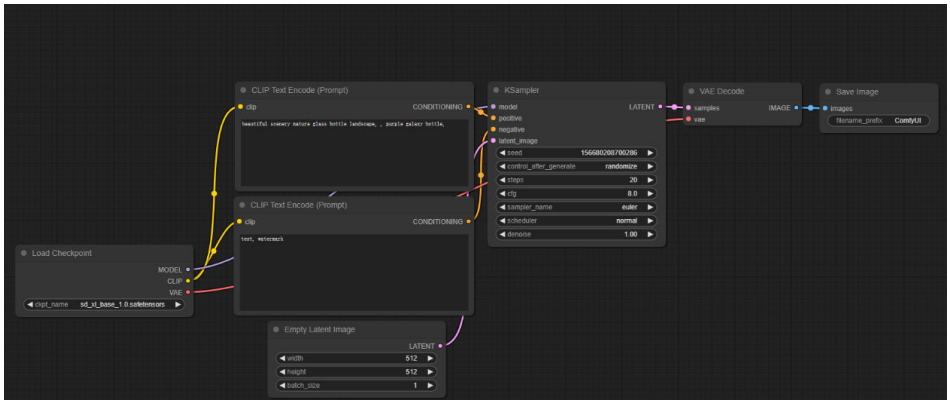
```
cd /home/ma-user/ComfyUI
python main.py --port 8443 --listen ${docker_ip} --force-fp16
```

${docker_ip}替换为容器实际的IP地址。可以在宿主机上通过docker inspect容器ID |grep IPAddress命令查询。

Step5 服务调用

1. 在浏览器中输入http://ip:8443访问界面，页面如下图。

图 4-101 访问界面

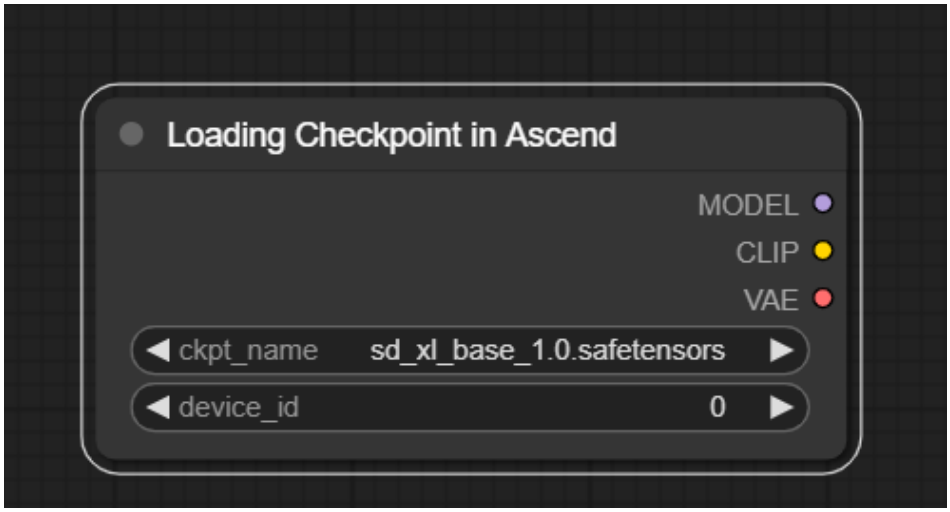


2. 双击访问页面，并搜索“Ascend”，单击“AscendNode”，如下图。

图 4-102 搜索 Ascend

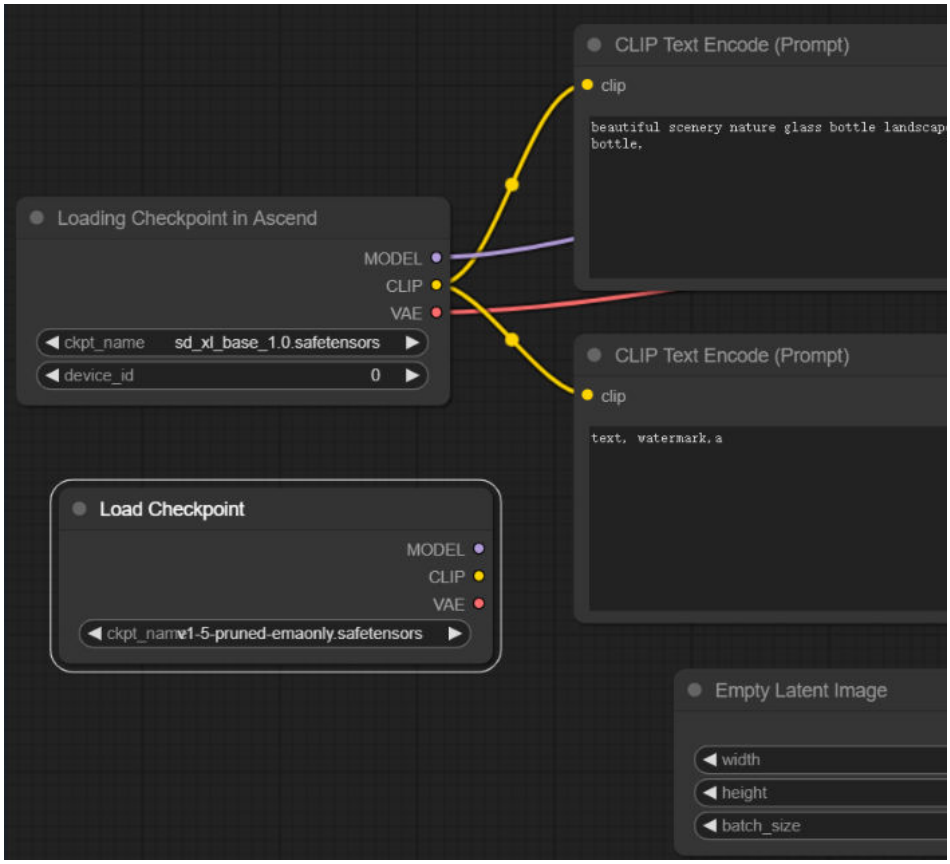
会得到一个新的关于NPU的checkpoint，如下图。

图 4-103 NPU 的 checkpoint



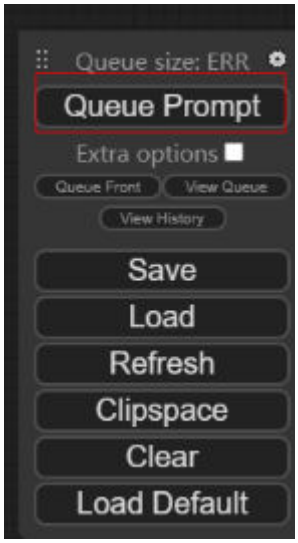
3. 根据上面checkpoint的箭头，对新的NPU的checkpoint进行规划，如下图。

图 4-104 规划 checkpoint



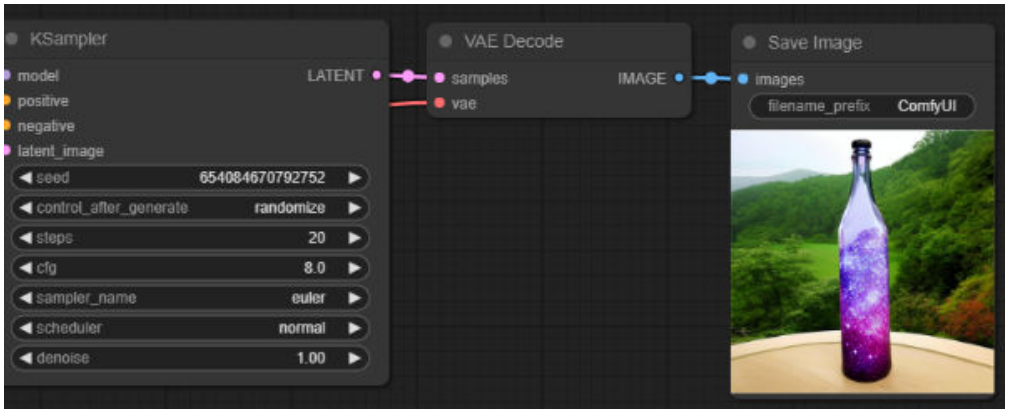
4. 在ckpt_name中选择要使用的权重文件，device_id为要使用的NPU卡号，单击“Queue Prompt”加入推理队列进行推理，如下图。

图 4-105 加入推理队列



成功之后结果如下图。

图 4-106 推理成功



首次加载或切换模型进行推理时，需要加载模型并进行相关的初始化工作，首次推理时间较长，请耐心等待。

4.21 SD1.5 基于 DevServer 适配 PyTorch NPU Finetune 训练指导（6.3.904）

Stable Diffusion（简称SD）是一种基于Latent Diffusion（潜在扩散）模型，应用于文生图场景。对于输入的文字，它将会通过一个文本编码器将其转换为文本嵌入，然后和一个随机高斯噪声，一起输入到U-Net网络中进行不断去噪。在经过多次迭代后，最终模型将输出和文字相关的图像。

SD1.5 Finetune是指在已经训练好的SD1.5模型基础上，使用新的数据集进行微调（fine-tuning）以优化模型性能的过程。

本文档主要介绍如何利用训练框架PyTorch_npu+华为自研Ascend Snt9B硬件，对Stable Diffusion模型下不同数据集进行高性能训练调优，同时启用多卡作业方式提升训练速度，完成SD1.5 Finetune训练。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B。

表 4-40 环境要求

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取软件和镜像

表 4-41 获取软件和镜像

分类	名称	获取路径
插件代码包	ascendcloud-aigc-6.3.904-xxx.tar.gz 文件名中的xxx表示具体的时间戳，以包的实际时间为准。	获取路径：**Support-E网站**。 **说明** 如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	SWR上拉取

Step1 检查环境

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

   说明

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

2. SSH登录机器后，检查NPU卡状态。运行如下命令，返回NPU设备信息。

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查是否安装docker。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 启动镜像

1. 获取基础镜像。建议使用官方提供的镜像。镜像地址{image_url}参考**表4-41**。

   ```
   docker pull {image_url}
   ```

2. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像地址"
// 启动一个容器去运行镜像
docker run -itd \
        --device=/dev/davinci0 \
        --device=/dev/davinci1 \
        --device=/dev/davinci2 \
        --device=/dev/davinci3 \
        --device=/dev/davinci4 \
        --device=/dev/davinci5 \
        --device=/dev/davinci6 \
        --device=/dev/davinci7 \
        --device=/dev/davinci_manager \
        --device=/dev/devmm_svm \
        --device=/dev/hisi_hdc \
        -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
        -v /usr/local/dcmi:/usr/local/dcmi \
        -v /etc/ascend_install.info:/etc/ascend_install.info \
        -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
        -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
        --shm-size 32g \
        --net=bridge \
        -v ${work_dir}:${container_work_dir} \
        --name ${container_name} \
        ${image_name} bash
```

参数说明：

- work_dir：工作目录，目录下存放着训练所需代码、数据等文件。
- container_work_dir：容器工作目录，一般同work_dir。
- container_name：自定义容器名。
- image_name：容器镜像的名称。

3. 进入容器。需要将${container_name}替换为实际的容器名称。

```
docker exec -it ${container_name} bash
```

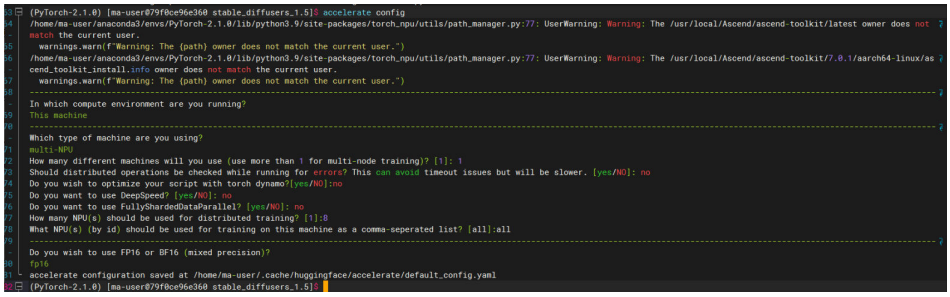
Step3 获取 SD1.5 插件代码包并安装依赖

1. 将下载的SD1.5插件代码包ascendcloud-aigc-xxx-xxx.tar.gz文件，上传到容器的/home/ma-user/目录下，解压并安装相关依赖。插件代码包获取路径参见**表4-41**。

```
mkdir -p /home/ma-user/stable_diffusers_1.5    #创建stable_diffusers_1.5目录
cd /home/ma-user/stable_diffusers_1.5          #进入stable_diffusers_1.5目录

tar -zxvf ascendcloud-aigc-*.tar.gz
tar -zxvf ascendcloud-aigc-poc-stable_diffusers_1.5.tar.gz
rm -rf ascendcloud-aigc-xxx-xxx

pip install -r requirements.txt    #安装依赖
```

2. 启动前配置。有两种方式修改配置文件：
   - 方式一：可以参考解压出来的default_config.yaml或者deepspeed_default_config.yaml文件，再通过在启动脚本命令中增加--config_file=xxx.yaml参数来指定其为配置文件。
   - 方式二：通过命令accelerate config进行配置，如下图所示。

图 4-107 通过命令 accelerate config 进行配置



3. （可选）文件替换。

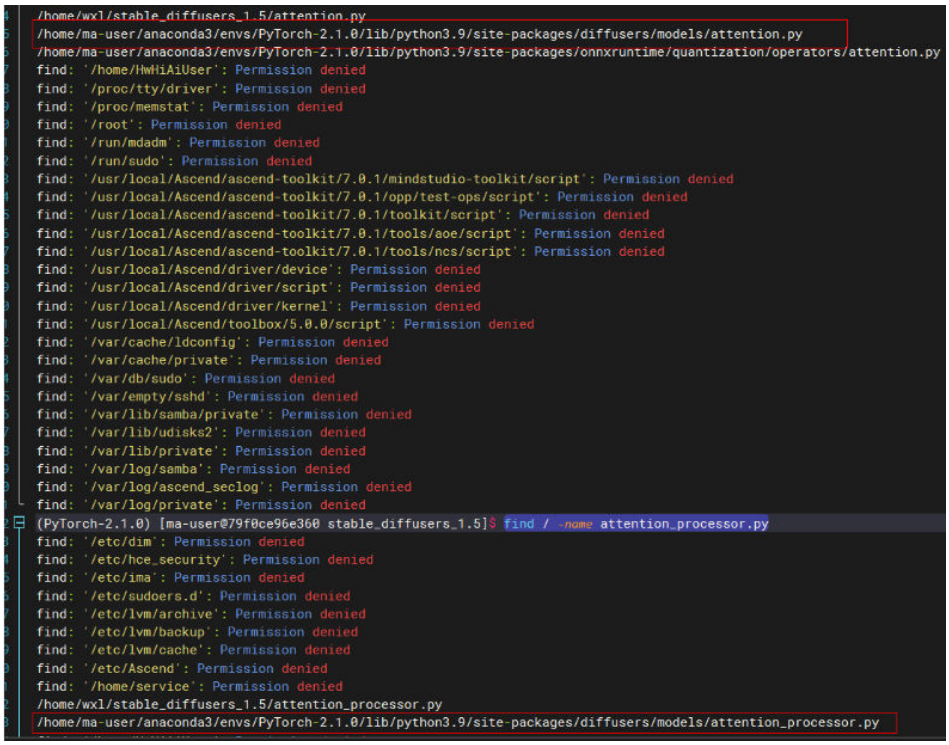
   因增加nfa和使用npu_geglu算子（用于训练和推理加速），将diffusers源码包中的attention.py和attention_processor.py替换成代码包中对应的文件。

图 4-108 文件替换

可以使用find命令来查找diffusers源码包位置。

```
find / -name attention.py
find / -name attention_processor.py
```

图 4-109 查找 diffusers 源码包位置



找到具体位置后可以cp替换，替换前可对diffusers原始文件做备份，如果没有备份则可以通过删除diffusers包重新安装的方式获取原始文件。

4. 执行bash stable_diffusers_train.sh。

```
bash stable_diffusers_train.sh
```

Step4 下载模型和数据集

数据集下载地址：**https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions**。

启动脚本前的两个声明为本次训练的模型和数据集，第一次执行程序时若本地没有模型和数据集，会自动下载。但由于lambdalabs/pokemon-blip-captions数据集下载现在需要登录HuggingFace账号，请先下载数据集到本地，再挂载到对应目录。

```
export MODEL_NAME="runwayml/stable-diffusion-v1-5"
export DATASET_NAME="lambdalabs/pokemon-blip-captions"
```

Step5 启动训练服务

train_text_to_image_0304.py是训练的核心代码，通过stable_diffusers_train.sh来启动。

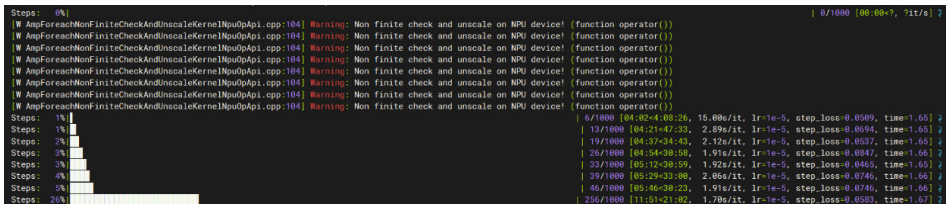
```
sh stable_diffusers_train.sh
```

说明

如果启动前配置采用的是**•可以参考解压出来的default_config...**方式指定配置文件，就是在此stable_diffusers_train.sh脚本中增加--config_file=xxx.yaml参数。

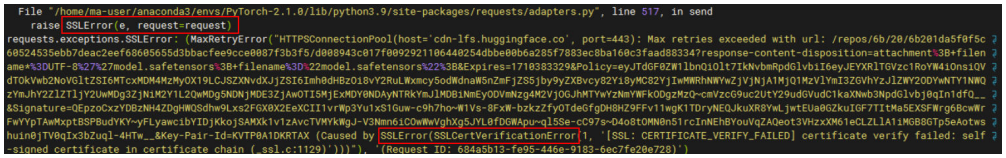
刚开始会报一些Warning，可忽略。正常启动如下图所示，出现Steps: 1%字样。

图 4-110 启动服务



如果启动过程中报SSL相关错误，如下图所示。

图 4-111 启动过程中报 SSL 相关错误



请修改相应路径下的/home/ma-user/anaconda3/envs/PyTorch-2.1.0/lib/python3.9/site-packages/requests/sessions.py文件，将self.verify的值由True改成False，如下图所示。

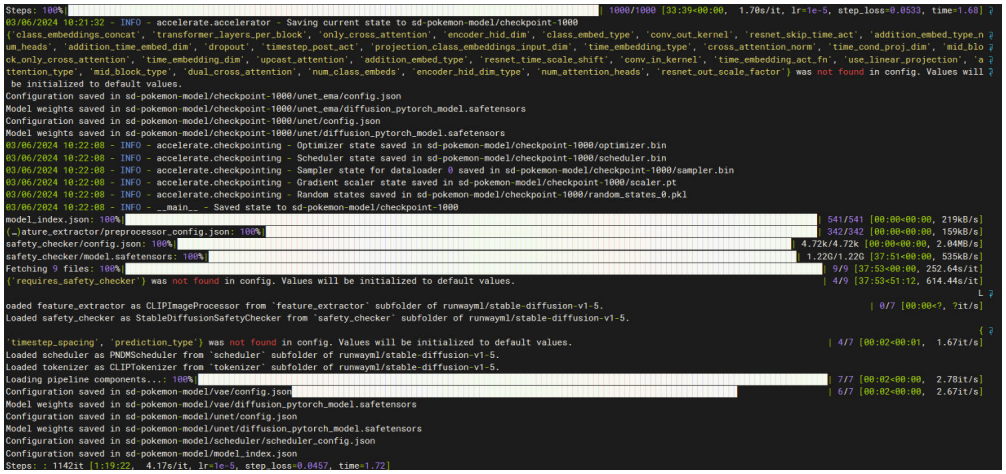
图 4-112 修改 self.verify 参数值

```
#: SSL Verification default.
#: Defaults to `True`, requiring requests to verify the TLS certificate at the
#: remote end.
#: If verify is set to `False`, requests will accept any TLS certificate
#: presented by the server, and will ignore hostname mismatches and/or
#: expired certificates, which will make your application vulnerable to
#: man-in-the-middle (MitM) attacks.
#: Only set this to `False` for testing.
self.verify = False
```

Step6 保存并查看训练结果

正常运行完成训练，会显示如下内容。

图 4-113 训练完成



精度一般问题不大，step_loss都是一个较小值。

训练过程中，训练日志会在最后的Rank节点打印。可以使用可视化工具**TrainingLogParser**查看loss收敛情况。

其它注意事项

- 默认500step保存一个checkpoint，可以通过在启动脚本里添加参数--checkpointing_steps=num修改。
- 若显存较低可以调整batch_size保证正常运行，改为8或者更小。
- 本次训练step为1000，训练时间较长，可以改为500。
- 如开启deepspeed训练时，需要设置参数checkpointing_steps>max_train_steps（严格大于），否则会报错。

4.22 SDXL Diffusers 框架基于 Devserver 适配 PyTorch NPU 推理指导（6.3.902）

本文档主要介绍如何在ModelArts Lite的DevServer环境中部署Stable Diffusion的Diffusers框架，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署Diffusers框架用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推理部署推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

获取软件

获取插件代码包ascendcloud-aigc-6.3.902-*.tar.gz文件。获取路径：**Support网站**。

说明

如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。

ascendcloud-aigc-6.3.902-*.tar.gz文件名中的*表示具体的时间戳，以包名的实际时间为准。

Step1 准备环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 检查环境。

   a. SSH登录机器后，检查NPU设备检查。运行如下命令，返回NPU设备信息。
   ```
   npu-smi info
   ```
   如出现错误，可能是机器上的NPU设备没有正常安装，或者NPU镜像被其他容器挂载。请先正常**安装固件和驱动**，或释放被挂载的NPU。

   b. 检查docker是否安装。
   ```
   docker -v   #检查docker是否安装
   ```
   如尚未安装，运行以下命令安装docker。
   ```
   yum install -y docker-engine.aarch64 docker-engine-selinux.noarch docker-runc.aarch64
   ```

   c. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
   ```
   sysctl -p | grep net.ipv4.ip_forward
   ```
   如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
   ```
   sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
   sysctl -p | grep net.ipv4.ip_forward
   ```

3. 获取基础镜像。建议使用官方提供的镜像部署推理服务。

   镜像地址{image_url}为：

   西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240312154948-219655b

   ```
   docker pull {image_url}
   ```

4. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。
   ```
   docker run -itd \
   --name sdxl-diffusers \
   -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
   -p 8443:8443 \
   -v /etc/localtime:/etc/localtime \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   --shm-size 60g \
   --device=/dev/davinci_manager \
   --device=/dev/hisi_hdc \
   --device=/dev/devmm_svm \
   --device=/dev/davinci3 \
   --network=bridge \
   ${image_name} bash
   ```

   参数说明：

   - --name ${conta iner_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如sdxl-diffusers。
   - --device=/dev/davinci3：挂载主机的/dev/davinci3到容器的/dev/davinci3。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
   - ${image_name} 代表 ${image_name}。

5. 进入容器。需要将${container_name}替换为实际的容器名称，例如：sdxl-diffusers。
   ```
   docker exec -it ${container_name} bash
   ```

Step2 安装依赖和模型包

1. 安装Diffusers相关依赖。
   ```
   pip install -i https://pypi.tuna.tsinghua.edu.cn/simple diffusers bottle invisible_watermark transformers accelerate safetensors
   ```

2. 获取SDXL模型包并解压到/home/ma-user目录下。提供2种模型包下载方式。

- 模型包直接下载（如果不能访问HuggingFace官网，推荐此方式）

  下载到容器/home/ma-user目录下后，解压。

  ```
  cd /home/ma-user/
  wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/stable-diffusion-xl-model.tar.gz
  tar -zxvf stable-diffusion-xl-model.tar.gz
  rm -rf stable-diffusion-xl-model.tar.gz
  ```

- 也可以从HuggingFace官网下载到本地后，通过docker cp命令复制到容器中/home/ma-user目录下，如下图所示。

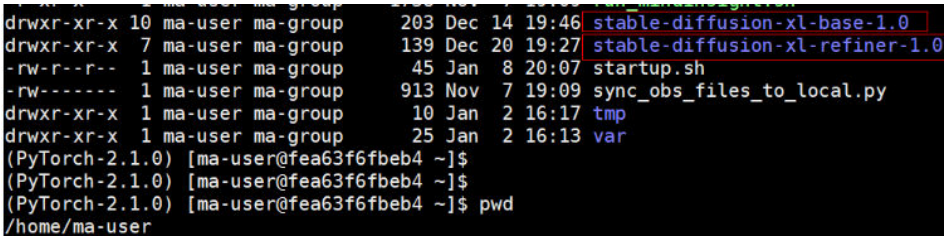
  在线下载地址：

  https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main

  https://huggingface.co/stabilityai/stable-diffusion-xl-refiner-1.0/tree/main

  由于本实例采用的都是FP16的模型，相应模型建议都只下载FP16的，节约下载和传送时间。

  图 4-114 下载 SDXL 模型包并解压

3. 获取controlnet模型包并解压到/home/ma-user目录下。提供2种模型包下载方式。

   - 模型包直接下载（如果不能访问HuggingFace官网，推荐此方式）

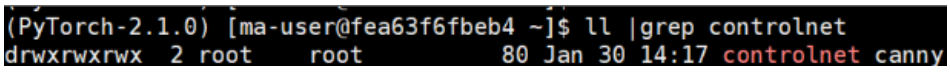
     下载到容器/home/ma-user目录下后，解压。

     ```
     cd /home/ma-user/
     wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/controlnet_canny.zip
     unzip controlnet_canny.zip
     ```

   - 也可以从HuggingFace官网下载到本地后，通过docker cp命令复制到容器中/home/ma-user目录下。

     在线下载地址：**https://huggingface.co/diffusers/controlnet-canny-sdxl-1.0/tree/main**

     图 4-115 下载 controlnet 模型包并解压

4. 安装插件代码包。

   a. 将获取到的插件代码包ascendcloud-aigc-6.3.902-*.tar.gz文件上传到容器的/home/ma-user/temp目录下。获取路径：**Support网站**。

   b. 解压插件代码包ascendcloud-aigc-6.3.902-*到/home/ma-user/temp目录下。

   ```
   cd /home/ma-user/temp
   tar -zxvf ascendcloud-aigc-6.3.902-20240205145924.tar.gz  #解压
   ```

   c. 将获取到的ascendcloud-aigc-extensions-diffusers.tar.gz包复制到/home/ma-user下后解压。

```
docker cp ascendcloud-aigc-extensions-diffusers.tar.gz sdxl-diffusers:/home/ma-user/
tar -zxvf ascendcloud-aigc-extensions-diffusers.tar.gz
```

Step3 运行并验证 SDXL 模型

1. 首先在容器中运行命令。
   ```
   source /usr/local/Ascend/ascend-toolkit/set_env.sh
   ```
2. 在/home/ma-user目录下已经存在infer_server.py脚本文件，启动infer_server.py命令如下。
   ```
   python infer_server.py
   ```

   图 4-116 启动脚本

   ```
   (PyTorch-2.1.0) [ma-user@fea63f6fbeb4 ~]$ python infer_server.py
   Loading pipeline components...: 100%|| 7/7 [00:02<00:00,  3.33it/s]
   Loading pipeline components...: 100%|| 5/5 [00:00<00:00,  7.41it/s]
     0%|| 0/32 [00:00<?, ?it/s]
   100%|| 32/32 [00:22<00:00,  1.42it/s]
     0%|| 0/8 [00:00<?, ?it/s]
   100%|| 8/8 [00:17<00:00,  2.16s/it]
   Bottle v0.12.25 server starting up (using WSGIRefServer())...
   Listening on http://0.0.0.0:8443/
   Hit Ctrl-C to quit.
   ```

3. 在宿主机上另外打开一个终端，使用curl命令发送请求。完整的请求参数请参考**表4-42**。
   ```
   curl -kv -X POST localhost:8443/ -H "Content-Type: application/json" -d '{"prompt":"ultrarealistic shot of a furry blue bird"}'
   ```
   服务端打印如下信息，表示发送请求成功。

   图 4-117 发送请求

   ```
   >prompt: ultrarealistic shot of a furry blue bird
   100%|| 32/32 [00:07<00:00,  4.40it/s]
   100%|| 8/8 [00:02<00:00,  3.57it/s]
   ms_pipeline_time: 11.121 s
   172.17.0.1 - - [04/Feb/2024 11:05:46] "POST / HTTP/1.1" 200 1665156
   ```

   客户端返回图像的base64编码。

   图 4-118 图像的 base64 编码

4. 将客户端返回的base64编码转换为图片。
   ```
   from PIL import Image
   from io import BytesIO
   import base64

   def base64_to_image(base64_str):
       image = base64.b64decode(base64_str, altchars=None, validate=False)
       image = BytesIO(image)
       image = Image.open(image)
       image.save(“./out_put_image.png”)
   ```

Step4 运行并验证带 controlnet 的模型

1. 首先下载一个默认输入文件。

   https://huggingface.co/lllyasviel/sd-controlnet-canny/blob/main/images/bird_canny.png

2. 文件下载后重命名为canny_input_bird.png，然后复制到容器/home/ma-user目录下，在宿主机上的执行命令如下。
```
mv bird_canny.png canny_input_bird.png
chmod 777 canny_input_bird.png
docker cp canny_input_bird.png sdxl-diffusers:/home/ma-user/
```
3. 在/home/ma-user目录下已经存在infer_server_with_controlnet.py脚本文件，运行带controlnet的sdxl，运行命令如下。
```
python infer_server_with_controlnet.py
```
4. 在宿主机上另外打开一个终端，使用curl命令发送请求。完整的请求参数请参考**表4-42**。
```
curl -kv -X POST localhost:8443/ -H "Content-Type: application/json" -d '{"prompt":"ultrarealistic shot of a furry blue bird"}'
```
服务端打印如下信息，表示发送请求成功。

带controlnet时，可以读取本地图片得到输入参数。
```
from diffusers.utils import load_image
from io import BytesIO
import base64

def image_to_base64(img_path):
    image = load_image(img_path)
    buffered = BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue())
```

附录 1：请求参数表

使用curl命令发送请求的请求参数表如下。

表 4-42 请求参数列表

参数	说明
prompt	正向文本，必选
negative_prompt	负向文本，非必选
height	图像高度，非必选
width	图像宽度，非必选
num_inference_steps	对图片进行噪声优化的次数，非必选
denoising_end	二阶段去噪，非必选
refiner_switch	refiner模型开关，是否开启refiner，非必选
seed	添加噪音的随机数种子，非必选
image_path	带controlnet时需要，此时image_path需要赋值null，传入图片的base64编码值，非必选
image_base64	带controlnet时需要，和image_path二选一，传入图片的base64编码值，非必选

附录 2：Dockerfile

基于Dockerfile可以方便的构建完整可运行的自定义镜像，在宿主机创建一个空的目录，然后vi Dockerfile将上面内容复制进去，然后参考4在创建目录中下载华为插件代码包后，执行如下docker构建命令。

```
docker build -t sdxl-diffusers:0.0.1 .
```

Dockerfile文件内容如下。

```
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240312154948-219655b

RUN wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/stable-diffusion-xl-model.tar.gz && \
tar -zxvf stable-diffusion-xl-model.tar.gz && \
rm -rf stable-diffusion-xl-model.tar.gz

RUN wget https://llm-mindspore.obs.cn-southwest-2.myhuaweicloud.com/ascend-poc/controlnet_canny.zip && \
unzip controlnet_canny.zip &&\
rm -rf controlnet_canny.zip

RUN mkdir /home/ma-user/temp
COPY --chown=ma-user:ma-group ascendcloud-aigc-6.3.902-20240205145924.tar.gz /home/ma-user/temp/

RUN cd /home/ma-user/temp &&\
tar -zxvf ascendcloud-aigc-6.3.902-20240205145924.tar.gz &&\
cp ascendcloud-aigc-extensions-diffusers.tar.gz /home/ma-user &&\
cd /home/ma-user && tar -zxvf ascendcloud-aigc-extensions-diffusers.tar.gz &&\
rm -rf /home/ma-user/temp && rm -rf ascendcloud-aigc-extensions-diffusers.tar.gz

RUN pip install diffusers bottle invisible_watermark transformers accelerate safetensors

CMD source /usr/local/Ascend/ascend-toolkit/set_env.sh && python /home/ma-user/infer_server.py
```

4.23 SDXL WebUI 基于 Devserver 适配 PyTorch NPU 推理指导（6.3.902）

本文档主要介绍如何在ModelArts Lite的DevServer环境中部署Stable Diffusion的WebUI套件，使用NPU卡进行推理。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署Stable Diffusion WebUI套件用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推理部署推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

获取软件

获取插件代码包ascendcloud-aigc-6.3.902-*.tar.gz文件。获取路径：**Support网站**。

说明

如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。

ascendcloud-aigc-6.3.902-*.tar.gz文件名中的*表示具体的时间戳，以包名的实际时间为准。

Step1 准备环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 检查环境。

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

   ```
   npu-smi info
   ```

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

   b. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

3. 获取基础镜像。建议使用官方提供的镜像部署推理服务。

   镜像地址{image_url}为：

   西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240312154948-219655b

   ```
   docker pull {image_url}
   ```

4. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

   ```
   docker run -itd \
   --name sdwebui \
   -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
   -p 8183:8183 \
   -v /etc/localtime:/etc/localtime \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
   --shm-size 60g \
   --device=/dev/davinci_manager \
   --device=/dev/hisi_hdc \
   --device=/dev/devmm_svm \
   --device=/dev/davinci3 \
   --network=bridge \
   ${image_name} bash
   ```

   参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称，例如sdxl-diffusers。
- --device=/dev/davinci3：挂载主机的/dev/davinci3到容器的/dev/davinci3。可以使用npu-smi info查看空闲卡号，修改davinci后数字可以更改挂载卡。
- ${image_name} 代表 ${image_name}。

5. 进入容器。需要将${container_name}替换为实际的容器名称，例如：sdwebui。
```
docker exec -it ${container_name} bash
```

Step2 下载软件包

1. 下载stable-diffusion-webui-1.7.0.zip文件后解压，重命名为stable-diffusion-webui，然后拷贝到容器/home/ma-user目录下。

   sdwebui 1.7.0版本软件包的官网下载地址：**https://github.com/AUTOMATIC1111/stable-diffusion-webui/tree/v1.7.0**
```
docker cp stable-diffusion-webui sdwebui:/home/ma-user/
```
2. 修改文件夹权限。启动容器时默认用户为ma-user用户，在使用其他属组如root用户上传的数据和文件时，可能会存在权限不足的问题。
```
# 修改文件夹权限（注意：重新启动一个终端，使用root用户登录容器修改文件权限，修改完后关闭终端。）
docker exec -it --user root sdwebui bash
chown -R ma-user:ma-group stable-diffusion-webui
```
3. 下载SD基础模型，并拷贝到容器/home/ma-user/stable-diffusion-webui/models/Stable-diffusion目录下。

   SD基础模型的官网下载地址。

   https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors

   https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.safetensors
```
docker cp sd_xl_base_1.0.safetensors sdwebui:/home/ma-user/stable-diffusion-webui/models/Stable-diffusion/
docker cp v1-5-pruned-emaonly.safetensors sdwebui:/home/ma-user/stable-diffusion-webui/models/Stable-diffusion/

# 修改文件夹权限
docker exec -it --user root sdwebui bash
chown -R ma-user:ma-group stable-diffusion-webui/models/Stable-diffusion/
```
4. 下载controlnet插件sd-webui-controlnet-main.zip文件后解压，重命名为sd-webui-controlnet，然后拷贝到容器stable-diffusion-webui/extensions/目录下。

   controlnet插件的官网下载地址：**https://github.com/Mikubill/sd-webui-controlnet**。
```
docker cp sd-webui-controlnet sdwebui:/home/ma-user/stable-diffusion-webui/extensions/
# 修改文件夹权限
docker exec -it --user root sdwebui bash
chown -R ma-user:ma-group stable-diffusion-webui/extensions/
```
5. 根据需要下载controlnet模型，放在/home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models目录下。
```
docker cp control_v11p_sd15_canny.pth sdwebui:/home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/
docker cp control_v11p_sd15_canny.yaml sdwebui:/home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/
docker cp diffusers_xl_canny_mid.safetensors sdwebui:/home/ma-user/stable-diffusion-webui/extensions/sd-webui-controlnet/models/

# 修改文件夹权限
docker exec -it --user root sdwebui bash
chown -R ma-user:ma-group stable-diffusion-webui/extensions/sd-webui-controlnet/models/
```

controlnet模型官网下载地址：

https://huggingface.co/lllyasviel/ControlNet-v1-1/tree/main

https://huggingface.co/lllyasviel/sd_control_collection/tree/main

选择下载sd1.5 canny：

https://huggingface.co/lllyasviel/ControlNet-v1-1/blob/main/control_v11p_sd15_canny.pth

https://huggingface.co/lllyasviel/ControlNet-v1-1/blob/main/control_v11p_sd15_canny.yaml

选择下载sdxl canny：

https://huggingface.co/lllyasviel/sd_control_collection/blob/main/diffusers_xl_canny_mid.safetensors

6. 安装插件代码包。
   a. 将获取到的插件代码包ascendcloud-aigc-6.3.902-*.tar.gz文件上传到容器的/home/ma-user/temp目录下。获取路径：**Support网站**。
   b. 解压插件代码包ascendcloud-aigc-6.3.902-*到/home/ma-user/temp目录下。
      ```
      tar -zxvf ascendcloud-aigc-6.3.902-*.tar.gz  #解压
      ```
   c. 再解压ascendcloud-aigc-extensions-webui.tar.gz
      ```
      tar -zxvf ascendcloud-aigc-extensions-webui.tar.gz
      ```
   d. 拷贝NPU插件代码webui_npu_extension拷贝到stable-diffusion-webui/extensions/目录下。
      ```
      cp -rf webui_npu_extension /home/ma-user/stable-diffusion-webui/extensions/
      ```
   e. 拷贝safety-checker代码到/home/ma-user/stable-diffusion-webui/modules/目录下。
      ```
      cp third_paties/stable-diffusion-webui/safety_checker.py /home/ma-user/stable-diffusion-webui/modules/
      ```
   f. 然后在/home/ma-user/stable-diffusion-webui/modules/目录下，修改processing.py文件。
      ```
      cd /home/ma-user/stable-diffusion-webui/modules
      sed -i '17 i\from modules.safety_checker import check_safety' processing.py
      sed -i '621 i\        x_checked_image = sample.cpu().unsqueeze(0).permute(0, 2, 3, 1).numpy()' processing.py
      sed -i '622 i\        x_checked_image, has_nsfw_concept = check_safety(x_checked_image)' processing.py
      sed -i '623 i\        sample = torch.tensor(x_checked_image).permute(0, 3, 1, 2).squeeze(0).to(sample.device)' processing.py
      sed -i 's#\r##g' processing.py
      ```
7. 下载safety-checker模型包。

   safety-checker的官网下载地址：**https://huggingface.co/CompVis/stable-diffusion-safety-checker/tree/main**

   在宿主机当前目录下创建CompVis/stable-diffusion-safety-checker目录，然后下载所有文件，如下图所示。

   图 4-119 下载文件

   ```
   [root@devserver-bms-28310a65-tmp1228 stable-diffusion-safety-checker]# ll
   total 1187584
   -rwx------ 1 root root       4549 Jan 17 14:16 config.json
   -rwx------ 1 root root        342 Jan 17 14:16 preprocessor_config.json
   -rwx------ 1 root root 1216067303 Jan 17 14:16 pytorch_model.bin
   ```

   然后将CompVis目录整个拷贝到/home/ma-user/stable-diffusion-webui目录下。

```
docker cp CompVis sdwebui:/home/ma-user/stable-diffusion-webui/

# 修改文件夹权限
docker exec -it --user root sdwebui bash
chown -R ma-user:ma-group stable-diffusion-webui/CompVis
```

Step3 安装依赖

1. 在容器中执行如下命令，安装pip依赖。

```
cd /home/ma-user/stable-diffusion-webui
pip install --upgrade pip
pip install -r requirements.txt --no-deps
pip install lightning_utilities torchmetrics gradio_client matplotlib pydantic aiofiles starlette ffmpy
pydub uvicorn orjson semantic_version altair antlr4-python3-runtime==4.8.0 ftfy regex
pytorch_lightning==1.6.5 gitdb trampoline clip aenum facexlib torch==2.1.0 python-multipart gdown
pip install -r requirements_versions.txt
pip install httpx==0.24.1
pip install diffusers
```

2. 安装Stable Diffusion依赖。

   a. 下载stablediffusion-main.zip文件解压后，重命名为stable-diffusion-stability-ai，然后拷贝到容器stable-diffusion-webui/repositories/目录下。stablediffusion-main.zip文件的官网下载地址：**https://github.com/Stability-AI/stablediffusion**。

   ```
   docker cp stable-diffusion-stability-ai sdwebui:/home/ma-user/stable-diffusion-webui/
   repositories/
   ```

   说明

   如果stable-diffusion-webui/repositories/目录不存在，需要通过mkdir创建。

   b. 下载generative-models-main.zip文件解压后，重命名为generative-models，然后拷贝到容器stable-diffusion-webui/repositories/目录下。generative-models-main.zip文件的官网下载地址：**https://github.com/Stability-AI/generative-models.git**。

   ```
   docker cp generative-models sdwebui:/home/ma-user/stable-diffusion-webui/repositories/
   ```

   c. 下载k-diffusion-master.zip文件解压后，重命名为k-diffusion，然后拷贝到容器stable-diffusion-webui/repositories/目录下。k-diffusion-master.zip文件的官网下载地址：**https://github.com/Stability-AI/k-diffusion**。

   ```
   docker cp k-diffusion sdwebui:/home/ma-user/stable-diffusion-webui/repositories/
   # 修改文件夹权限
   docker exec -it --user root sdwebui bash
   chown -R ma-user:ma-group stable-diffusion-webui/repositories/
   ```

3. 安装vaeapprox-sdxl.pt。

   下载vaeapprox-sdxl.pt文件后，拷贝到容器/home/ma-user/stable-diffusion-webui/models/VAE-approx/目录下。vaeapprox-sdxl.pt的官网下载地址：**https://github.com/AUTOMATIC1111/stable-diffusion-webui/releases/tag/v1.0.0-pre**。

   ```
   docker cp vaeapprox-sdxl.pt sdwebui:/home/ma-user/stable-diffusion-webui/models/VAE-approx/
   # 修改文件夹权限
   docker exec -it --user root sdwebui bash
   chown -R ma-user:ma-group stable-diffusion-webui/models/VAE-approx/
   ```

Step4 运行并验证 SDXL 模型

1. 首先在容器中运行命令。

```
cd /home/ma-user/stable-diffusion-webui
source /usr/local/Ascend/ascend-toolkit/set_env.sh
```

2. 在/home/ma-user目录下已经存在launch.py脚本文件，启动launch.py命令如下。

```
python3 launch.py --skip-torch-cuda-test --port 8183 --enable-insecure-extension-access --listen --log-startup --disable-safe-unpickle --skip-prepare-environment --api
```

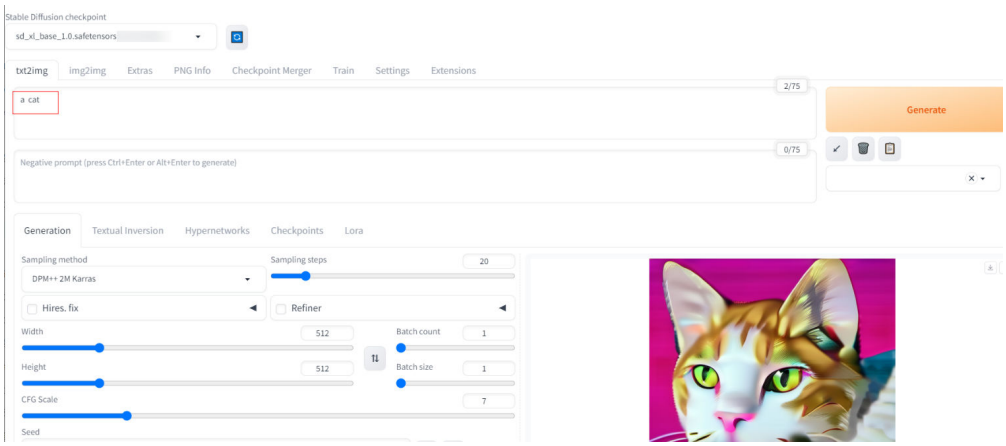
启动成功后，打印如下信息。

图 4-120 启动成功

```
Loading weights [31e35c80fc] from /home/ma-user/stable-diffusion-webui/models/Stable-diffusion/sd_xl_base_1.0.safetensors
reload hypernetworks: done in 0.010s
initialize extra networks: done in 0.016s
scripts before_ui_callback: done in 0.002s
2024-02-06 14:16:05,743 INFO: import AscendPlugin
fatal: not a git repository (or any of the parent directories): .git
fatal: not a git repository (or any of the parent directories): .git
create ui: done in 1.362s
Running on local URL:  http://0.0.0.0:8183

To create a public link, set `share=True` in `launch()`.
gradio launch: done in 1.205s
add APIs: done in 0.818s
  app_started_callback:
  lora_script.py: done in 0.001s
  api.py: done in 0.004s
Startup time: 21.9s (import torch: 7.0s, import gradio: 2.1s, setup paths: 2.1s, initialize shared: 0.1s, other imports: 5.5s, load scr
pts: 1.6s, create ui: 1.4s, gradio launch: 1.2s, add APIs: 0.8s).
Creating model from config: /home/ma-user/stable-diffusion-webui/repositories/generative-models/configs/inference/sd_xl_base.yaml
Applying attention optimization: InvokeAI... done.
Model loaded in 39.5s (load weights from disk: 3.9s, create model: 2.0s, apply weights to model: 32.7s, apply half(): 0.1s, move model
o device: 0.2s, calculate empty prompt: 0.4s).
```

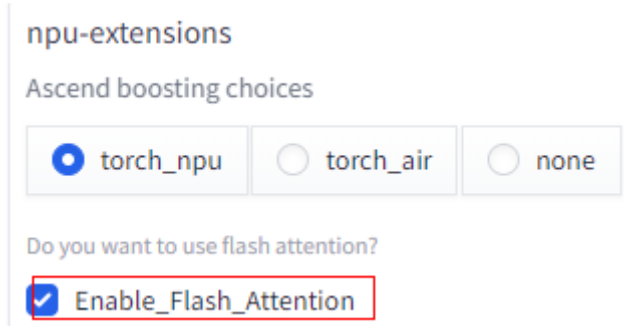
3. 使用http://{宿主机ip}:8183 可以访问前端页面，通过输入文字生成图片。

图 4-121 文生图



注意开启fa优化按钮。

图 4-122 开启 fa 优化按钮

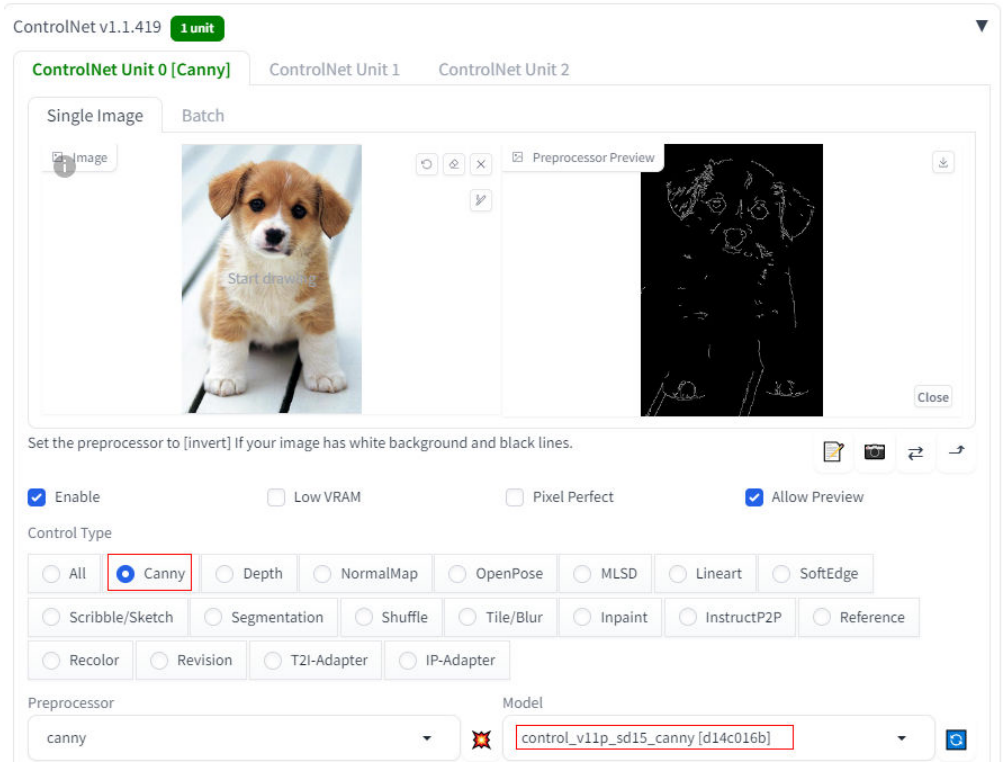


如果使用涉黄文字，输出的图片会返回黑图，用于验证safety-checker功能。同时，服务端会打印如下信息。

图 4-123 服务端返回信息

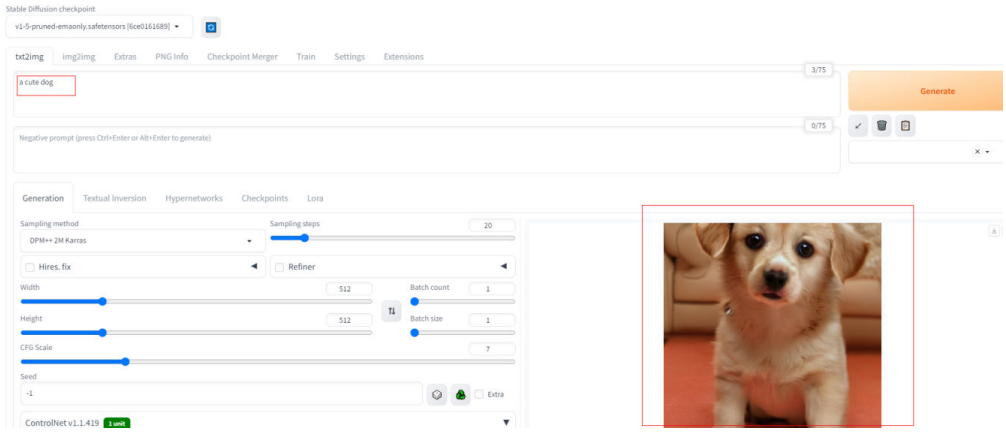
4. 带controlnet运行，默认使用canny。

图 4-124 带 controlnet 运行



可以观察到输出的图片与canny输入图片很相近，坐姿和样子比较符合，如下图所示。

图 4-125 文生图



5. 使用后台API调用文生图接口。

```
curl -kv -X POST localhost:8183/sdapi/v1/txt2img -H "Content-Type: application/json" -d
'{"prompt":"a dog"}'
```

客户端返回图像的base64编码，将base64编码再转换为图片，转换代码如下。

```
from PIL import Image
from io import BytesIO
import base64

def base64_to_image(base64_str):
    image = base64.b64decode(base64_str, altchars=None, validate=False)
    image = BytesIO(image)
    image = Image.open(image)
    image.save("./out_put_image.png")
```

4.24 Open-Clip 基于 DevServer 适配 PyTorch NPU 训练指导

Open-Clip广泛应用于AIGC和多模态视频编码器的训练。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾NPU计算资源开展Open-clip训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 4-43 环境要求

模型	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取镜像

表 4-44 获取镜像

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	从SWR拉取。

获取软件

本教程使用的是Open-clip源码包。

昇腾适配过程通过修改训练脚本方式实现，不涉及其他软件获取。

Step1 准备环境

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

   说明

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

2. 检查环境。

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

   b. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取镜像

获取基础镜像。建议使用官方提供的镜像部署推理服务。镜像地址{image_url}参考**获取镜像**。

```
docker pull ${image_url}
```

Step3 启动容器

启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

```
docker run -itd \
    --device=/dev/davinci0 \
    --device=/dev/davinci_manager \
    --device=/dev/devmm_svm \
    --device=/dev/hisi_hdc \
    -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
    -v /usr/local/dcmi:/usr/local/dcmi \
    -v /etc/ascend_install.info:/etc/ascend_install.info \
    -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
    -v /etc/localtime:/etc/localtime \
    -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
    --shm-size 32g \
    --net=bridge \
    -v ${work_dir}:${container_work_dir} \
```

```
    --name ${container_name} \
    ${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- ${image_name} 代表镜像地址。

通过容器名称进入容器中。使用默认用户ma-user启动容器。

```
docker exec -it ${container_name} bash
```

Step4 下载并安装 Open-clip 源码包

1. 从官网下载Open-clip源码包。

   ```
   git clone https://github.com/mlfoundations/open_clip.git
   cd open_clip
   git reset --hard 37b2c6b321ee697df4c709ca95d6dc849fc7d214
   ```

   37b2c6b321ee697df4c709ca95d6dc849fc7d214是commit号。
2. 复制Open-clip源码包到容器/home/ma-user目录下。

   ```
   docker cp open_clip open-clip:/home/ma-user/
   ```
3. 修改文件夹权限（注意：此处需要重新启动一个终端，使用root用户登录容器，修改文件夹权限，修改完后关闭这个终端。）

   ```
   docker exec -it --user root open-clip bash
   chown -R ma-user:ma-group open_clip
   exit
   ```
4. 在步骤2打开的终端中，使用默认用户ma-user安装源码。

   ```
   cd open_clip
   make install
   ```
5. 在步骤2打开的终端中，使用默认用户ma-user安装依赖。

   ```
   pip install -r requirements-training.txt
   pip install -r requirements-test.txt
   pip install tensorboard
   ```

Step5 获取训练数据集

使用img2dataset工具下载数据集。首先需要在容器安装img2dataset，安装命令如下。

```
pip install img2dataset
```

参考**官方指导**下载开源mscoco数据集。

```
#下载metadata
wget https://huggingface.co/datasets/ChristophSchuhmann/MS_COCO_2017_URL_TEXT/resolve/main/mscoco.parquet
#使用img2dataset工具下载数据集
img2dataset --url_list mscoco.parquet --input_format "parquet"\
        --url_col "URL" --caption_col "TEXT" --output_format webdataset\
```



```
    --output_folder mscoco --processes_count 16 --thread_count 64 --image_size 256\
     --enable_wandb True
```

Step6 训练 Open clip 模型

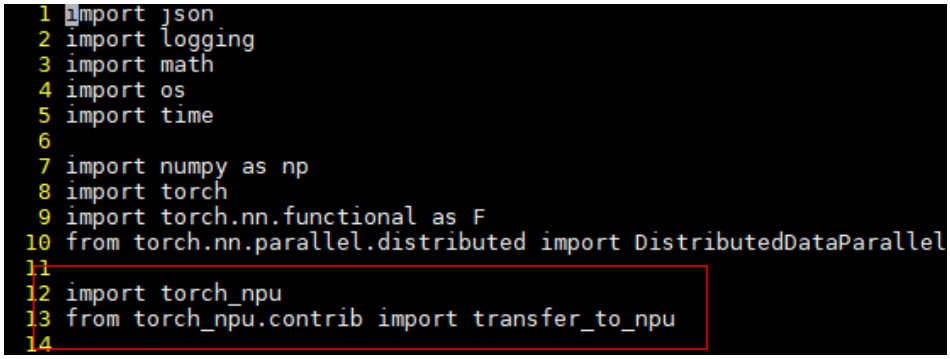
1. 适配昇腾代码。

   在目录/home/ma-user/open_clip/src/training下，修改main.py文件，在第10行添加如下代码。

   ```
   import torch_npu
   from torch_npu.contrib import transfer_to_npu
   ```

   同样，修改train.py文件，在第11行添加如上代码，如图4-126所示。

   图 4-126 修改 train.py 文件

2. 单卡训练。

   训练命令参考如下。

   ```
   cd /home/ma-user/open_clip
   python -m training.main \
       --save-frequency 1 \
       --zeroshot-frequency 1 \
       --report-to tensorboard \
       --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
       --train-num-samples 102400 \
       --dataset-type webdataset \
       --warmup 10000 \
       --batch-size=256 \
       --lr=1e-3 \
       --wd=0.1 \
       --epochs=30 \
       --workers=8 \
       --model ViT-B-32
   ```

   参数说明：

   - save-frequency：指定运行多少个epoch就保存模型参数，可以调大。
   - report-to tensorboard：指定输出loss指标到tensorboard，一般需要做精度评估才需要带上。
   - train-num-samples：指定每个epoch需要训练的样本个数，不超过总样本个数。
   - batch-size：指定一次处理的数据batch。
   - epochs：指定训练的epoch个数。

   训练结束后，模型输出目录为：

   /home/ma-user/open_clip/logs/xxx-model_ViT-B-32-lr_0.001-b_32-j_8-p_amp/checkpoints

3. 多卡训练

   训练命令参考如下。

```
cd /home/ma-user/open_clip/src
torchrun --nproc_per_node 4 -m training.main \
    --save-frequency 1 \
    --zeroshot-frequency 1 \
    --report-to tensorboard \
    --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
    --train-num-samples 102400 \
    --dataset-type webdataset \
    --warmup 10000 \
    --batch-size=256 \
    --lr=1e-3 \
    --wd=0.1 \
    --epochs=30 \
    --workers=8 \
    --model ViT-B-32
```

Step7 推理验证

首先将上面训练的最终模型文件epoch_29.pt 复制到/home/ma-user/open_clip目录下，然后在/home/ma-user/open_clip下，执行如下命令。

```
vi inference.py
```

将下面的代码复制进去后保存。

```
import os
import torch
from PIL import Image
import open_clip

if 'DEVICE_ID' in os.environ:
   print("DEVICE_ID:", os.environ['DEVICE_ID'])
else:
   os.environ['DEVICE_ID'] = "0"

model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', pretrained='/home/ma-user/
open_clip/epoch_29.pt')
model = model.to("npu")
tokenizer = open_clip.get_tokenizer('ViT-B-32')

image = preprocess(Image.open("./docs/CLIP.png")).unsqueeze(0)
text = tokenizer(["a diagram", "a dog", "a cat"])

print("input image shape:", image.shape)
print("input text shape:", text.shape)

with torch.no_grad(), torch.cuda.amp.autocast():
    image = image.to("npu")
    text = text.to("npu")
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)

    print("output image shape:", image_features.shape)
    print("output text shape:", text_features.shape)

    image_features /= image_features.norm(dim=-1, keepdim=True)
    text_features /= text_features.norm(dim=-1, keepdim=True)

    text_probs = (100.0 * image_features @ text_features.T).softmax(dim=-1)

print("Label probs:", text_probs)  # prints: [[1., 0., 0.]]
```

运行推理脚本。

```
python inference.py
```

由于./docs/CLIP.png图片是一张图表，因此结果值和第一个文本"a diagram"吻合，结果值会接近[[1., 0., 0.]]。

Step8 精度评估

1. 关闭数据集shuffle，保证训练数据一致。

   修改/home/ma-user/open_clip/src/training/data.py文件，搜索get_wds_dataset函数，将两处shuffle关闭，修改代码如下。

```
if is_train:
        if not resampled:
            print("dataset unshuffled.")
            #pipeline.extend([
            #    detshuffle2(
            #        bufsize=_SHARD_SHUFFLE_SIZE,
            #        initial=_SHARD_SHUFFLE_INITIAL,
            #        seed=args.seed,
            #        epoch=shared_epoch,
            #    ),
            #    wds.split_by_node,
            #    wds.split_by_worker,
            #])
        print("wds unshuffled.")
        pipeline.extend([
            # at this point, we have an iterator over the shards assigned to each worker at each node
            tarfile_to_samples_nothrow,  # wds.tarfile_to_samples(handler=log_and_continue),
        #    wds.shuffle(
        #        bufsize=_SAMPLE_SHUFFLE_SIZE,
        #        initial=_SAMPLE_SHUFFLE_INITIAL,
        #    ),
        ])
```

2. 重新训练1个epoch。脚本参考内容如下。

```
cd /home/ma-user/open_clip
python -m training.main \
    --save-frequency 1 \
    --zeroshot-frequency 1 \
    --report-to tensorboard \
    --train-data '/home/ma-user/open_clip/mscoco/{00000..00059}.tar' \
    --train-num-samples 102400 \
    --dataset-type webdataset \
    --warmup 10000 \
    --batch-size=256 \
    --lr=1e-3 \
    --wd=0.1 \
    --epochs=1 \
    --workers=8 \
    --model ViT-B-32
```

   训练完成后，tensorboard统计的记录会保存在/home/ma-user/open_clip/logs/xxx-model_ViT-B-32-lr_0.001-b_32-j_8-p_amp/tensorboard目录下。

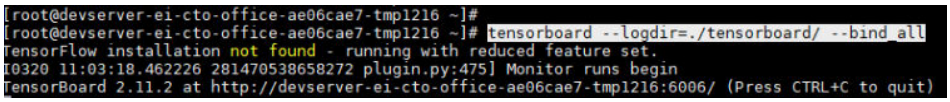
3. 通过docker cp命令将容器内tensorboard子目录复制到宿主机 /home下。

4. 在宿主机上安装tensorboard并启动。

```
pip install tensorboard  #安装
tensorboard --logdir=/home/tensorboard --bind_all  #启动
```

   启动成功后如下图所示。

   图 4-127 启动 tensorboard



5. 在浏览器访问http://{宿主机ip}:6006/。将train/loss导出为json，和GPU训练下导出的文件比较。

4.25 moondream2 基于 DevServer 适配 PyTorch NPU 推理指导

方案概览

本文档从模型部署的环境配置、模型转换、模型推理等方面进行介绍moondream2模型在ModelArts DevServer上部署，支持NPU推理场景。

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

资源规格要求

推理部署推荐使用DevServer资源和Ascend Snt9B单机单卡。

表 4-45 环境要求

名称	版本
CANN	cann_8.0.rc1
PyTorch	pytorch_2.1.0

获取镜像

表 4-46 获取镜像

分类	名称	获取路径
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc1-py_3.9-hce_2.0.2312-aarch64-snt9b-20240516142953-ca51f42	从SWR拉取。

Step1 准备环境

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

   说明

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

2. 检查环境。

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

   b. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表4-46**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -itd \
      --device=/dev/davinci1 \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
      -v /usr/local/dcmi:/usr/local/dcmi \
      -v /etc/ascend_install.info:/etc/ascend_install.info \
      -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
      -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
      --shm-size 32g \
      --net=bridge \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      ${image_name} bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。

- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。
```
docker exec -it ${container_name} bash
```

Step4 下载原始模型包

从HuggingFace官网下载moondream2模型包到本地，下载地址：**https://huggingface.co/vikhyatk/moondream2/tree/2024-03-06**。

在宿主机上创建一个空目录/home/temp，将下载的模型包存放在宿主机/home/temp/moondream2目录下，修改目录权限后，复制到容器中。

```
mkdir /home/temp       #创建一个空目录，将下载的模型包存放在宿主机/home/temp/moondream2目录下
chmod -R 777 moondream2     #修改moondream2目录权限
docker cp moondream2 moondream2:/home/ma-user/     #复制moondream2目录到容器中
```

Step5 准备测试数据

需要用户自己准备测试图片。

将测试图片存放在宿主机/home/temp/data目录下，修改目录权限后，复制到容器中。

```
chmod -R 777 data    #修改data目录权限
docker cp data moondream2:/home/ma-user/   #复制data目录到容器中
```

Step6 安装依赖

执行如下命令安装推理依赖。
```
pip install transformers timm einops torch==2.1.0 &&
pip install --upgrade sympy
```

Step7 启动推理

在容器/home/ma-user下运行启动推理脚本infer.py，NPU推理脚本内容参见**附录1：在NPU上运行infer.py脚本内容**。

```
python infer.py
```

运行结束后，会打印所有图片预测的平均时延。

NPU上运行后，结果会保存在/home/ma-user/result.txt下。

如果在GPU上运行，推荐直接在GPU宿主机上执行，因此不需要启动容器，直接将模型和数据复制到相应目录，然后安装PIP依赖后就可以运行。GPU推理脚本内容参见**附录2：在GPU上运行infer.py脚本内容**。

附录 1：在 NPU 上运行 infer.py 脚本内容

NPU上运行推理的infer.py脚本内容如下：

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from PIL import Image
import torch
import os
import time

import torch_npu
#from torch_npu.contrib import transfer_to_npu

import torchair as tng
from torchair.configs.compiler_config import CompilerConfig
```

```
#import logging
#from torchair.core.utils import logger
# 是否开启DEBUG日志
# logger.setLevel(logging.DEBUG)

model_id = "./moondream2"
revision = "2024-03-13"
model = AutoModelForCausalLM.from_pretrained(
    model_id, trust_remote_code=True, revision=revision
)

device = 'npu:0'
model = model.to(device)
tokenizer = AutoTokenizer.from_pretrained(model_id, revision=revision)


config = CompilerConfig()
npu_backend = tng.get_npu_backend(compiler_config=config)
model.text_model.transformer = torch.compile(model.text_model.transformer, backend=npu_backend,
dynamic=True, fullgraph=True)

filenames = os.listdir(r'./data')
filenames = sorted(filenames)
count = 0
total_time = 0.0
not_num = 1
with open("./result.txt", 'w+') as f:
  for file in filenames:
    t1 = time.time()
    image = Image.open('./data/'+file)
    enc_image = model.encode_image(image)
    enc_image = enc_image.to(device)
    result = model.answer_question(enc_image, "Describe in detail what is in the video frame. The rule is:
first describe the main body of the character in the video frame, including action, state, characteristics, etc.,
do not make associations or summarize. Then describe the environment, such as the background; then
describe how the video was shot, such as close-ups. Do not appear 'seems', 'may' and other words, need to
be sure of the description, do not need to be ambiguous description.", tokenizer)
    cost = time.time()-t1
    if not_num <=0:
      count = count+1
      total_time += cost
      print("infer time:"+str(cost))
      print("average infer time:"+str(total_time/count), " total count:"+str(count))
    else:
      not_num = not_num -1
    f.write(file + ":" + "\n")
    f.write(result + "\n\n")
```

附录 2：在 GPU 上运行 infer.py 脚本内容

GPU上运行推理的infer.py脚本内容如下：

```
from transformers import AutoModelForCausalLM, AutoTokenizer
from PIL import Image
import torch
import os
import time

model_id = "./moondream2"
revision = "2024-03-13"
model = AutoModelForCausalLM.from_pretrained(
    model_id, trust_remote_code=True, revision=revision
)

device = 'cuda:0'
model = model.to(device)
tokenizer = AutoTokenizer.from_pretrained(model_id, revision=revision)

filenames = os.listdir(r'./data')
```

```
filenames = sorted(filenames)
count = 0
total_time = 0.0
not_num = 1
with open("./result.txt", 'w+') as f:
  for file in filenames:
    t1 = time.time()
    image = Image.open('./data/'+file)
    enc_image = model.encode_image(image)
    enc_image = enc_image.to(device)
    result = model.answer_question(enc_image, "Describe in detail what is in the video frame. The rule is: first describe the main body of the character in the video frame, including action, state, characteristics, etc., do not make associations or summarize. Then describe the environment, such as the background; then describe how the video was shot, such as close-ups. Do not appear 'seems', 'may' and other words, need to be sure of the description, do not need to be ambiguous description.", tokenizer)
    cost = time.time()-t1
    if not_num <=0:
      count = count+1
      total_time += cost
      print("infer time:"+str(cost))
      print("average infer time:"+str(total_time/count), " total count:"+str(count))
    else:
      not_num = not_num -1
    f.write(file + ":" + "\n")
    f.write(result + "\n\n")
```

4.26 AIGC 工具 tailor 使用指导

tailor 简介

tailor是AIGC场景下用于模型转换（onnx到mindir）和性能分析的辅助工具，当前支持以下功能。

表 4-47 功能总览

功能大类	具体功能
模型转换	• 固定shape转模型 • 动态shape传入指定档位转模型 • 支持fp32 • 支持AOE优化
benchmark	• 支持测试性能 • 支持精度测试
profiling	支持分析算子的profiling

环境准备

本工具支持x86和ARM的系统环境，使用前需要安装以下软件。

表 4-48 安装软件及步骤

软件	安装步骤
mindspore-lite	安装版本：2.2.10
	下载地址：https://www.mindspore.cn/lite/docs/zh-CN/r2.2/use/downloads.html 需要下载的安装包与机器规格有关，请根据需要选择合适的安装包。 • 如果机器规格为Snt9B，则下载操作系统为Linux-aarch64的tag包：**mindspore-lite-2.2.10-linux-aarch64.tar.gz**。 • 如果机器规格为Snt3P，则下载操作系统为Linux-x86_64的tag包：**mindspore-lite-2.2.10-linux-x64.tar.gz**。

	安装方式如下：
	MindSpore Lite云侧推理包解压缩后，设置`LITE_HOME`环境变量为解压缩的路径，例如：
	`export LITE_HOME=$some_path/mindspore-lite-2.2.10-linux-aarch64`
	设置环境变量LD_LIBRARY_PATH：
	`export LD_LIBRARY_PATH=$LITE_HOME/runtime/lib:$LITE_HOME/runtime/third_party/dnnl:$LITE_HOME/tools/converter/lib:$LD_LIBRARY_PATH`
	如果需要使用convert_lite或者benchmark工具，则需要设置环境变量PATH。
	`export PATH=$LITE_HOME/tools/converter/converter:$LITE_HOME/tools/benchmark:$PATH`

cann	安装版本：CANN 7.0.0
	下载地址：https://support.huawei.com/enterprise/zh/ascend-computing/cann-pid-251168373/software/258923273?idAbsPath=fixnode01%7C23710424%7C251366513%7C22892968%7C251168373 请下载toolkit和对应机器的kernels包，以Snt9B为例则下载“Ascend-cann-toolkit_7.0.0_linux-aarch64.run”和“Ascend-cann-kernels-型号_7.0.0_linux.run”。

	安装命令（以Snt9B的cann安装为例）：
	`./Ascend-cann-toolkit_7.0.0_linux-aarch64.run --full`
	`./Ascend-cann-kernels-型号_7.0.0_linux.run --install`
	请安装在默认路径下：/usr/local/Ascend，暂不支持安装在自定义路径下。

tailor	安装版本：0.3.4
	下载地址： https://cneast3-modelarts-sdk.obs.cn-east-3.myhuaweicloud.com/tailor-0.3.4-py3-none-any.whl SHA-256： https://cneast3-modelarts-sdk.obs.cn-east-3.myhuaweicloud.com/tailor-0.3.4-py3-none-any.whl/1713929258832/tailor-0.3.4-py3-none-any.whl.sha256
	安装命令： `pip install tailor-0.3.4-py3-none-any.whl`

使用指导

tailor支持“命令行”和“Python API”两种方式使用。

- **命令行方式**

  命令行运行样例：

  ```
  tailor --model_path="./resnet50-v2-7.onnx"--config_path="./config.ini"--
  input_shape="data:1,3,224,224"--output_path="/home/"--accuracy="fp32"--aoe=True
  ```

  config.ini参考内容如下：

  ```
  [ascend_context]
  input_shape=data:[-1,3,224,224]
  dynamic_dims=[1],[2],[3]
  ```

  表 4-49 参数说明

参数名称	功能描述	参数类型	是否必填	默认值	备注
--model_path	指定onnx模型路径。	string	是	-	-
--config_path	指定模型配置文件路径。	string	否	-	tailor支持动态分档转换功能，需要指定配置文件路径，需要注意即便有配置文件，只要是动态模型就需要指定--input_shape参数。
--input_shape	指定模型转换的shape。	string	否	-	固定shape模型转换可以不填，动态模型转换必填。
--output_path	指定结果输出路径。	string	否	默认为当前目录下。	-

参数名称	功能描述	参数类型	是否必填	默认值	备注
--aoe	是否在转换时进行AOE优化。	bool	否	False	AOE优化可以提升模型性能，但不是一定有提升，需要注意开启AOE，会导致模型转换耗时极大延长。
--accuracy	指定模型精度，只支持fp16和fp32。	string	否	fp16	-

- **Python API**
  - 导入包并创建tailor对象。
    ```
    from tailor.tailor import Tailor
    onnx_model_path = "./resnet50-v2-7.onnx"    # 相对路径或者绝对路径均可以
    t = Tailor(onnx_model_path)
    ```
  - 查询onnx模型的输入信息。
    ```
    # 查询onnx模型的输入信息
    t.get_model_input_info()
    ```

    图 4-128 查询 onnx 模型的输入输出信息

    ```
    In [3]: t.get_model_input_info()
    Out[3]:
    [{'name': 'data',
      'type': 'tensor(float)',
      'shape': ['N', 3, 224, 224],
      'np_type': numpy.float32}]
    ```
  - 查询onnx模型的输出信息。
    ```
    # 查询模型的输出信息
    t.get_model_output_info()
    ```

    图 4-129 查询 onnx 模型的输出信息

    ```
    In [4]: t.get_model_output_info()
    Out[4]:
    [{'name': 'resnetv24_dense0_fwd',
      'type': 'tensor(float)',
      'shape': ['N', 1000]}]
    ```
  - 固定shape模型，可以直接运行。

```
t.run()
```

- 指定档位信息运行。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape)
```

- 动态档位执行config_path运行。需要注意，只要是动态模型，就必须要传入input_shape，因为转换模型后的benchmark和profiling都依赖单个shape操作。

```
input_shape="data:1,3,224,224"
config_path = "./resnet/config.ini"
t.run(input_shape=input_shape, config_path=config_path)
```

- 指定精度为fp32。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape, accuracy='fp32')
```

- 开启AOE优化。

```
input_shape="data:1,3,224,224"
t.run(input_shape=input_shape, aoe=True)
```

- 指定输出位置。

```
input_shape="data:1,3,224,224"
# 不指定输出路径，则默认在当前执行目录存储结果
t.run(input_shape=input_shape, output_path="/home/xxx")
```

运行结果将存储在output文件夹中，如果用户指定了output_path，会指定位置保存，如果不指定则在当前代码执行目录生成文件夹保存输出。整体运行的结果都存放在output文件夹中，每转一次模型就会根据模型名称以及相关参数生成结果文件，如下图所示。

图 4-130 output 文件

```
ll output/
total 8
drwx------ 6 root 4096 Nov  6 15:21 resnet50-v2-7_fp16_20231106152024/
drwx------ 6 root 4096 Nov 15 10:06 resnet50-v2-7_fp16_20231115100511/
drwx------ 6 root 4096 Nov  6 23:57 resnet50-v2-7_fp16_aoe_20231106194012/
drwx------ 6 root 4096 Nov  6 15:33 unet_fp16_20231106152314/
drwx------ 6 root 4096 Nov 10 22:36 unet_fp16_aoe_20231107061337/
drwx------ 6 root 4096 Nov  6 15:23 yolox_m_mmyolo_fp16_20231106152141/
drwx------ 6 root 4096 Nov  7 06:13 yolox_m_mmyolo_fp16_aoe_20231106235726/
```

在每次运行的结果文件中，分为三部分：convert、benchmark、profiling，相关的文件及存储内容如下。

表 4-50 输出文件介绍（以模型名称为 resnet50-v2-7.onnx 为例）

类别	文件名称	是否一定生成	文件存储内容
convert	resnet50-v2-7.mindir	是	转换后的mindir模型。
	resnet50-v2-7.om	否	转换过程中的om文件，不是必定生成。
	onnx_to_mindspore.sh	是	模型转换命令，可以本地直接运行。

类别	文件名称	是否一定生成	文件存储内容
	resnet50-v2-7_convert.log	是	模型转换过程的日志。
	config.ini	否	配置文件，在指定fp32精度或者AOE打开时会生成。
	onnx_to_mindspore_aoe.sh	否	在打开AOE功能时会生成。
benchmark	run_benchmark.sh	是	运行benchmark的脚本，可本地直接运行。
	run_benchmark_accuracy.sh	是	benchmark运行精度的脚本，可本地直接运行。
	performance.txt	是	benchmark性能测试结果。
	accuracy.txt	是	精度测试结果。
	*.bin	是	自动构造的输入随机bin文件，可能存在多个。
	resnet50-v2-7_output.txt	是	上述bin文件作为输入时onnx模型运行的结果。
profiling	run_profiling.sh	是	运行profiling的脚本，可本地直接运行。
	profiling.config	是	运行profiling的配置文件。
	profiling.json	是	运行profiling的配置文件。
	PROF_xxx开头的文件夹	是	运行profiling的结果文件夹。
	run_aggregate.sh	是	运行数据聚合的脚本，可直接本地运行。
	run_profiling.log	是	存储运行profiling的日志信息。

5 数字人模型训练推理

5.1 Wav2Lip 推理基于 DevServer 适配 PyTorch NPU 推理指导（6.3.907）

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署Wav2Lip模型用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 5-1 环境要求

名称	版本
PyTorch	pytorch_2.1.0
驱动	23.0.6

获取软件和镜像

表 5-2 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907-xxx.zip软件包中的AscendCloud-AIGC-6.3.907-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表5-2**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

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

   说明

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

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-2**。

```
docker pull {image_url}
```

Step3 获取代码并上传

上传推理代码AscendCloud-AIGC-6.3.907-xxx.zip到宿主机的工作目录中，包获取路径请参见**表5-2**。

Step4 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
docker run -itd  --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step5 下载并适配代码

1. 在容器中解压代码包。

```
unzip AscendCloud-AIGC-6.3.907-*.zip
rm -rf AscendCloud-AIGC-6.3.907-*
```

2. 执行wav2lip推理插件的安装脚本。

```
cd multimodal_algorithm/Wav2Lip/inference/f361e9527b917a435928a10931fee9ac7be109cd
source install.sh
```


3. 从官网下载**Wav2lip权重文件**和**Wav2Lip+GAN权重文件**，并放在容器的checkpoints目录下。上一步执行完source install.sh命令后，会自动生成checkpoints目录。
4. 从官网下载模型**s3fd-619a316812.pth**，并重命名为s3fd.pth，放在容器路径face_detection/detection/sfd下。上一步执行完source install.sh命令后，会自动生成face_detection/detection/sfd目录。

Step6 服务调用

1. 提前准备人物图片，支持'jpg', 'png', 'jpeg'格式。推荐测试图片大小1280*720或1920*1080。
2. 提前准备音频文件audio，支持'wav', 'mp3', 'mp4'格式。
3. 在代码根目录Wav2lip下创建test_wav2lip.sh, 复制以下内容粘贴至test_wav2lip.sh中，参数参照下方说明进行配置。

```
#!/bin/bash
start_time=$(date +%s)
python inference.py --checkpoint_path <ckpt_path> --face <jpg_path> --audio <audio_path> --outfile <output_path>
end_time=$(date +%s)
execution_time=$((end_time - start_time))
echo "wav2lip cost: $execution_time s"
```

<ckpt_path>：模型权重路径 checkpoints/wav2lip.pth或 checkpoints/wav2lip_gan.pth。

<jpg_path>: 人物图片路径，需要指定到具体的文件，例如 xxx/xxx.jpg。

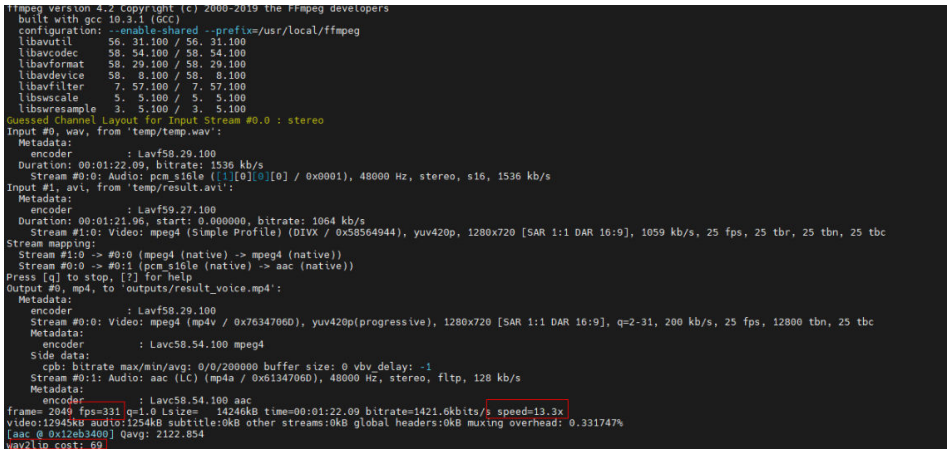
<audio_path>：音频路径，需要指定到具体的文件，例如 xxx/xxx.mp4。

<output_path>：视频结果输出路径，需要指定到具体的输出文件名，例如 xxx/xxx.mp4。

4. 执行test_wav2lip.sh脚本进行推理。

```
cd Wav2Lip
bash test_wav2lip.sh
```

图 5-1 输出日志截图



5.2 Wav2Lip 训练基于 DevServer 适配 PyTorch NPU 训练指导（6.3.907）

本文档主要介绍如何在ModelArts Lite的DevServer环境中，使用NPU卡训练Wav2Lip模型。本文档中提供的Wav2Lip模型，是在原生Wav2Lip代码基础上适配后的模型，可以用于NPU芯片训练。

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

Wav2Lip模型的输入为任意的一段视频和一段语音，输出为一段唇音同步的视频。

Wav2Lip的网络模型总体上分成三块：生成器、判别器和一个预训练好的唇音同步判别模型Pre-trained Lip-sync Expert。

- 生成器是基于encoder-decoder的网络结构，分别利用2个encoder（speech encoder和identity encoder）去对输入的语音和视频人脸进行编码，并将二者的编码结果进行拼接，送入到face decoder中进行解码得到输出的视频帧。
- 判别器Visual Quality Discriminator对生成结果的质量进行规范，提高生成视频的清晰度。
- 引入预训练的唇音同步判别模型Pre-trained Lip-sync Expert，作为衡量生成结果的唇音同步性的额外损失，可以更好的保证生成结果的唇音同步性。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源开展Wav2Lip训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 5-3 环境要求

名称	版本
driver	23.0.6
PyTorch	pytorch_2.1.0

获取软件和镜像

表 5-4 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.907-xxx.zip软件包中的AscendCloud-AIGC-6.3.907-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果上述软件获取路径打开后未显示相应的软件信息，说明您没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一： swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240727152329-0f2c29a	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.907版本，请参考**表5-4**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

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

   说明

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

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-4**。

```
docker pull {image_url}
```

Step3 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。

```
export work_dir="自定义挂载的工作目录"
export container_work_dir="自定义挂载到容器内的工作目录"
export container_name="自定义容器名称"
export image_name="镜像名称或ID"
// 启动一个容器去运行镜像
docker run  -itd --net=bridge \
   -p 8080:8080 \
   --device=/dev/davinci0 \
   --device=/dev/davinci_manager \
   --device=/dev/devmm_svm \
   --device=/dev/hisi_hdc \
   --shm-size=32g \
   -v /usr/local/dcmi:/usr/local/dcmi \
   -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
   -v /var/log/npu/:/usr/slog \
   -v /usr/local/sbin/npu-smi:/usr/local/sbin/npu-smi \
   -v ${work_dir}:${container_work_dir} \
   --name ${container_name} \
   ${image_name} \
   /bin/bash
```

参数说明：

- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明

  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -p 8080:8080：开启一个端口，可以web访问（如冲突，可自行更换其他端口）。
- ${image_name}：容器镜像的名称。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。

```
docker exec -it ${container_name} bash
```

Step4 安装依赖和软件包

1. 从github拉取Wav2Lip代码。

```
cd /home/ma-user
git clone https://github.com/Rudrabha/Wav2Lip.git
cd /home/ma-user/Wav2Lip
git reset --hard f361e9527b917a435928a10
```


如果出现报错SSL certificate problem: self signed certificate in certificate chain

图 5-2 报错 SSL certificate problem

可采取忽略SSL证书验证：使用以下命令来克隆仓库，它将忽略SSL证书验证。

```
git clone -c http.sslVerify=false https://github.com/Rudrabha/Wav2Lip.git
```

2. 安装Wav2Lip Ascend软件包。
   a. 将获取到的Wav2Lip Ascend软件包AscendCloud-AIGC-*.zip文件上传到容器的/home/ma-user目录下。获取路径：**Support网站**。
   b. 解压AscendCloud-AIGC-*.zip文件，解压后将里面指定文件与对应Wave2Lip文件进行替换。

```
cd  /home/ma-user
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
cp AscendCloud/multimodal_algorithm/Wav2Lip/train/f361e9527b917a435928a10/* /home/ma-user/Wav2Lip/
rm -rf AscendCloud*
```

说明

AscendCloud-AIGC-*.zip后面的*表示时间戳，请按照实际替换。

要替换的文件目录结构如下所示：

```
|---Wav2Lip_code/
      --- requirements.txt          #建议的依赖包版本

注：需要对以下文件进行修改
 --- color_syncnet_train.py     #训练expert discriminator唇形同步鉴别器
 --- wav2lip_train.py           #训练 Wav2Lip 模型
 --- preprocess.py              #对初始视频数据进行推理
       在以上三个文件内import末尾增加import如下：
          import torch_npu
          from torch_npu.contrib import transfer_to_npu
```

3. 安装Python依赖包，文件为requirements.txt文件。

```
pip install -r requirements.txt
```

Step5 训练 Wav2Lip 模型

1. 准备预训练模型。下载需要使用的预训练模型。
   - 人脸检测预训练模型，**下载链接**。
   - 专家唇形同步鉴别器，**下载链接** ，此链接是官方提供的预训练模型。训练Wav2Lip模型时需要使用专家唇形同步鉴别器，用户可以用自己的数据训练，也可以直接使用官方提供的预训练模型。
2. 处理初始视频数据集。
   a. 将下载好的**人脸检测预训练模型**修改名字为s3fd.pth，上传到/home/ma-user/Wav2Lip/face_detection/detection/sfd/s3fd.pth目录。
   b. 下载**LRS2数据集**。数据集文件夹结构如下：

```
├── LRS2_partly
|   ├── main
|   |   ├── five-digit numbered video IDs ending with (.mp4)
|   |   ├── 00001.mp4
|   |   ├── 00002.mp4
```

   c. 对数据集进行预处理。具体命令如下。

```
python preprocess.py --data_root ./LRS2_partly --preprocessed_root lrs2_preprocessed/
```

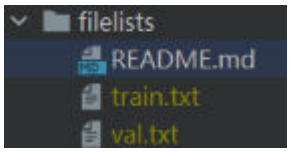
data_root参数为原始视频根目录，preprocessed_root参数为处理后生成的数据集目录。

处理后数据目录如下所示。

```
preprocessed_root (lrs2_preprocessed)
 ├── main
|    ├── Folders with five-digit numbered video IDs ( 00001 )
|    |    ├── *.jpg
|    |    ├── audio.wav
|    ├── 00001
|    |    ├── *.jpg
|    |    ├── audio.wav
```

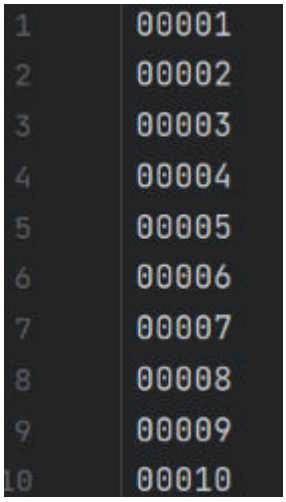
d. 将LRS2文件列表中的.txt文件（train、val）放入该filelists文件夹中。

图 5-3 filelists 文件夹



train.txt和val.txt内容参考如下，为处理后视频数据的目录名字。

图 5-4 train.txt 和 val.txt 内容



3. 训练专家唇形同步鉴别器。

   如果使用LRS2数据集，可选择跳过此步骤。如果使用自己的数据集，训练命令参考如下。

   ```
   python color_syncnet_train.py --data_root ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel/syncnet_model/ --checkpoint_path ./checkpoints/lipsync_expert.pth
   ```

   参数说明：

   - --data_root ： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
   - --checkpoint_dir ： 此目录用于保存模型。
   - -checkpoint_path ： （可选）可基于此目录的lipsync_expert模型继续进行训练，如果重新训练则不需要此参数。

   默认每10000 step保存一次模型。

4. 训练Wav2Lip模型。

   训练Wav2Lip模型时需要使用专家唇形同步鉴别器。可以使用上一步3中的训练结果，也可以直接下载官方提供的**预训练权重**来使用。

具体训练命令如下。

```
python wav2lip_train.py --data_root  ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel --
syncnet_checkpoint_path ./checkpoints/lipsync_expert.pth --checkpoint_path ./checkpoints/wav2lip.pth
```

首次训练会进行模型评估，默认为700 step，请耐心等待，结束之后会进行正式训练。

参数说明：

- --data_root ： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
- --checkpoint_dir ： 此目录用于保存模型。
- --syncnet_checkpoint_path ： 专家鉴别器的目录。
- --checkpoint_path ： （可选）可基于此目录的Wav2Lip模型继续进行训练，如果重新训练则不需要此参数。

默认每3000 step保存一次模型。

注：

- 专家鉴别器的评估损失应降至约 0.25，Wav2Lip评估同步损失应降至约 0.2，以获得良好的结果。
- 可以在文件设置其他不太常用的超参数hparams.py，常用超参如下：

```
nepochs  训练总步数
checkpoint_interval  Wav2Lip模型保存间隔步数
eval_interval        Wav2Lip模型评估间隔步数
syncnet_eval_interval          专家鉴别器模型评估间隔步数
syncnet_checkpoint_interval     专家鉴别器模型保存间隔步数
```

5.3 Wav2Lip 基于 DevServer 适配 PyTorch NPU 推理指导（6.3.906）

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源部署Wav2Lip模型用于推理的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

资源规格要求

推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 5-5 环境要求

名称	版本
PyTorch	pytorch_2.1.0

名称	版本
驱动	23.0.5

获取软件和镜像

表 5-6 获取软件和镜像

分类	名称	获取路径
插件代码包	AscendCloud-6.3.906-xxx.zip软件包中的AscendCloud-AIGC-6.3.906-xxx.zip **说明** 包名中的xxx表示具体的时间戳，以包名的实际时间为准。	获取路径：**Support-E** **说明** 如果没有下载权限，请联系您所在企业的华为方技术支持下载获取。
基础镜像	西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.0.rc2-py_3.9-hce_2.0.2312-aarch64-snt9b-20240606190017-b881580	从SWR拉取。

约束限制

- 本文档适配昇腾云ModelArts 6.3.904版本，请参考**表5-6**获取配套版本的软件包和镜像，请严格遵照版本配套关系使用本文档。
- 确保容器可以访问公网。

Step1 准备环境

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

   说明

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

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

   ```
   npu-smi info                    # 在每个实例节点上运行此命令可以看到NPU卡状态
   npu-smi info -l | grep Total    # 在每个实例节点上运行此命令可以看到总卡数
   ```

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

3. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

4. 配置IP转发，用于容器内的网络访问。执行以下命令查看net.ipv4.ip_forward配置项的值，如果为1，可跳过此步骤。
```
sysctl -p | grep net.ipv4.ip_forward
```
如果net.ipv4.ip_forward配置项的值不为1，执行以下命令配置IP转发。
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step2 获取基础镜像

建议使用官方提供的镜像部署服务。镜像地址{image_url}参见**表5-6**。

```
docker pull {image_url}
```

Step3 获取代码并上传

上传推理代码AscendCloud-AIGC-6.3.906-xxx.zip到宿主机的工作目录中，包获取路径请参见**表5-6**。

Step4 启动容器镜像

1. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。
```
docker run -itd  --net=host \
--device=/dev/davinci0 \
--device=/dev/davinci_manager \
--device=/dev/devmm_svm \
--device=/dev/hisi_hdc \
--shm-size=1024g \
-v /usr/local/dcmi:/usr/local/dcmi \
-v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
-v /var/log/npu/:/usr/slog \
-v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
-v ${work_dir}:${container_work_dir} \
--name ${container_name} \
${image_id}  \
/bin/bash
```
参数说明：
- -v ${work_dir}:${container_work_dir}：代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。

  说明
  - 容器不能挂载到/home/ma-user目录，此目录为ma-user用户家目录。如果容器挂载到/home/ma-user下，拉起容器时会与基础镜像冲突，导致基础镜像不可用。
  - driver及npu-smi需同时挂载至容器。
- --name ${container_name}：容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- ${image_id}：镜像ID，通过docker images查看刚拉取的镜像ID。

2. 通过容器名称进入容器中。默认使用ma-user用户，后续所有操作步骤都在ma-user用户下执行。
```
docker exec -it ${container_name} bash
```

Step5 下载并适配代码

1. 在容器中解压代码包。


```
unzip AscendCloud-AIGC-6.3.906-*.zip
rm -rf AscendCloud-AIGC-6.3.906-*
```

2. 执行wav2lip推理插件的安装脚本。
```
cd multimodal_algorithm/Wav2Lip/inference/f361e9527b917a435928a10931fee9ac7be109cd
source install.sh
```
3. 从官网下载**Wav2lip权重文件**和**Wav2Lip+GAN权重文件**，并放在容器的checkpoints目录下。上一步执行完source install.sh命令后，会自动生成checkpoints目录。
4. 从官网下载模型**s3fd-619a316812.pth**，并重命名为s3fd.pth，放在容器路径face_detection/detection/sfd下。上一步执行完source install.sh命令后，会自动生成face_detection/detection/sfd目录。

Step6 服务调用

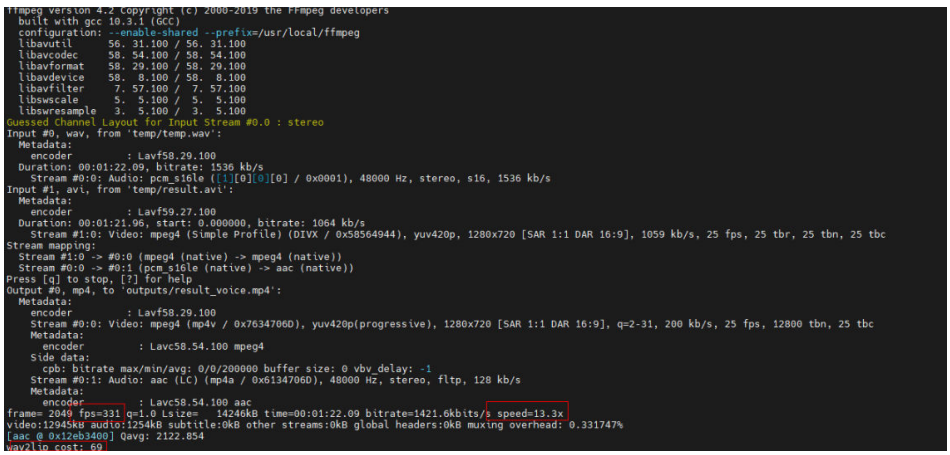
1. 提前准备人物图片，支持'jpg', 'png', 'jpeg'格式。推荐测试图片大小1280*720或1920*1080。
2. 提前准备音频文件audio，支持'wav', 'mp3', 'mp4'格式。
3. 在代码根目录Wav2lip下创建test_wav2lip.sh, 复制以下内容粘贴至test_wav2lip.sh中，参数参照下方说明进行配置。
```
#!/bin/bash
start_time=$(date +%s)
python inference.py --checkpoint_path <ckpt_path> --face <jpg_path> --audio <audio_path> --outfile <output_path>
end_time=$(date +%s)
execution_time=$((end_time - start_time))
echo "wav2lip cost: $execution_time s"
```
<ckpt_path>：模型权重路径 checkpoints/wav2lip.pth或 checkpoints/wav2lip_gan.pth。

<jpg_path>: 人物图片路径，需要指定到具体的文件，例如 xxx/xxx.jpg。

<audio_path>：音频路径，需要指定到具体的文件，例如 xxx/xxx.mp4。

<output_path>：视频结果输出路径，需要指定到具体的输出文件名，例如 xxx/xxx.mp4。
4. 执行test_wav2lip.sh脚本进行推理。
```
cd Wav2Lip
bash test_wav2lip.sh
```

图 5-5 输出日志截图



5.4 Wav2Lip 基于 DevServer 适配 PyTorch NPU 训练指导（6.3.902）

本文档主要介绍如何在ModelArts Lite的DevServer环境中，使用NPU卡训练Wav2Lip模型。本文档中提供的Wav2Lip模型，是在原生Wav2Lip代码基础上适配后的模型，可以用于NPU芯片训练。

Wav2Lip是一种基于对抗生成网络的由语音驱动的人脸说话视频生成模型。主要应用于数字人场景。不仅可以基于静态图像来输出与目标语音匹配的唇形同步视频，还可以直接将动态的视频进行唇形转换，输出与输入语音匹配的视频，俗称“对口型”。该技术的主要作用就是在将音频与图片、音频与视频进行合成时，口型能够自然。

Wav2Lip模型的输入为任意的一段视频和一段语音，输出为一段唇音同步的视频。

Wav2Lip的网络模型总体上分成三块：生成器、判别器和一个预训练好的唇音同步判别模型Pre-trained Lip-sync Expert。

- 生成器是基于encoder-decoder的网络结构，分别利用2个encoder（speech encoder和identity encoder）去对输入的语音和视频人脸进行编码，并将二者的编码结果进行拼接，送入到face decoder中进行解码得到输出的视频帧。
- 判别器Visual Quality Discriminator对生成结果的质量进行规范，提高生成视频的清晰度。
- 引入预训练的唇音同步判别模型Pre-trained Lip-sync Expert，作为衡量生成结果的唇音同步性的额外损失，可以更好的保证生成结果的唇音同步性。

方案概览

本方案介绍了在ModelArts的DevServer上使用昇腾计算资源开展Wav2Lip训练的详细过程。完成本方案的部署，需要先联系您所在企业的华为方技术支持购买DevServer资源。

本方案目前仅适用于企业客户。

环境配置要求

准备一台ModelArts的DevServer物理机环境，推荐使用“西南-贵阳一”Region上的DevServer资源和Ascend Snt9B单机单卡。

表 5-7 环境要求

模型	版本
CANN	7.0.1
PyTorch	2.1
Python	3.10

获取软件

获取Wav2Lip Ascend适配代码ascendcloud-aigc-6.3.902-*.tar.gz文件。获取路径：Support网站。

说明

如果没有软件下载权限，请联系您所在企业的华为方技术支持下载获取。

ascendcloud-aigc-6.3.902-*.tar.gz文件名中的*表示具体的时间戳，以包名的实际时间为准。

Step1 准备环境

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

   说明

   购买DevServer资源时如果无可选资源规格，需要联系华为云技术支持申请开通。

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

2. 检查环境。

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

   ```
   npu-smi info
   ```

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

   b. 检查docker是否安装。

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

   如尚未安装，运行以下命令安装docker。

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

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

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

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

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

3. 获取基础镜像。建议使用官方提供的镜像部署推理服务。

   镜像地址{image_url}为：

   西南-贵阳一：swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-snt9b-20240312154948-219655b

   ```
   docker pull ${image_url}
   ```

4. 启动容器镜像。启动前请先按照参数说明修改${}中的参数。可以根据实际需要增加修改参数。

   ```
   export work_dir="自定义挂载的工作目录"
   export container_work_dir="自定义挂载到容器内的工作目录"
   export container_name="自定义容器名称"
   export image_name="swr.cn-southwest-2.myhuaweicloud.com/atelier/
   pytorch_2_1_ascend:pytorch_2.1.0-cann_7.0.0-py_3.9-hce_2.0.2312-aarch64-
   snt9b-20240312154948-219655b"
   // 启动一个容器去运行镜像
   docker run -itd \
           --device=/dev/davinci0 \
           --device=/dev/davinci_manager \
   ```

```
    --device=/dev/devmm_svm \
    --device=/dev/hisi_hdc \
    -v /usr/local/bin/npu-smi:/usr/local/bin/npu-smi \
    -v /usr/local/dcmi:/usr/local/dcmi \
    -v /etc/ascend_install.info:/etc/ascend_install.info \
    -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
    -v /usr/local/Ascend/driver:/usr/local/Ascend/driver \
    --shm-size 32g \
    --net=bridge \
    -v ${work_dir}:${container_work_dir} \
    --name ${container_name} \
    ${image_name} bash
```

参数说明：

- --name ${container_name} 容器名称，进入容器时会用到，此处可以自己定义一个容器名称。
- -v ${work_dir}:${container_work_dir} 代表需要在容器中挂载宿主机的目录。宿主机和容器使用不同的文件系统。work_dir为宿主机中工作目录，目录下存放着训练所需代码、数据等文件。container_work_dir为要挂载到的容器中的目录。为方便两个地址可以相同。
- ${image_name} 代表 ${image_name}。

5. 通过容器名称进入容器中。

```
docker exec -it ${container_name} bash
```

Step2 安装依赖和软件包

1. Python版本要求3.10，如果不满足的话，建议更新容器的conda环境的Python版本。

```
# 输入如下命令，待conda界面准备完成后输入y，等待自动下载安装
conda create --name py310 python=3.10
```

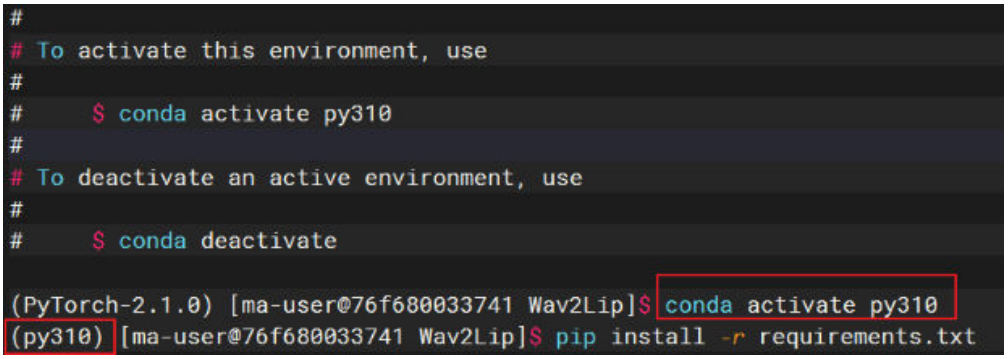
参数说明：

- --name：该参数为新环境名字，可以自定义一个，此处以py310举例。
- python=新环境Python版本

```
# 完成后输入如下命令激活新环境
conda activate py310
```

激活新conda环境后控制台显示（ py310 ）即为切换成功，如下图所示。

图 5-6 激活新 conda 环境



2. 从github拉取Wav2Lip代码。

```
cd /home/ma-user
git clone https://github.com/Rudrabha/Wav2Lip.git
```

如果出现报错SSL certificate problem: self signed certificate in certificate chain

图 5-7 报错 SSL certificate problem

可采取忽略SSL证书验证：使用以下命令来克隆仓库，它将忽略SSL证书验证。

```
git clone -c http.sslVerify=false https://github.com/Rudrabha/Wav2Lip.git
```

3. 安装Wav2Lip Ascend软件包。
   a. 将获取到的Wav2Lip Ascend软件包ascendcloud-aigc-*.tar.gz文件上传到容器的/home/ma-user/Wav2Lip目录下。获取路径：**Support网站**。
   b. 解压ascendcloud-aigc-*.tar.gz文件，解压后将里面文件与对应Wave2Lip文件进行替换。

```
cd  /home/ma-user/Wav2Lip
tar -zxvf ascendcloud-aigc-6.3.902-*.tar.gz
tar -zxvf ascendcloud-aigc-poc-Wav2Lip_Ascend.tar.gz
mv Wav2Lip_code/* ./
rm -rf ascendcloud-aigc-* Wav2Lip_code/
```

   说明

   ascendcloud-aigc-6.3.902-*.tar.gz后面的*表示时间戳，请按照实际替换。

   要替换的文件目录结构如下所示：

```
|---Wav2Lip_code/
      --- color_syncnet_train.py     #训练expert discriminator唇形同步鉴别器
      --- inference.py               #推理代码，可以与任意音频或视频进行口型同步
      --- preprocess.py              #对初始视频数据进行推理
      --- read.txt                   #关于包版本兼容问题的一些处理方案
      --- requirements.txt           #建议的依赖包版本
      --- wav2lip_train.py           #训练 Wav2Lip 模型
```

4. 安装Python依赖包，文件为requirements.txt文件。

```
pip install -r requirements.txt
```

   由于librosa、numba、llvmlite包的版本兼容问题，会出现报错ModuleNotFoundError: No module named 'numba.decorators'。

   此时进入Python包librosa安装位置，打开文件site-packages/librosa/util/decorators.py，修改文件如下：

```
import warnings
from decorator import decorator import six
#注释此行
#from numba.decorators import jit as optional_jit
#修改此行如下
#__all__ = ['moved', 'deprecated', 'optional_jit']
__all__ = ['moved', 'deprecated']
```

Step3 训练 Wav2Lip 模型

1. 准备预训练模型。下载需要使用的预训练模型。
   - 人脸检测预训练模型，**下载链接**。
   - 专家唇形同步鉴别器，**下载链接** ，此链接是官方提供的预训练模型。训练Wav2Lip模型时需要使用专家唇形同步鉴别器，用户可以用自己的数据训练，也可以直接使用官方提供的预训练模型。
2. 处理初始视频数据集。
   a. 将下载好的**人脸检测预训练模型**上传到/home/ma-user/Wav2Lip/face_detection/detection/sfd/s3fd.pth目录。
   b. 下载**LRS2数据集**。数据集文件夹结构如下：

```
├── LRS2_partly
|    ├── main
```

```
|   |   ├── five-digit numbered video IDs ending with (.mp4)
|   |   ├── 00001.mp4
|   |   ├── 00002.mp4
```

c. 对数据集进行预处理。具体命令如下。

```
python preprocess.py --data_root ./LRS2_partly --preprocessed_root lrs2_preprocessed/
```

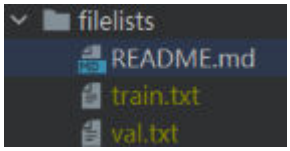
data_root参数为原始视频根目录，preprocessed_root参数为处理后生成的数据集目录。

处理后数据目录如下所示。

```
preprocessed_root (lrs2_preprocessed)
 ├── main
|   ├── Folders with five-digit numbered video IDs ( 00001 )
|   |   ├── *.jpg
|   |   ├── audio.wav
|   ├── 00001
|   |   ├── *.jpg
|   |   ├── audio.wav
```

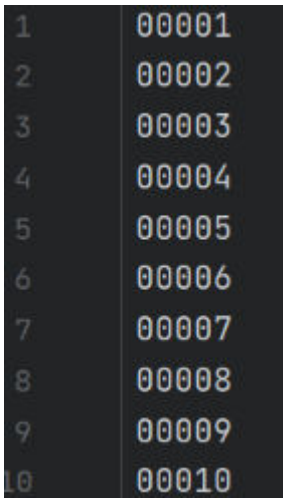
d. 将LRS2文件列表中的.txt文件（train、val）放入该filelists文件夹中。

图 5-8 filelists 文件夹



train.txt和val.txt内容参考如下，为处理后视频数据的目录名字。

图 5-9 train.txt 和 val.txt 内容



3. 训练专家唇形同步鉴别器。

如果使用LRS2数据集，可选择跳过此步骤。如果使用自己的数据集，训练命令参考如下。

```
python color_syncnet_train.py --data_root ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel/
syncnet_model/ --checkpoint_path ./checkpoints/lipsync_expert.pth
```

参数说明：

- --data_root： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
- --checkpoint_dir： 此目录用于保存模型。

- -checkpoint_path ：（可选）可基于此目录的lipsync_expert模型继续进行训练，如果重新训练则不需要此参数。

默认每10000 step保存一次模型。

4. 训练Wav2Lip模型。

   训练Wav2Lip模型时需要使用专家唇形同步鉴别器。可以使用上一步**3**中的训练结果，也可以直接下载官方提供的**预训练权重**来使用。

   具体训练命令如下。

```
python wav2lip_train.py --data_root  ./lrs2_preprocessed/main/ --checkpoint_dir ./savedmodel --
syncnet_checkpoint_path ./checkpoints/lipsync_expert.pth --checkpoint_path ./checkpoints/wav2lip.pth
```

   参数说明：

   - --data_root ： 处理后的视频数据目录，与train.txt内容拼接后得到单个数据目录，例如：lrs2_preprocessed/main/00001。
   - --checkpoint_dir ： 此目录用于保存模型。
   - --syncnet_checkpoint_path ： 专家鉴别器的目录。
   - --checkpoint_path ：（可选）可基于此目录的Wav2Lip模型继续进行训练，如果重新训练则不需要此参数。

   默认每3000 step保存一次模型。

   专家鉴别器的评估损失应降至约 0.25，Wav2Lip评估同步损失应降至约 0.2，以获得良好的结果。

常见问题

如果训练时遇到报错ImportError: /usr/lib64/libc.so.6: version `GLIBC_2.34' not found，是由于编译Python的glibc环境版本过旧导致，建议重新安装python。

重新安装python命令如下。

```
# 输入如下命令，待conda界面准备完成后输入y，等待自动下载安装
conda create --name py310 python=3.10
```

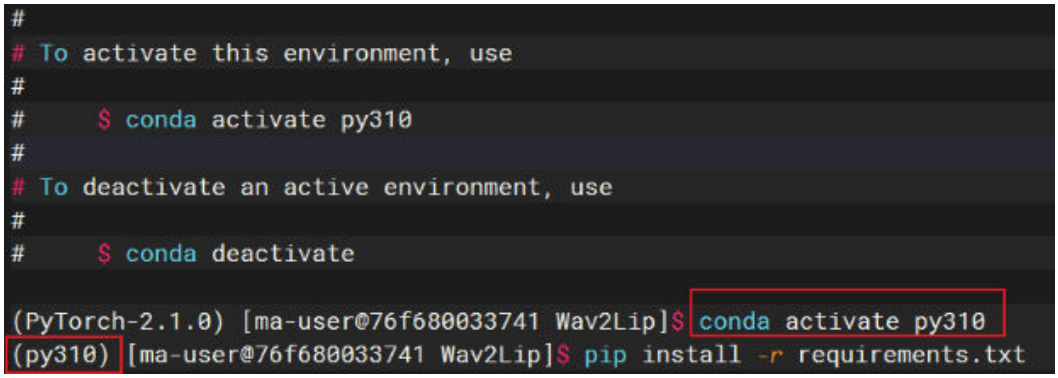
参数说明：

- --name：该参数为新环境名字，可以自定义一个，此处以py310举例。
- python=新环境Python版本

```
# 完成后输入如下命令激活新环境
conda activate py310
```

激活新conda环境后控制台显示（py310）即为切换成功，如下图所示。

图 5-10 激活新 conda 环境



6 GPU 业务迁移至昇腾训练推理

6.1 ModelArts 昇腾迁移调优工具总览

ModelArts集成了多个昇腾迁移调优工具，方便您在ModelArts平台环境中进行训练推理迁移、精度调试、性能调优等工作，您可在下表中查看当前ModelArts支持的昇腾迁移调优工具及对应指导。

表格中的部分工具已集成到ModelArts基础镜像中（镜像地址详见**基础镜像**章节），若您使用的是ModelArts基础镜像，可先尝试直接使用工具命令，如果相关命令不存在则需要参考工具安装指导自行安装。

表 6-1 ModelArts 昇腾迁移调优工具总览表

使用场景	类别	工具名称	工具描述	工具安装	使用指导
PyTorch GPU训练迁移至PyTorch NPU训练	训练迁移	Transfer2NPU	**代码自动迁移工具**，通过简单import命令可将PyTorch训练脚本从GPU平台迁移至NPU平台运行。	包含在torch_npu包中。	• **自动迁移工具使用指导** • **训练业务代码适配昇腾PyTorch代码适配**
		PyTorch Analyse	**迁移分析工具**，可以使用工具扫描用户的训练脚本，识别出源码中不支持的torch API和cuda API信息。	包含在cann toolkit中。	**分析工具使用指导**
	精度调试	api_accuracy_checker	**精度API预检工具**，能在昇腾NPU上扫描用户训练模型中所有API，输出**单API**级别的精度情况的诊断和分析。	下载工具**源码**使用。	**Ascend模型精度预检工具**

使用场景	类别	工具名称	工具描述	工具安装	使用指导
		ptdbg_ascend	**精度整网对比工具**，可以对NPU整网数据进行dump并与GPU dump数据进行比较，输出**整网**的精度情况的诊断和分析。 • 支持模块级dump，可按模块级别做对比。 • 支持溢出检测功能，可检测API的溢出情况。 • 支持梯度监控功能，可辅助定位长训精度问题。	下载工具**whl包安装**使用，推荐使用最新版本。	**PyTorch精度工具**
	性能调优	PyTorch Profiler	**性能采集工具**，在训练脚本中调用Ascend PyTorch Profiler接口，可在训练过程中采集性能数据文件，包括PyTorch层算子信息、CANN层算子信息、底层NPU算子信息、以及算子内存占用信息等。	包含在torch_npu包中。	**Ascend PyTorch Profiler数据采集与分析**
		MA-Advisor	**性能自动诊断工具**，采集好的Profiling数据通过该工具进行自动扫描分析，可给出性能瓶颈的诊断和修改建议。当迁移开箱性能较低时，通过该工具给出的建议修改代码后，通常可提升10%~30%。	whl包，地址见教程中下载链接。	**自动诊断工具MA-Advisor**
		compare_tools	**性能比对工具**，将在GPU和NPU采集的Profiling数据进行性能拆解和分类比对，展示算子、通信、内存等类别的性能比对数据。	下载工具**源码**使用。	**性能比对工具**

使用场景	类别	工具名称	工具描述	工具安装	使用指导
		cluster_analyse	**集群性能分析工具**，采集好的多机Profiling数据可通过该工具分析集群通信耗时、通信带宽矩阵等内容，从而辅助定位慢卡、慢节点等问题。工具的输出数据为csv格式，可直接拖入Ascend Insight进行可视化查看。	下载工具**源码**使用。	**集群分析工具**
		Ascend Insight	**性能可视化工具**，采集好的profiling数据可通过该工具进行可视化展示，辅助人工进行profiling数据查看和分析。	windows版本工具，下载链接见教程内。	**Ascend Insight用户指南**
PyTorch GPU推理迁移至MindSpore Lite NPU推理	模型迁移	Tailor	Mindspore-lite模型转换、精度误差分析、性能分析。	whl包，地址见教程中下载链接。	**Tailor使用指导**
	性能调优	msprof	msprof命令行工具提供了AI任务运行性能数据、昇腾AI处理器系统数据等性能数据的采集和解析能力。	包含在cann toolkit中。	**msprof**
		AOE	自动调优工具，提供子图调优和算子调优功能，在静态shape场景下有较好的调优效果。推荐在mindspore-lite离线推理场景下使用。	包含在cann toolkit中。	**AOE性能自动调优**
		AKG	MindSpore自动调优工具，提供算子自动优化和算子自动融合的功能，推荐在mindspore-lite离线推理场景下使用。	下载工具**源码**使用。	**AKG**

使用场景	类别	工具名称	工具描述	工具安装	使用指导
PyTorch GPU推理迁移至PyTorch ascend-vllm /atb/ torchair推理	模型迁移	-	需要用户自行代码适配，或者使用ModelArts迁移好的模型。	-	ModelArts迁移好的模型可参考最佳实践中的案例，使用AscendCloud软件包中的模型，例如：**主流开源大模型基于DevServer适配PyTorch NPU推理指导**。
	模型量化	model slim	**模型量化工具**，通过量化提升模型的推理性能。	包含在cann toolkit中。	**ModelSlim**
	精度调试	ait llm	**大模型精度调试工具**，支持加速库（atb）和torchair的大模型推理的精度数据dump及比对功能，辅助大模型推理精度问题定位。	下载工具**whl包安装**使用，推荐使用最新版本。	**大模型推理精度工具**

6.2 基于 LLM 模型的 GPU 训练业务迁移至昇腾指导

6.2.1 场景介绍

本文以ChatGLM-6B为例，介绍如何将模型迁移至昇腾设备上训练、模型精度对齐以及性能调优。

主要包含以下步骤：

- **环境准备**
- **迁移适配**
- **精度对齐**
- **性能调优**

6.2.2 环境准备

步骤1 开通裸金属服务器资源（请见**DevServer资源开通**），并在裸金属服务器上搭建迁移环境请见**裸金属服务器环境配置指导**。

步骤2 启动华为云预置镜像环境，本案例使用的贵阳一的镜像环境。

```
#shell
docker run --privileged --name chatglm-test --cap-add=SYS_PTRACE -e ASCEND_VISIBLE_DEVICES=0-7 -u=0 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.11.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b-20231107190844-50a1a83  bash
```

此处“-e ASCEND_VISIBLE_DEVICES” 用于指定容器中启动的NPU device，0-7表示从0-7号卡，请按照实际NPU卡情况修改。

步骤3 安装相关依赖库。

ChatGLM-6B是完全基于Python开发的模型，训练之前需要事先安装与之依赖的Python库。其中部分依赖库可以使用pip工具安装，执行如下脚本：

```
#shell
pip install rouge_chinese nltk jieba sentencepiece datasets==2.12.0 fsspec==2022.11.0 transformers==4.29.2
deepspeed==0.9.2
```

与昇腾NPU适配的依赖库有torch_npu，多卡训练也需要deepspeed_npu，本文适配的版本如下：deepspeed_npu(0.1)，torch_npu(1.11)。其中torch_npu在镜像环境中已经预置安装，deepspeed_npu安装配置详见**deepspeed_npu**。

此外， transformers执行需要高版本的scikit-learn、acclerate，详见**常见问题5**、**常见问题6**。此处执行升级命令：

```
#shell
pip install scikit-learn accelerate --upgrade
```

transformers库的training_args.py有部分操作是适配的cuda设备，详见**常见问题7**，本文使用昇腾ModelZoo的适配版本脚本替换（ **下载链接** ）。

步骤4 下载ChatGLM-6B源代码、模型权重与数据集到容器环境。

- 源代码：**chatglm-6B**
- 模型权重：**weights**
- 数据集：**Firefly(流萤)**、**ADGEN (广告生成)**

说明

- 源代码、模型权重使用的清华官方在Github和Hugging Face开源的版本，源代码适配的main分支，权重当前使用1d240ba固定分支。其他分支版本理论上也可以进行迁移工作，不过注意可能由于权重不同原因最后训练结果也不太一致，此处建议您使用固定分支进行迁移。
- 数据集Firefly为本文用于多卡训练使用的数据集，数据集ADGEN为ChatGLM-6B ptuning训练适配的数据集，如果您运行环境为单卡环境下载数据集ADGEN。

----结束

6.2.3 迁移适配

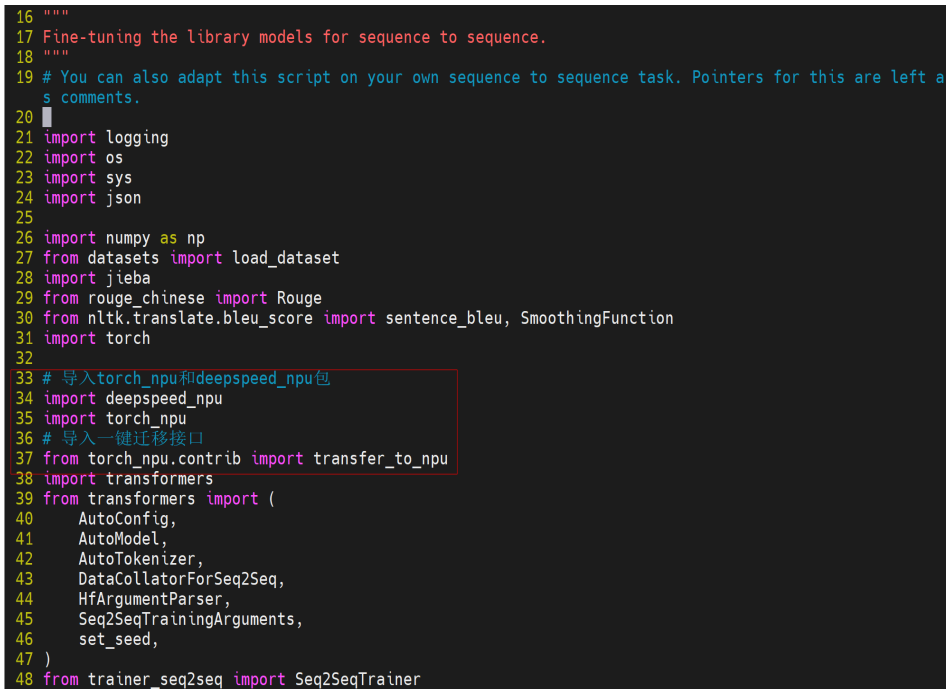
本文以PyTorch框架在NPU上完成自动迁移为例，对适配过程需要修改的部分进行说明。并且针对单卡环境以及单机多卡deepspeed环境提供训练脚本。无特别说明，以ChatGLM-6B源代码根目录作为当前目录。

自动迁移适配

修改“ptuning/main.py”，添加deepspeed_npu、torch_npu、transfer_to_npu依赖库，如下图所示。

```
# 导入deepspeed_npu和torch_npu
import deepspeed_npu
import torch_npu
# 导入一键迁移接口
from torch_npu.contrib import transfer_to_npu
```

图 6-1 自动迁移适配



```
"""
Fine-tuning the library models for sequence to sequence.
"""
# You can also adapt this script on your own sequence to sequence task. Pointers for this are left as comments.

import logging
import os
import sys
import json

import numpy as np
from datasets import load_dataset
import jieba
from rouge_chinese import Rouge
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
import torch

# 导入torch_npu和deepspeed_npu包
import deepspeed_npu
import torch_npu
# 导入一键迁移接口
from torch_npu.contrib import transfer_to_npu
import transformers
from transformers import (
    AutoConfig,
    AutoModel,
    AutoTokenizer,
    DataCollatorForSeq2Seq,
    HfArgumentParser,
    Seq2SeqTrainingArguments,
    set_seed,
)
from trainer_seq2seq import Seq2SeqTrainer
```

单卡方式训练

单卡执行脚本如下：

```
# ptuning/run_npu_1d.sh
export ASCEND_RT_VISIBLE_DEVICES=0  # 指定 0 号卡对当前进程可见
PRE_SEQ_LEN=128
LR=2e-2

python3 ptuning/main.py \
    --do_train \
    --train_file ${HOME}/AdvertiseGen/train.json \
    --validation_file ${HOME}/AdvertiseGen/dev.json \
    --prompt_column content \
    --response_column summary \
    --overwrite_cache \
    --model_name_or_path ${HOME}/chatglm \
    --output_dir output/adgen-chatglm-6b-pt-$PRE_SEQ_LEN-$LR \
    --overwrite_output_dir \
    --max_source_length 64 \
    --max_target_length 64 \
    --per_device_train_batch_size 4 \
    --per_device_eval_batch_size 1 \
    --gradient_accumulation_steps 1 \
    --predict_with_generate \
    --max_steps 3000 \
    --logging_steps 10 \
    --save_steps 1000 \
    --learning_rate $LR \
    --pre_seq_len $PRE_SEQ_LEN \
    --local_rank -1
```

通过设定ASCEND_RT_VISIBLE_DEVICES环境变量为0，控制0号卡对当前进程可见，PRE_SEQ_LEN和LR分别是soft prompt长度和训练的学习率，可以进行调节以取得最佳的效果。此外，这里去掉了int 4量化默认为FP16精度。${HOME} 目录需要根据读者实际数据集及模型路径匹配，适配的数据集是ADGEN数据集，如果需要读者也可以使用自定义的数据集训练，具体请参考**使用自己数据集**。另外通过指定local_rank为-1为

单卡模式，多卡模式下无需指定，会默认启动DistributedDataParallel（DDP）多卡并行模式，具体详情见**常见问题1**。GPU环境单卡执行同样需要指定local_rank为 -1。

多卡分布式执行

PyTorch框架下常见的多卡分布式执行主要包括**DataParallel**（DP）和**Distributed Data Parallel**（DDP）。torch_npu环境下针对DDP场景的多卡训练有提供支持，具体请参见**迁移单卡脚本为多卡脚本**。此外，针对deepspeed环境，昇腾有专门的适配环境deepspeed-npu。在此提供一种基于deepspeed的多卡训练脚本，内容如下：

```
# ds_run_npu.sh
LR=1e-4
TRAIN_FILE=${HOME}/YeungNLPfirefly-train-1.1M/firefly-train-1.1M.jsonl
PER_DEVICE_TRAIN_BATCH_SIZE=4
GRADIENT_ACCUMULATION_STEPS=32
MODEL_DIR=${HOME}/chatglm
OUTPUT_DIR=${HOME}/ChatGLM-6B-main/ptuning/output
DS_CONFIG=${HOME}/ChatGLM-6B-main/ptuning/ds_config.json
APP_SCRIPT=${HOME}/ChatGLM-6B-main/ptuning/main.py
MASTER_PORT=$(shuf -n 1 -i 10000-65535)

deepspeed --num_gpus=8 --master_port $MASTER_PORT ${APP_SCRIPT} \
    --deepspeed ${DS_CONFIG} \
    --log_level debug \
    --model_name_or_path ${MODEL_DIR} \
    --train_file ${TRAIN_FILE} \
    --prompt_column input \
    --response_column target \
    --max_source_length 512 \
    --max_target_length 512 \
    --output_dir ${OUTPUT_DIR}/chatglm-6b-${LR} \
    --per_device_train_batch_size ${PER_DEVICE_TRAIN_BATCH_SIZE} \
    --per_device_eval_batch_size 1 \
    --gradient_accumulation_steps ${GRADIENT_ACCUMULATION_STEPS} \
    --gradient_checkpointing False \
    --num_train_epochs 1 \
    --predict_with_generate \
    --logging_steps 10 \
    --save_strategy "steps" \
    --save_total_limit 3 \
    --learning_rate $LR \
    --dataloader_num_workers 60 \
    --preprocessing_num_workers 60 \
    --do_train \
    --overwrite_output_dir \
    --max_steps 100 \
    --fp16
```

LR、PER_DEVICE_TRAIN_BATCH_SIZE、GRADIENT_ACCUMULATION_STEPS分别代表学习率、单个设备训练批次大小、梯度累计步数，作为超参数可以调优获得较好模型。同样，${HOME} 需要根据数据集模型等路径做对应替换，这里脚本适配的数据集是Firefly，其中deepspeed使用了**zero 1**显存优化方式，配置方式如下：

```
{
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "initial_scale_power": 16,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "bf16": {
    "enabled": "auto"
  },
  "optimizer": {
```

```
        "type": "AdamW",
        "params": {
            "lr": "auto",
            "betas": "auto",
            "eps": "auto",
            "weight_decay": "auto"
        }
    },
    "scheduler": {
        "type": "WarmupLR",
        "params": {
            "warmup_min_lr": "auto",
            "warmup_max_lr": "auto",
            "warmup_num_steps": "auto"
        }
    },
    "zero_optimization": {
        "stage": 1,
        "offload_optimizer": {
            "device": "cpu",
            "pin_memory": true
        },
        "contiguous_gradients": true,
        "overlap_comm": true,
        "allgather_partitions": true,
        "reduce_scatter": true,
        "allgather_bucket_size": 2e8,
        "reduce_bucket_size": 4e8
    },
    "flops_profiler":{
        "enabled": true,
        "profile_step": 1,
        "module_path": -1,
        "top_modules": 1,
        "detailed": false,
        "output_file": null
    },
    "zero_allow_untested_optimizer": "true",
    "gradient_clipping": "auto",
    "gradient_accumulation_steps": "auto",
    "train_batch_size": "auto",
    "train_micro_batch_size_per_gpu": "auto",
    "wall_clock_breakdown": false
}
```

6.2.4 精度对齐

精度问题是指模型从GPU设备迁移到昇腾NPU设备之后由于软硬件差异引入的精度问题。根据是否在单卡环境下，可分为单卡精度问题与多卡精度问题。多卡相对于单卡，会有卡与卡之间的通信，这可能也是精度偏差的一种来源。所以多卡的精度对齐问题相对于单卡会更复杂。不过针对多卡的精度问题，可以分步骤先保证单卡对齐精度，然后分析通信过程的偏差。本文针对单卡的情形给出基于ptdbg-ascend精度对比工具的精度排查过程。

loss 曲线对比

训练结束后，在output_dir参数指定目录下会输出trainer_state.json文件，该文件保存了训练过程loss以及learning_rate的log信息。

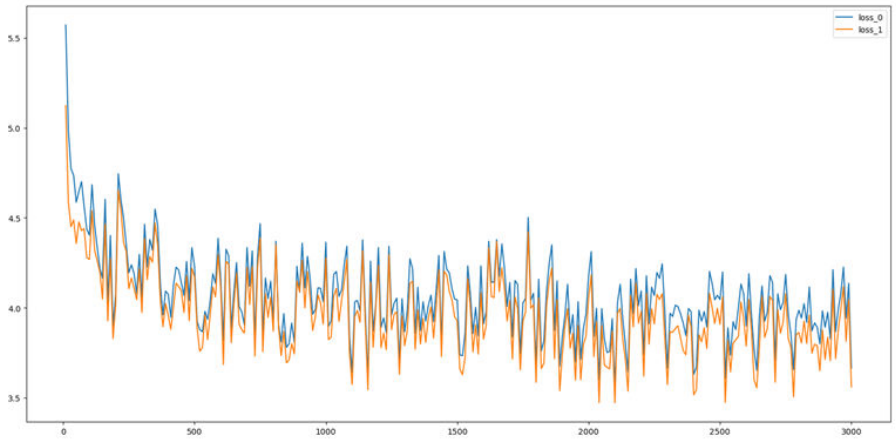
将GPU设备训练输出的trainer_state.json文件重命名为trainer_state_gpu.json，并复制到NPU节点的容器内，将NPU设备训练输出的trainer_state.json文件重命名为trainer_state_npu.json。

对其进行解析就可以获取loss信息，这里可以使用如下脚本进行loss曲线的绘制。

```
# compare_metric.py
import json
import os
from typing import List, Dict

import matplotlib.pyplot as plt
import numpy as np

## 解析 json 文件
def load_trainer_status(file_path):
    with open(file_path, "r") as f:
        trainer_status = json.load(f)
    return trainer_status.get("log_history")

def plot_curve(data_source: List[Dict], tags: List[str]):
    fig, ax = plt.subplots()
    for tag in tags:
        # print(data_source[0], len(data_source[0]))
        # assert all([tag in status.keys() for status in data_source]), f"Tag {tag} is missing for data source."
        for index, source in enumerate(data_source):
            y = []
            x = []
            for log in source:
                x.append(log.get("step"))
                y.append(log.get(tag))
            ax.plot(x, y, label=f"{tag}_{index}")

    ax.legend()
    plt.savefig("loss.png")

if __name__ == "__main__":
    state_npu_path = os.path.join("trainer_state_npu.json")
    state_gpu_path = os.path.join("trainer_state_gpu.json")
    state_npu = load_trainer_status(state_npu_path)
    state_gpu = load_trainer_status(state_gpu_path)
    plot_curve([state_npu, state_gpu], ["loss"])
```

对比单卡模式下NPU和GPU训练曲线，发现loss曲线下降趋势不一致。这说明迁移的模型存在精度偏差。

图 6-2 loss 曲线对比



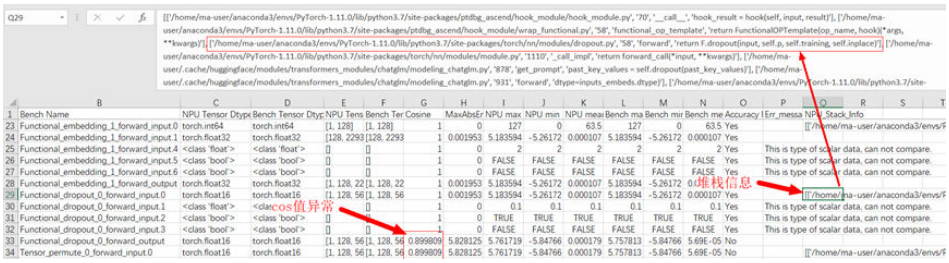
图中蓝色loss_0是NPU迭代曲线，黄色loss_1是GPU的迭代曲线。

问题定位解决

使用ptdbg_ascend工具dump全网数据，dump接口设置方法具体参考**PyTorch精度工具**。dump完成后compare GPU和NPU结果进行分析。

1. dropout算子引入了随机性偏差，如下图：

图 6-3 随机性偏差



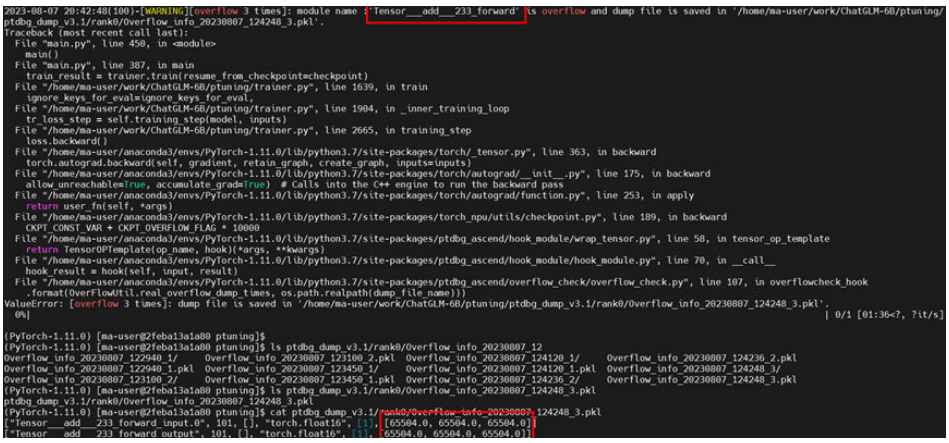
根据堆栈信息定位得知dropout是使用的torch.nn.Dropout()，为消除随机性需要将随机因子p改为0或者1，此处是将model_chatglm.py中随机因子改为了0，如下修改：

图 6-4 随机因子改为 0

```
854         if self.pre_seq_len is not None:
855             for param in self.parameters():
856                 param.requires_grad = False
857             self.prefix_tokens = torch.arange(self.pre_seq_len).long()
858             self.prefix_encoder = PrefixEncoder(config)
859             # self.dropout = torch.nn.Dropout(0.1)
860             self.dropout = torch.nn.Dropout(0.0)
```

2. 使用ptdbg修改register_hook方式做精度溢出检查。结果显示Tensor___add___233_forward执行时有溢出，这里使用浮点数精度的是float16，结果显示输入的最大、最小、平均值都为65504（float16的精度范围是-65504至65504），如下图所示：

图 6-5 精度溢出检查



因为在NPU下对INF和NAN的支持默认是饱和模式，会将INF置为MAX，NAN置为0，此处Tensor___add___233_forward的输入输出都是fp16的，会将Inf置为65504。 但是在GPU下采用的是INF_NAN模式（保留INF及NAN的结果），所以

在做精度对比时先修改NPU支持模式为INF_NAN模式与GPU保持一致，请参考**INF_NAN_MODE_ENABLE**。

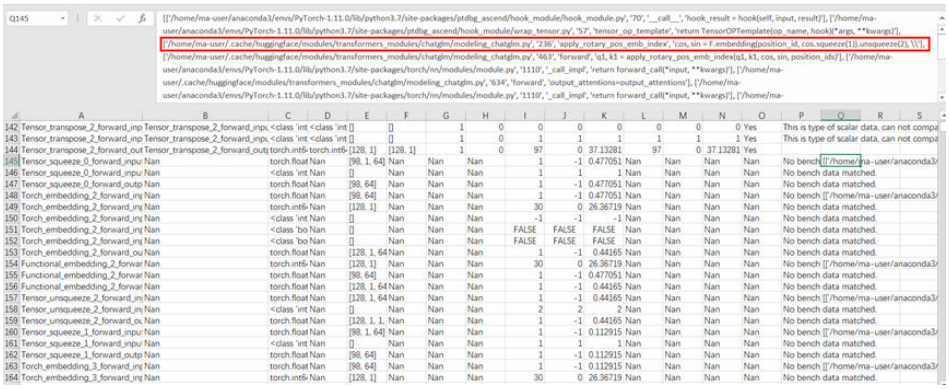
开启INF_NAN模式方式命令如下：

```
#shell
export INF_NAN_MODE_ENABLE=1
```

修改之后再次做溢出检查显示所有API正常，无溢出情况。

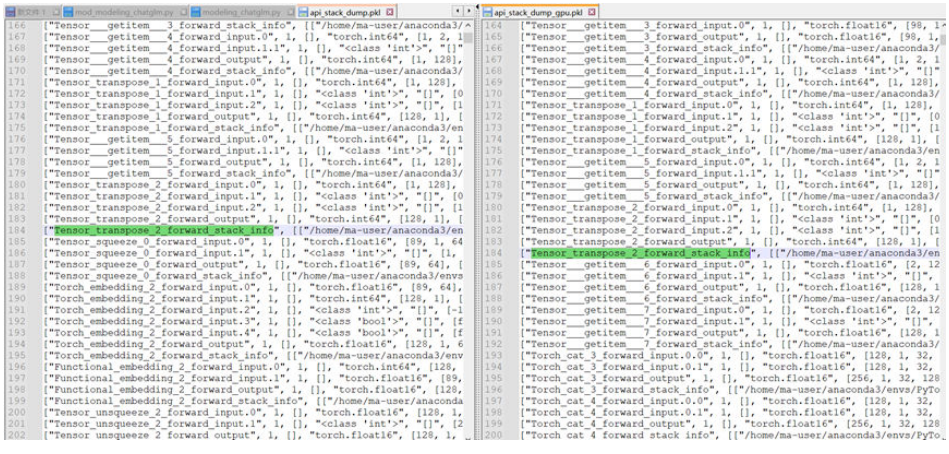
3. GPU dump数据缺失，从Tensor_transpose_2_forward_output之后没有与NPU对应的bench data数据。

图 6-6 GPU dump 数据



在pkl文件中找到对应缺失的位置，发现Tensor_transpose_2_forward_output之后，NPU下一个执行的算子是Tensor_squeeze_0_forward_input，而GPU下一个执行的算子是Tensor___getitem___6_forward_input。

图 6-7 api_stack_dump.pkl



根据stack信息查找到对应源码的代码行，发现对应函数上添加了@torch.jit.script装饰器，经过调试发现，GPU也执行了这个函数，但是没有dump算子执行信息，而且pdb无法在函数中正常中断，删除此装饰器后，GPU能够正常dump数据。

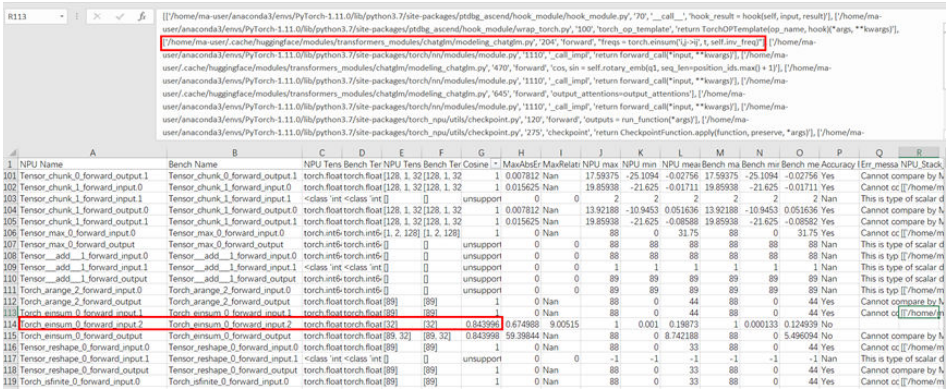
图 6-8 删除@torch.jit.script 装饰器

```
#@torch.jit.script
def apply_rotary_pos_emb_index(q, k, cos, sin, position_id):
    # position_id: [sq, b], q, k: [sq, b, np, hn], cos: [sq, 1, hn] -> [sq, b, 1, hn]
    cos, sin = F.embedding(position_id, cos.squeeze(1)).unsqueeze(2), \
        F.embedding(position_id, sin.squeeze(1)).unsqueeze(2)
    q, k = (q * cos) + (rotate_half(q) * sin), (k * cos) + (rotate_half(k) * sin)
    return q, k
```

加了@torch.jit.script装饰器，torch_npu能采到数据，而GPU上则不行的原因为：@torch.jit.script装饰器会将装饰函数作为ScriptFunction对象返回，不会产生dump数据。而目前该装饰器在torch_npu下不生效，NPU会按照普通函数执行，因此能够采集到数据。从精度对比角度考虑，先删除@torch.jit.script可以保证这块GPU和NPU dump的数据对齐。

4. compare表中Cosine列第一个出现偏差的位置，为einsum算子的输入。

图 6-9 Cosine 列的偏差



查看堆栈信息发现是self.inv_freq的值存在精度偏差，再追溯到self.inv_freq的定义片段。

图 6-10 inv_freq 的定义片段

```
class RotaryEmbedding(torch.nn.Module):
    def __init__(self, dim, base=10000, precision=torch.half, learnable=False):
        super().__init__()
        inv_freq = 1. / (base ** (torch.arange(0, dim, 2).float() / dim))
        inv_freq = inv_freq.half()
        self.learnable = learnable
        if learnable:
            self.inv_freq = torch.nn.Parameter(inv_freq)
            self.max_seq_len_cached = None
        else:
            self.register_buffer('inv_freq', inv_freq)
            self.max_seq_len_cached = None
            self.cos_cached = None
            self.sin_cached = None
        self.precision = precision
```

通过构造该计算公式，发现在x86上：torch+CPU和torch+GPU以及aarch64 torch+NPU场景的结果都是一致的，而aarch64 torch+CPU结果不同，如下：

图 6-11 torch+CPU

```
import torch

inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).float() / 64))
inv_freq.min()
```

torch + CPU x86

```
tensor(0.0001)
```

图 6-12 torch+GPU

```
import torch

inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).float().cuda() / 64))
inv_freq.min()
```

torch + CUDA x86

```
tensor(0.0001, device='cuda:0')
```

图 6-13 aarch64 torch+NPU

```
>>> import torch
>>> import torch_npu
Warning : ASCEND_HOME_PATH environment variable is not set.
>>> inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).float().npu() / 64))
>>> inv_freq.min()
tensor(0.0001, device='npu:0')
```

torch npu

图 6-14 aarch64 torch+CPU

```
>>> import torch
>>> inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).float() / 64))
>>> inv_freq.min()
tensor(0.0010)
```

torch + Ascend pytorch

而inv_freq恰好都是在CPU上初始化的。修改NPU版代码，强制使用torch+NPU进行初始化后，可以消除einsum算子输入偏差的问题。修改如下：

```
inv_freq = 1. / (base ** (torch.arange(0, dim, 2).float().npu() / dim))
```

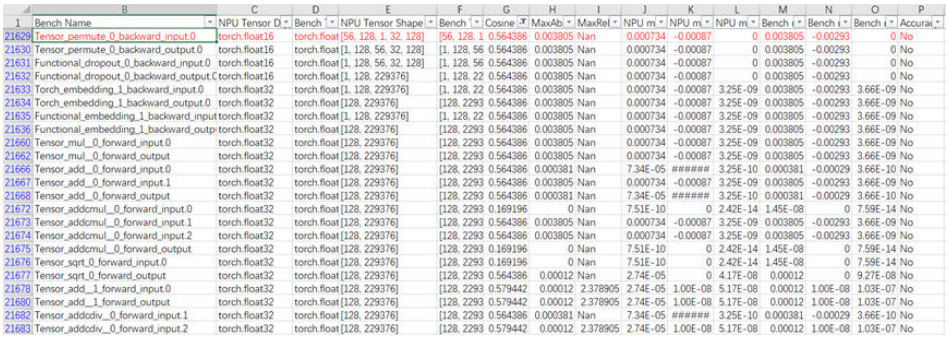
另外的一种修改方式是转换到dobule下进行计算。

图 6-15 转换到 dobule 下进行计算

```
>>> import torch
>>> inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).float() / 64))
>>> inv_freq.min()
tensor(0.0010)
>>> inv_freq = 1. / (10000 ** (torch.arange(0, 64, 2).double() / 64))
>>> inv_freq.min()
tensor(0.0001, dtype=torch.float64)
```

5. 修复上述问题后，Cosine值第一次出现偏差的位置为permute算子，在backward阶段作为input引入。

图 6-16 permute 算子偏差



	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Bench Name	NPU Tensor D	Bench	NPU Tensor Shape	Bench	Cosine	MaxAb	MaxRel	NPU mi	NPU mi	NPU mi	Bench (	Bench (	Bench (	Accura
21629	Tensor_permute_0_backward_input.0	torch.float16	torch.float	[56, 128, 1, 32, 128]	[56, 128, 1	0.564386	0.003805	Nan	0.000734	-0.00087	0	0.003805	-0.00293	0	No
21630	Tensor_permute_0_backward_output.0	torch.float16	torch.float	[1, 128, 56, 32, 128]	[1, 128, 56	0.564386	0.003805	Nan	0.000734	-0.00087	0	0.003805	-0.00293	0	No
21631	Functional_dropout_0_backward_input.0	torch.float16	torch.float	[1, 128, 56, 32, 128]	[1, 128, 56	0.564386	0.003805	Nan	0.000734	-0.00087	0	0.003805	-0.00293	0	No
21632	Functional_dropout_0_backward_output.(	torch.float16	torch.float	[1, 128, 229376]	[1, 128, 22	0.564386	0.003805	Nan	0.000734	-0.00087	0	0.003805	-0.00293	0	No
21633	Torch_embedding_1_backward_input.0	torch.float32	torch.float	[1, 128, 229376]	[1, 128, 22	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21634	Torch_embedding_1_backward_output.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21635	Functional_embedding_1_backward_input	torch.float32	torch.float	[1, 128, 229376]	[1, 128, 22	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21636	Functional_embedding_1_backward_outp	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21660	Tensor_mul_0_forward_input.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21662	Tensor_mul_0_forward_output	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21666	Tensor_add_0_forward_input.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.000381	Nan	7.34E-05	######	3.25E-10	0.000381	-0.00029	3.66E-10	No
21667	Tensor_add_0_forward_input.1	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21668	Tensor_add_0_forward_output	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.000381	Nan	7.34E-05	######	3.25E-10	0.000381	-0.00029	3.66E-10	No
21672	Tensor_addcmul_0_forward_input.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.169196	0	Nan	7.51E-10	0	2.42E-14	1.45E-08	0	7.59E-14	No
21673	Tensor_addcmul_0_forward_input.1	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21674	Tensor_addcmul_0_forward_input.2	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.003805	Nan	0.000734	-0.00087	3.25E-09	0.003805	-0.00293	3.66E-09	No
21675	Tensor_addcmul_0_forward_output	torch.float32	torch.float	[128, 229376]	[128, 2293	0.169196	0	Nan	7.51E-10	0	2.42E-14	1.45E-08	0	7.59E-14	No
21676	Tensor_sqrt_0_forward_input.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.169196	0	Nan	7.51E-10	0	2.42E-14	1.45E-08	0	7.59E-14	No
21677	Tensor_sqrt_0_forward_output	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.00012	Nan	2.74E-05	0	4.17E-08	0.00012	0	9.27E-08	No
21678	Tensor_add_1_forward_input.0	torch.float32	torch.float	[128, 229376]	[128, 2293	0.579442	0.00012	2.378905	2.74E-05	1.00E-08	5.17E-08	0.00012	1.00E-08	1.03E-07	No
21680	Tensor_add_1_forward_output	torch.float32	torch.float	[128, 229376]	[128, 2293	0.579442	0.00012	2.378905	2.74E-05	1.00E-08	5.17E-08	0.00012	1.00E-08	1.03E-07	No
21682	Tensor_addcdiv_0_forward_input.1	torch.float32	torch.float	[128, 229376]	[128, 2293	0.564386	0.000381	Nan	7.34E-05	######	3.25E-10	0.000381	-0.00029	3.66E-10	No
21683	Tensor_addcdiv_0_forward_input.2	torch.float32	torch.float	[128, 229376]	[128, 2293	0.579442	0.00012	2.378905	2.74E-05	1.00E-08	5.17E-08	0.00012	1.00E-08	1.03E-07	No

由于在backward阶段ptdbg-ascend没有输出执行的堆栈信息，先查找了Tensor_permute_0在forward阶段相应的堆栈信息。

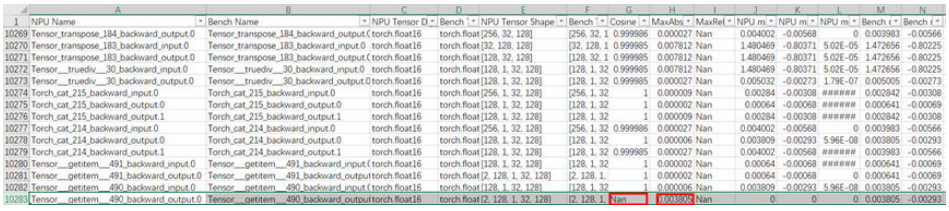
图 6-17 Tensor_permute_0 在 forward 阶段相应的堆栈信息

['/home/ma-user/.cache/huggingface/modules/transformers_modules/chatglm/modeling_chatglm.py', '927', 'get_prompt', 'past_key_values = past_key_values.permute([2, 1, 0, 3, 4]).split(2)'],

可以得知此处进行了换轴操作，但是在forward时输入输出均无精度异常。

因此转换排查思路，全局查找Cosine、MaxAbsErr值和Tensor_permute_0_backward相同的行。发现在Tensor___getitem___490_backward_output.0处MaxAbsErr的值和Tensor_permute_0_backward一样。

图 6-18 Tensor___getitem___490_backward_output.0



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	NPU Name	Bench Name	NPU Tensor D	Bench	NPU Tensor Shape	Bench	Cosine	MaxAbs	MaxRel	NPU mi	NPU mi	NPU mi	Bench (	Bench (
10269	Tensor_transpose_184_backward_output.0	Tensor_transpose_184_backward_output.(	torch.float16	torch.float	[256, 32, 128]	[256, 32, 1	0.999986	0.000027	Nan	0.004002	-0.00568	0	0.003983	-0.00566
10270	Tensor_transpose_183_backward_input.0	Tensor_transpose_183_backward_input.0	torch.float16	torch.float	[32, 128, 128]	[32, 128, 1	0.999985	0.007812	Nan	1.480469	-0.80371	5.02E-05	1.472656	-0.80225
10271	Tensor_transpose_183_backward_output.0	Tensor_transpose_183_backward_output.(	torch.float16	torch.float	[128, 32, 128]	[128, 32, 1	0.999985	0.007812	Nan	1.480469	-0.80371	5.02E-05	1.472656	-0.80225
10272	Tensor__truediv__30_backward_input.0	Tensor__truediv__30_backward_input.0	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	0.999985	0.007812	Nan	1.480469	-0.80371	5.02E-05	1.472656	-0.80225
10273	Tensor__truediv__30_backward_output.0	Tensor__truediv__30_backward_output.0	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	0.999985	0.000027	Nan	0.005032	-0.00273	1.79E-07	0.005005	-0.00273
10274	Torch_cat_215_backward_input.0	Torch_cat_215_backward_input.0	torch.float16	torch.float	[256, 1, 32, 128]	[256, 1, 32	1	0.000009	Nan	0.00284	-0.00308	######	0.002842	-0.00308
10275	Torch_cat_215_backward_output.0	Torch_cat_215_backward_output.0	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	1	0.000002	Nan	0.00064	-0.00068	######	0.000641	-0.00069
10276	Torch_cat_215_backward_output.1	Torch_cat_215_backward_output.1	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	1	0.000009	Nan	0.00284	-0.00308	######	0.002842	-0.00308
10277	Torch_cat_214_backward_input.0	Torch_cat_214_backward_input.0	torch.float16	torch.float	[256, 1, 32, 128]	[256, 1, 32	0.999986	0.000027	Nan	0.004002	-0.00568	0	0.003983	-0.00566
10278	Torch_cat_214_backward_output.0	Torch_cat_214_backward_output.0	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	1	0.000006	Nan	0.003809	-0.00293	5.96E-08	0.003805	-0.00293
10279	Torch_cat_214_backward_output.1	Torch_cat_214_backward_output.1	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	0.999985	0.000027	Nan	0.004002	-0.00568	######	0.003983	-0.00566
10280	Tensor__getitem__491_backward_input.0	Tensor__getitem__491_backward_input.(	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	1	0.000002	Nan	0.00064	-0.00068	######	0.000641	-0.00069
10281	Tensor__getitem__491_backward_output.0	Tensor__getitem__491_backward_output	torch.float16	torch.float	[2, 128, 1, 32, 128]	[2, 128, 1,	1	0.000002	Nan	0.00064	-0.00068	0	0.000641	-0.00069
10282	Tensor__getitem__490_backward_input.0	Tensor__getitem__490_backward_input.(	torch.float16	torch.float	[128, 1, 32, 128]	[128, 1, 32	1	0.000006	Nan	0.003809	-0.00293	5.96E-08	0.003805	-0.00293
10283	Tensor__getitem__490_backward_output.0	Tensor__getitem__490_backward_output	torch.float16	torch.float	[2, 128, 1, 32, 128]	[2, 128, 1,	Nan	0.003807	Nan	0	0	0	0.003805	-0.00293

并且Bench data列的max、min、mean对应值也一致，但是Tensor___getitem___490_backward_output.0在NPU下的max、min、mean值都是0，代表该处是全零的向量。猜想应该是梯度计算错误。使用PyTorch的index_select函数作为getitem函数的替代，对modeling_chatglm.py做如下修改：

图 6-19 modeling_chatglm.py 修改

```
256     if layer_past is not None:
257         # past_key, past_value = layer_past[0], layer_past[1]
258         past_key = layer_past.index_select(0, torch.tensor([0]).to(layer_past.device)).squeeze(0)
259         past_value = layer_past.index_select(0, torch.tensor([1]).to(layer_past.device)).squeeze(0)
```

再次dump对比精度，发现该算子精度问题得到解决。

图 6-20 Tensor_permute_0 精度对比

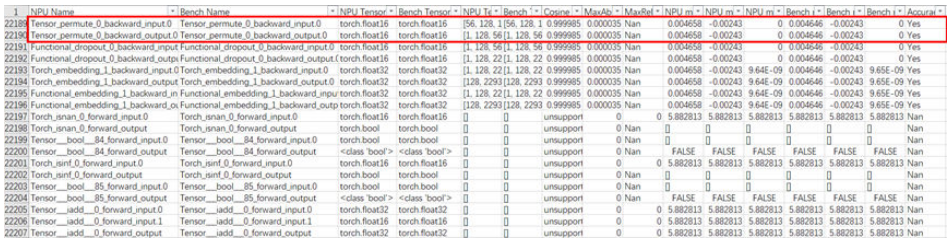
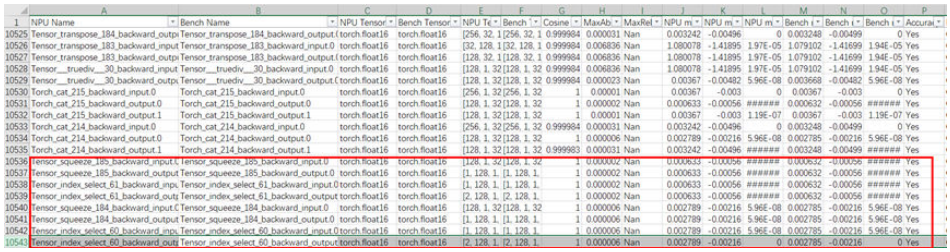
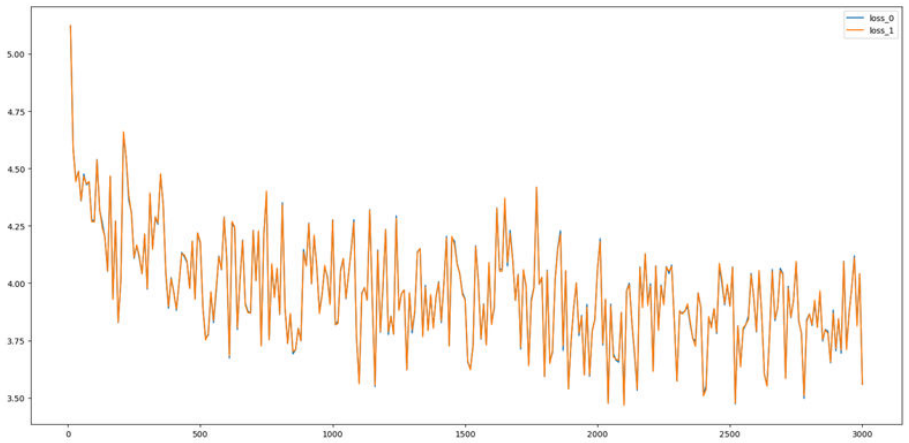


图 6-21 算子精度对比



修改上述问题之后，重新对比精度数据后发现，重新进行训练任务，通过对比NPU和GPU的loss曲线，可以发现，两者的下降趋势几乎是一致的。

图 6-22 loss 曲线



图中蓝色loss_0是NPU的loss曲线，黄色loss_1是GPU的loss曲线。

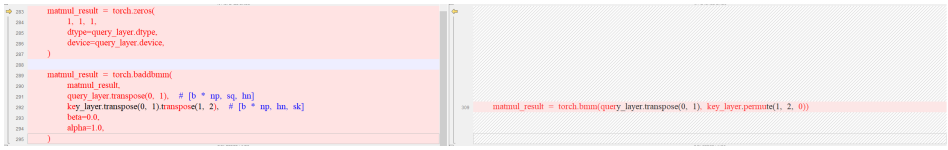
6.2.5 性能调优

算子优化

为了更好地发挥昇腾设备的性能，将ChatGLM-6B原模型中的部分算子替换成了NPU亲和的算子，修改的是modeling_chatglm.py文件，下图通过对比列举了对应的修改方式，图示中左边为原始方式，右边为修改后的方式。

1. 使用torch.bmm替换torch.baddbmm。

图 6-23 torch.bmm 替换



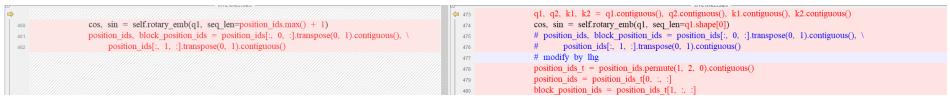
因为toch.baddbmm函数中beta=0.0、alpha=1.0，所以是等价替换。

2. npu_scaled_masked_softmax亲和api替换。

图 6-24 亲和 api 替换

3. 连续性转换。

图 6-25 连续性转换



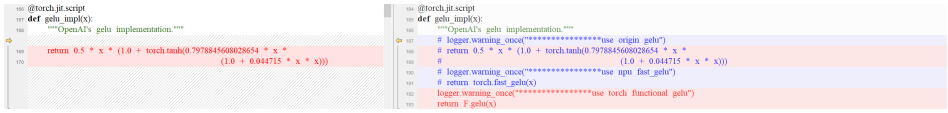
4. 数组切片操作改用torch接口方式。

图 6-26 数组切片操作修改 1

图 6-27 数组切片操作修改 2

5. gelu小算子使用torch的fast_gelu()、gelu()融合算子替换。

图 6-28 融合算子替换



profiling 数据采集

在本例chatglm-6B中，添加profiling接口入口在ptuning/trainer.py的_inner_training_loop()下。具体采集方式参考**Ascend PyTorch Porfiler数据采集与分析方式**。

调优结果

这里对deepspeed单机8卡环境下，调优之前和调优之后的train metrics做了统计，结果如下。

性能基线：

```
***** train metrics *****
epoch                    =       0.06
train_loss               =     3.0146
train_runtime            = 2:15:20.44
train_samples            =    1649399
```

```
train_samples_per_second =      12.61
train_steps_per_second   =      0.012
```

算子调优后结果：

```
***** train metrics *****
epoch                    =       0.06
train_loss               =     3.0128
train_runtime            = 1:39:41.32
train_samples            =    1649399
train_samples_per_second =      17.12
train_steps_per_second   =      0.017
```

6.2.6 常见问题

6.2.6.1 报错提示 RuntimeError: Default process group has not been initialized, please make sure to call init_process_group

问题现象

报错提示RuntimeError: Default process group has not been initialized, please make sure to call init_process_group。

原因分析

原因由于单卡脚本中未添加参数“--local_rank -1”，单卡执行脚本如下，需要指定local_rank为-1为单卡模式。

```
# ptuning/run_npu_1d.sh
export ASCEND_RT_VISIBLE_DEVICES=0  # 指定 0 号卡对当前进程可见
PRE_SEQ_LEN=128
LR=2e-2

python3 ptuning/main.py \
    --do_train \
    --train_file ${HOME}/AdvertiseGen/train.json \
    --validation_file ${HOME}/AdvertiseGen/dev.json \
    --prompt_column content \
    --response_column summary \
    --overwrite_cache \
    --model_name_or_path ${HOME}/chatglm \
    --output_dir output/adgen-chatglm-6b-pt-$PRE_SEQ_LEN-$LR \
    --overwrite_output_dir \
    --max_source_length 64 \
    --max_target_length 64 \
    --per_device_train_batch_size 4 \
    --per_device_eval_batch_size 1 \
    --gradient_accumulation_steps 1 \
    --predict_with_generate \
    --max_steps 3000 \
    --logging_steps 10 \
    --save_steps 1000 \
    --learning_rate $LR \
    --pre_seq_len $PRE_SEQ_LEN \
    --local_rank -1
```

处理方法

单卡执行脚本中添加参数“--local_rank -1”。

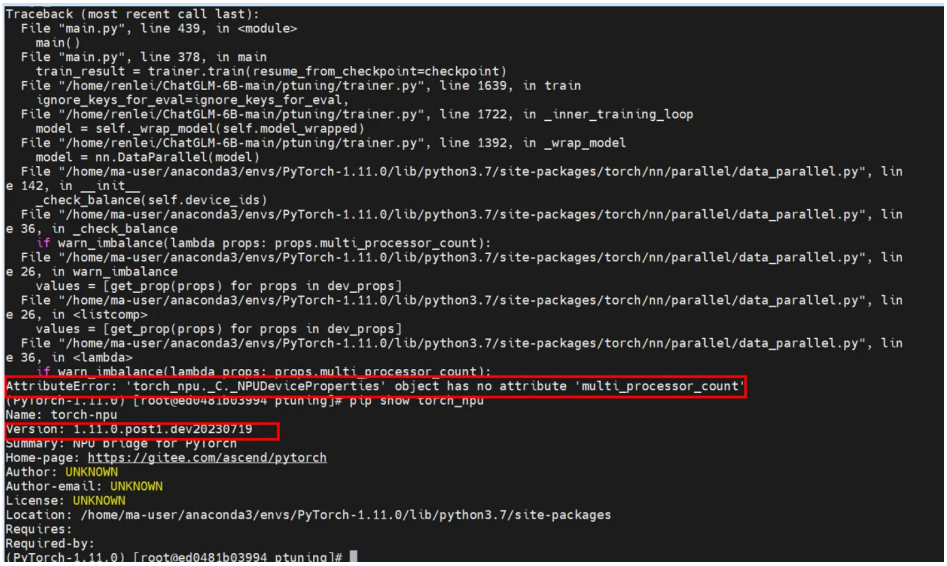
多卡模式下无需指定，会默认启动DistributedDataParallel（DDP）多卡并行模式。GPU环境单卡执行同样需要指定local_rank为 -1。

6.2.6.2 训练运行报错 AttributeError: 'torch_npu._C._NPUDeviceProperties' object has no attribute 'multi_processor_count'

问题现象

训练运行报错“AttributeError: 'torch_npu._C._NPUDeviceProperties' object has no attribute 'multi_processor_count'”。

图 6-29 报错信息



原因分析

这是因为torch_npu当前不支持DataParallel（DP）并行模式。

处理方法

如果是运行单卡模式，在训练脚本中加入export ASCEND_RT_VISIBLE_DEVICES=0（指定 0 号卡对当前进程可见）。多卡环境模式需要运行DDP并行模式。

6.2.6.3 deepspeed 多卡训练报错 TypeError: deepspeed_init() got an unexpected keyword argument 'resume_from_checkpoint'

问题现象

deepspeed多卡训练报错TypeError: deepspeed_init() got an unexpected keyword argument 'resume_from_checkpoint'。

原因分析

由于transformers版本问题，使用transformers==4.29.2。

处理方法

请参见**运行bash ds_train_finetune.sh报错**。

6.2.6.4 Huggingface 缓存目录空间不足，出现 OSError: [Errno 122] Disk quota exceeded

问题现象

报错提示OSError: [Errno 122] Disk quota exceeded。

原因分析

默认情况下，下载数据集缓存目录为“~/.cache/huggingface/dataset”，Huggingface缓存目录空间不足导致出现该报错。

处理方法

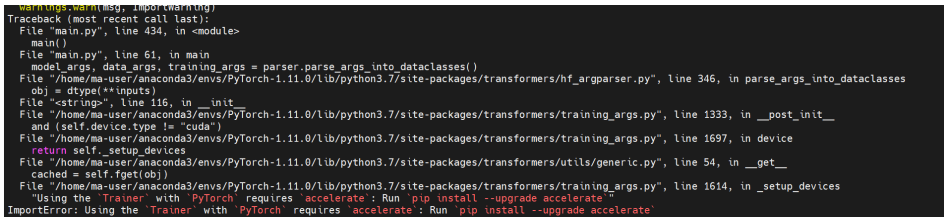
通过环境变量“HF_HOME”设置Huggingface的缓存目录为比较大的路径，或者对“.~/cache”目录扩容。

6.2.6.5 调用 transformers 出现 ImportError: Using the `Trainer` with `PyTorch` requires `accelerate`: Run `pip install --upgrade accelerate`

问题现象

调用transformers出现ImportError: Using the `Trainer` with `PyTorch` requires `accelerate`: Run `pip install --upgrade accelerate`。

图 6-30 报错信息



原因分析

accelerate库版本需要升级。

处理方法

升级accelerate库，执行“pip install accelerate --upgrade”。

6.2.6.6 调用 transformers 出现 ImportError: libcblas.so.3: cannot open shared object file: No such file or directory

问题现象

调用transformers出现“ImportError: libcblas.so.3: cannot open shared object file: No such file or directory”。

原因分析

scikit-learn库版本需要升级。

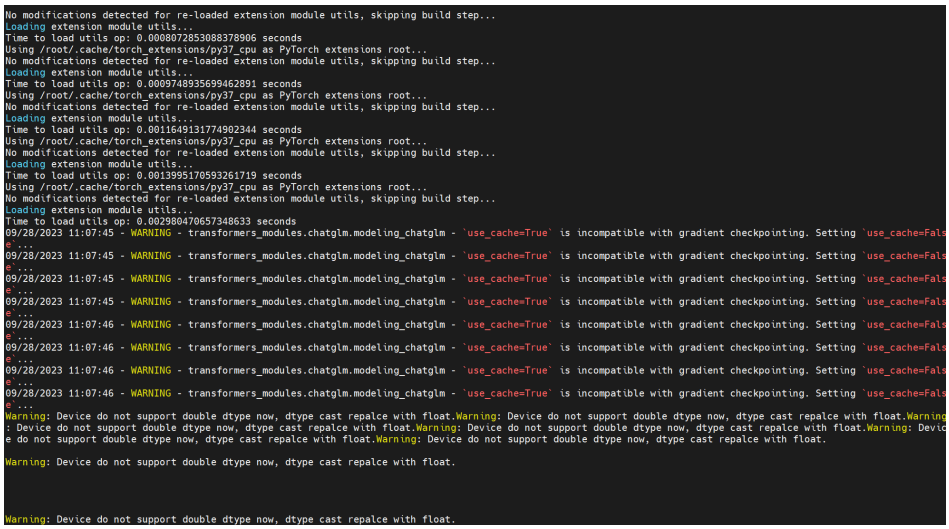
处理方法

升级scikit-learn库，执行“pip install scikit-learn --upgrade”。

6.2.6.7 transformers 调用 cuda 上的操作，或者执行卡死

问题现象

图 6-31 报错信息



```
No modifications detected for re-loaded extension module utils, skipping build step...
Loading extension module utils...
Time to load utils op: 0.0008072853088378906 seconds
Using /root/.cache/torch_extensions/py37_cpu as PyTorch extensions root...
No modifications detected for re-loaded extension module utils, skipping build step...
Loading extension module utils...
Time to load utils op: 0.0009748935699462891 seconds
Using /root/.cache/torch_extensions/py37_cpu as PyTorch extensions root...
No modifications detected for re-loaded extension module utils, skipping build step...
Loading extension module utils...
Time to load utils op: 0.0011649131774902344 seconds
Using /root/.cache/torch_extensions/py37_cpu as PyTorch extensions root...
No modifications detected for re-loaded extension module utils, skipping build step...
Loading extension module utils...
Time to load utils op: 0.0013995170593261719 seconds
Using /root/.cache/torch_extensions/py37_cpu as PyTorch extensions root...
No modifications detected for re-loaded extension module utils, skipping build step...
Loading extension module utils...
Time to load utils op: 0.002980470657348633 seconds
09/28/2023 11:07:45 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:45 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:45 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:45 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:46 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:46 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:46 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
09/28/2023 11:07:46 - WARNING - transformers_modules.chatglm.modeling_chatglm - `use_cache=True` is incompatible with gradient checkpointing. Setting `use_cache=Fals
e`...
Warning: Device do not support double dtype now, dtype cast repalce with float.Warning: Device do not support double dtype now, dtype cast repalce with float.Warning
: Device do not support double dtype now, dtype cast repalce with float.Warning: Device do not support double dtype now, dtype cast repalce with float.Warning: Devic
e do not support double dtype now, dtype cast repalce with float.Warning: Device do not support double dtype now, dtype cast repalce with float.

Warning: Device do not support double dtype now, dtype cast repalce with float.



Warning: Device do not support double dtype now, dtype cast repalce with float.
```

原因分析

transformers库的training_args.py目前适配的是CUDA的部分操作，需要替换为适配NPU的脚本。

处理方法

training_args.py替换为适配NPU的脚本，替换的脚本请见**training_args.py**。

6.3 GPU 训练业务迁移至昇腾的通用指导

6.3.1 训练业务迁移到昇腾设备场景介绍

场景介绍

本文介绍如何将客户已有的PyTorch训练业务迁移到昇腾设备上运行并获得较好的模型训练效果。华为云ModelArts针对该场景提供了系统化的迁移指导，包括迁移原理、迁移流程以及迁移后的精度调试及性能调优方法介绍。此外，ModelArts提供了即开即用的云上集成开发环境，包含迁移所需要的算力资源、AI框架、昇腾开发套件以及迁移调优工具链，最大程度减少客户自行配置环境的复杂度。

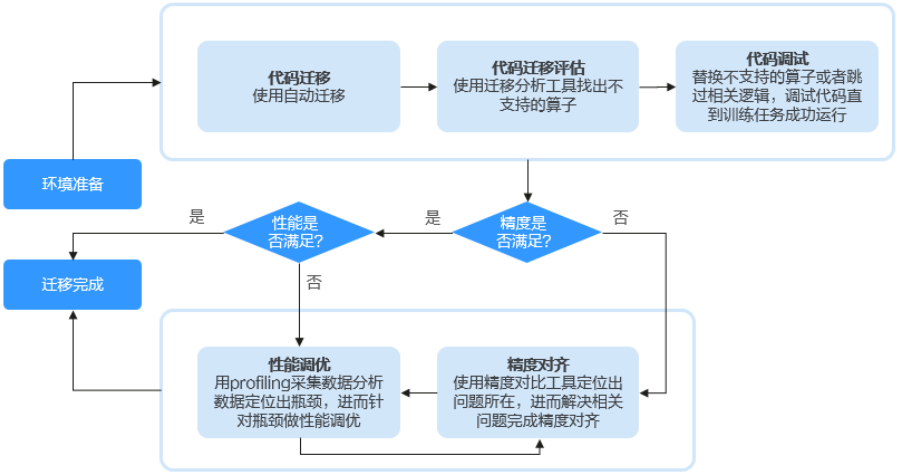
范围

本文涉及PyTorch训练的单卡和分布式业务迁移到昇腾的业务范围。当前针对常见的开源LLM/AIGC等领域的开源模型，ModelArts已经提供了迁移好的开箱即用模型，且保证了较优的精度和性能。如果用户业务同样使用这些开源模型，建议直接使用ModelArts提供的**模型运行指导**，其余场景再考虑使用本指导自行迁移和调优。

迁移流程

模型迁移主要指将开源社区中实现过的模型或客户自研模型迁移到昇腾AI处理器上，需要保证模型已经在CPU/GPU上运行成功。迁移到昇腾AI处理器的主要流程如下图所示。

图 6-32 迁移流程



6.3.2 训练迁移快速入门案例

本篇指导是迁移的总体思路介绍，便于用户对迁移过程有一个整体的认识。如果您希望通过具体案例直接实操，可参考

《**主流开源大模型基于DevServer适配PyTorch NPU训练指导**》，该案例以ChatGLM-6B为例，介绍如何将模型迁移至昇腾设备上训练、模型精度对齐以及性能调优。

6.3.3 迁移环境准备

本文以弹性裸金属作为开发环境，弹性裸金属支持深度自定义环境安装，可以方便的替换驱动、固件和上层开发包，具有root权限，结合配置指导、初始化工具及容器镜像可以快速搭建昇腾开发环境。

开通裸金属服务器资源请见**DevServer资源开通**，在裸金属服务器上搭建迁移环境请见**裸金属服务器环境配置指导**，使用ModelArts提供的基础容器镜像请见**容器环境搭建**。

6.3.4 训练代码迁移

前提条件

- 要迁移的训练任务代码在GPU上多次训练稳定可收敛。训练业务代码和数据，应该确保在GPU环境中能够运行，并且训练任务有稳定的收敛效果。
- 本文只针对基于PyTorch的训练代码迁移。这里假设用户使用的是基于PyTorch的训练代码进行迁移。其他的AI引擎如TensorFlow、Caffe等不在本指导的讨论范围中。
- 已经完成环境准备（参考**迁移环境准备**），并且代码、预训练模型、数据等训练必需内容已经上传到环境中。

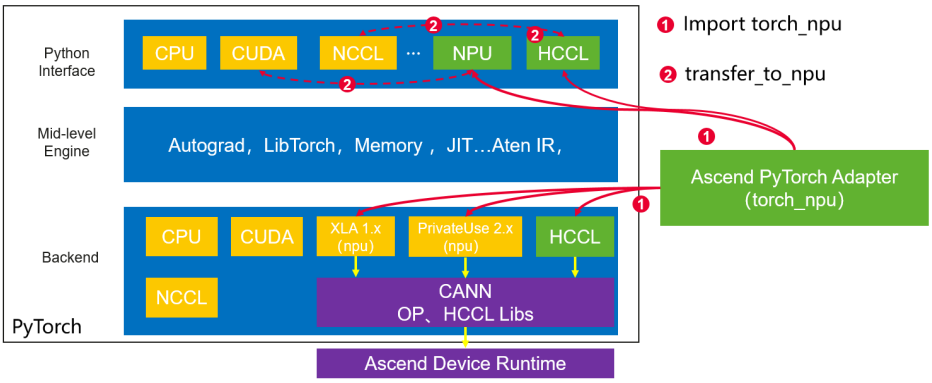
约束和限制

- 安装插件后，大部分能力能够对标在GPU上的使用，但并不是所有行为和GPU上是一一对应的，例如在torch_npu下，当PyTorch版本低于2.1.0时，一个进程只能操作一张昇腾卡，不支持一个进程操作多卡的能力，在PyTorch2.1.0及以上版本中torch_npu才支持一个进程中使用多张昇腾卡。
- 基于PyTorch上的第三方开发库非常多，例如transformers、accelerate、deepspeed以及Megatron-LM等，这些三方库昇腾也做了类似PyTorch Adapter的适配插件库，可以在Gitee的昇腾**官方仓库**中找到，请按需进行使用。部分三方库例如最新版本deepspeed已原生支持NPU，可以直接在昇腾设备上运行。

代码迁移基础知识

- PyTorch 2.1以下版本时，PyTorch官方并不直接支持昇腾的后端，仅直接支持CUDA和AMD ROCm，因此PyTorch在GPU上的训练代码无法直接在昇腾设备运行。PyTorch2.1版本提供了新硬件适配的插件机制，通过昇腾提供的Ascend Extension for PyTorch 插件，NPU可以成为PyTorch支持的硬件直接使用。
- **Ascend Extension for PyTorch** 作为一个PyTorch插件，支持在不改变PyTorch表达层的基础上，动态添加昇腾后端适配，包含增加了NPU设备、hccl等一系列能力的支持。安装后可以直接使用PyTorch的表达层来运行在NPU设备上。
- 当前提供了自动迁移工具进行GPU到昇腾适配，原理是通过**monkey-patch**的方式将torch下的CUDA、nccl等操作映射为NPU和hccl对应的操作。如果没有用到GPU的高阶能力，例如自定义算子、直接操作GPU显存等操作，简单场景下可以直接使用自动迁移。

图 6-33 torch_npu 工作原理示意图



- NPU（Neural Network Processing Unit）和GPU在构造结构上存在差异，因此迁移过程并不是完全平替的关系。昇腾训练芯片属于NPU的范畴，虽然在表达层可以通过torch.cuda和torch.npu的形式来替代，但是真实的算子下发、显存管理、集合通信等存在差异，用户需要了解NPU的运行机制才能更好的使用NPU设备，同时在遇到问题时快速找到原因。

代码迁移操作步骤

步骤1 在训练任务启动的Python脚本入口初始化Ascend Extension for PyTorch（torch_npu）。

在torch_npu安装后，该部分并没有直接植入到PyTorch中生效，需要用户显式调用。

```
#torch npu初始化
import torch_npu
```

调用后，前端会通过monkey-patch的方式注入到torch对象中，后端会注册NPU设备以及HCCL的参数面通信能力，这样就可以运行torch.npu相关接口。

图 6-34 torch_npu 导入

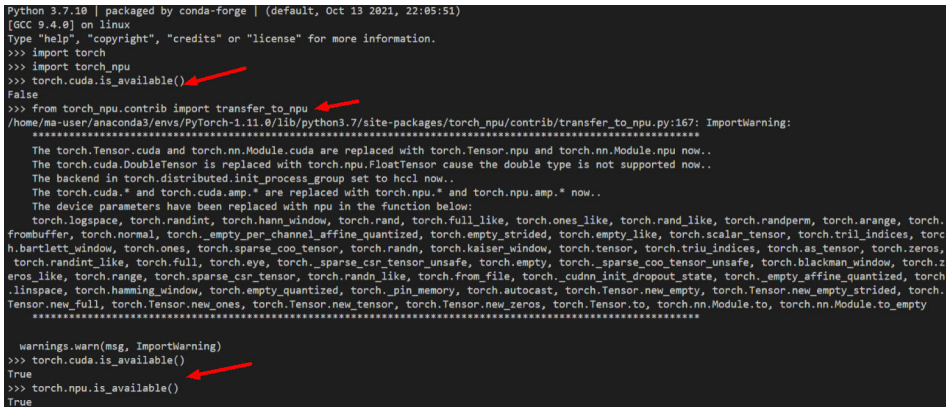
步骤2 自动迁移完成GPU代码到昇腾的快速适配。

torch_npu初始化后，原则上需要用户将原来代码中CUDA相关的内容迁移到NPU相关的接口上，包含算子API、显存操作、数据集操作、分布式训练的参数面通信nccl等，手动操作修改点较多且较为分散，因此昇腾提供了自动迁移工具transfer_to_npu帮助用户快速迁移。

自动迁移的原理是：通过注入的方式将当前Python运行环境中，运行时的torch.cuda等需要适配的接口和操作都映射成为torch.npu对应的接口。所以理论上常见场景下的代码不需要额外手工适配就可以运行到昇腾设备上了。

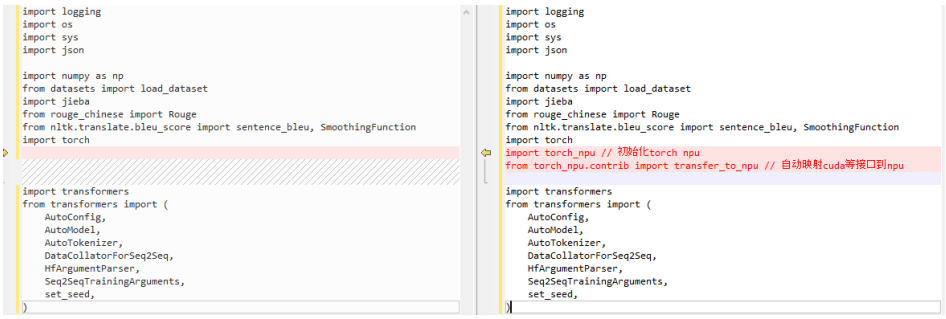
```
#自动映射cuda API到npu的代码
from torch_npu.contrib import transfer_to_npu
```

图 6-35 自动迁移后 cuda 映射为 npu 相关的 API



以**chatGLM-6b**为示例，在使用自动迁移时，在开发环境中克隆对应的代码，假设数据和预训练权重已经配置好，可以直接在ptuning目录下，训练入口代码main.py中添加两行代码来完成昇腾适配，注意添加位置为导入torch之后。启动训练脚本可以观察运行效果。

图 6-36 chatGLM-6b pTuning 训练入口导入自动迁移工具



说明

自动迁移适合没有使用CUDA高阶能力的简单场景，如果涉及自定义算子、主动申请GPU显存等操作，则需要额外进行手动迁移适配。

步骤3 手动迁移解决报错问题。

在完成代码自动迁移后，如果训练代码运行时还出现错误，则代表需要手动迁移适配。针对代码报错处，需要用户分析定位后将自动迁移未能迁移的GPU相关的代码调用修改为NPU对应的接口，可参考昇腾**手工迁移**文档进行操作。

----结束

常见问题

1. 如何检测当前的torch_npu是否正确安装?

   可以用如下的python命令在对应的运行环境中初步校验torch_npu是否正常安装。

   ```
   python3 -c "import torch;import torch_npu;print(torch_npu.npu.is_available())"
   ```

2. torch_npu使用报错看不懂怎么办？应该怎么求助?

   如果报错可以首先在**昇腾社区论坛**以及**Gitee的PyTorch**issues中查看是否有类似的问题找到一些线索。如果还无法解决可以通过提交工单的形式从华为云ModelArts入口来进行咨询以及求助对应的专业服务。

3. 自动迁移似乎还是要改很多脚本才能运行起来?

   因为自动迁移其实是对于torch运行环境中常用的GPU上的接口进行和昇腾设备的映射，原有的训练任务代码逻辑中例如数据集导入、预训练权重、GPU自定义算子的内容，以及对应的环境的超参数等内容都需要在实际的昇腾环境中进行调整。

6.3.5 PyTorch 迁移精度调优

完成代码迁移适配后，用户需要进一步验证训练精度是否达标。在保证迁移正确的前提下，迁移后精度偏差的来源，一方面是昇腾设备部分算子的实现和CUDA算子有差异，另外一方面则是硬件方面的差异，如Ascend Snt9芯片上的Matmul和Conv等cube算子只支持FP16，可能会导致数值溢出，从而引起精度误差。此外，网络随机参数初始化差异以及典型场景（比如dropout和数据集shuffle等操作）都可能引入误差，所以迁移模型精度校验以及精度调优的工作至关重要。

精度校验

迁移之后的精度校验工作是以CPU/GPU环境训练过程作为标杆的，这里的前提是在迁移前，模型已经在CPU/GPU环境达到预期训练结果。在此基础上，迁移过程的精度问题一般包括：

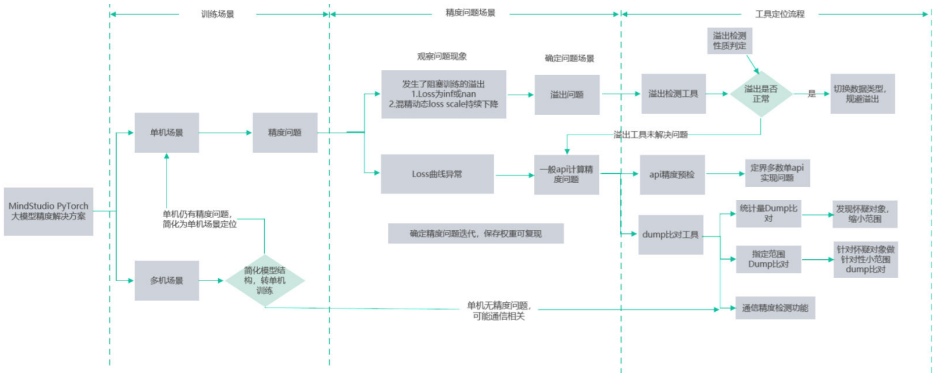
- loss曲线与CPU/GPU差异不符合预期。
- 验证准确度与CPU/GPU差异不符合预期。

在迁移到NPU环境下训练发现以上问题时，说明精度可能存在偏差，需要进一步做精度调优。下文将分别阐述精度诊断的整体思路和借助工具如何进行精度问题的定位。

精度调优总体思路

精度问题定位首先要能在昇腾环境上稳定地复现问题，大模型训练通常使用多机训练，而多机训练复现问题的成本通常较高，且直接使用工具可能会产生TB级的大量dump数据，存储和拷贝都比较困难，所以建议用户在复现前先进行模型裁剪，例如减小模型层数(通过num_layers参数控制)、将模型转为单机训练等，这样会大大降低后续定位的难度。在问题复现后，可以进一步根据问题现象选择对应的工具辅助定位，包括溢出检测工具、API预检工具、整网dump比对工具等，通过多组试验比较标杆（GPU/CPU）环境和昇腾环境上运行训练时的差异点来判断问题所在、整体流程如下图所示，更多介绍可参考**昇腾精度调试指南**。

图 6-37 精度调优流程



溢出检测和dump比对是通过在PyTorch模型中注入hook从而dump模型训练过程的输入输出数据，比对NPU环境和标杆环境的所有输入输出的差异来发现异常信息。更多介绍可参考精度比对工具**ptdbg-ascend**。

API精度预检是通过提取模型中所有的API前反向信息，通过工具构造相应的API单元测试，将NPU输出与标杆比对，从而检测出精度有差异的API。更多介绍可参考精度预检工具**api_accuracy_checker**。

准备工作

在定位精度问题之前，首先要排除其他因素的干扰。目前大部分精度无法对齐的问题都是由于模型超参数、Python三方库版本、模型源码等与标杆环境（GPU/CPU）设置的不一致导致的，为了在定位过程中少走弯路，需要在定位前先对训练环境及代码做有效排查。此外，问题定位主要基于GPU环境和NPU环境上运行的过程数据做对比，所以需要分别准备GPU和NPU训练环境，大部分场景需要规模相同的训练环境，如果已经将模型缩减到单机可运行，则只是单台GPU设备即可。

定位前的排查当前主要包含如下几个方面：

1. 训练超参数。常见的超参如下图所示：

图 6-38 训练超参数

```
python -m torch.distributed.launch $DISTRIBUTED_ARGS \
       pretrain_llama.py \
       --DDP-impl local \
       --tensor-model-parallel-size 8 \
       --pipeline-model-parallel-size 1 \
       --sequence-parallel \
       --num-layers 32 \
       --hidden-size 4096 \
       --ffn-hidden-size 11008 \
       --num-attention-heads 32 \
       --attention-dropout 0.0 \
       --hidden-dropout 0.0 \
       --init-method-std 0.01 \
       --micro-batch-size 4 \
       --global-batch-size 16 \
       --seq-length 4096 \
       --max-position-embeddings 4096 \
       --data-path $DATA_PATH \
       --tokenizer-name-or-path $TOKENIZER_PATH \
       --tokenizer-not-use-fast \
       --split 100,0,0 \
       --distributed-backend nccl \
       --lr 1.25e-5 \
       --min-lr 1.25e-6 \
       --lr-decay-style cosine \
       --weight-decay 1e-1 \
       --clip-grad 1.0 \
       --initial-loss-scale 65536.0 \
       --adam-beta1 0.9 \
       --adam-beta2 0.95 \
       --log-interval 1 \
       --load ${LOAD_CHECKPOINT_PATH} \
       --save ${SAVE_CHECKPOINT_PATH} \
       --save-interval 10000 \
       --eval-interval 10000 \
       --eval-iters 0 \
       --use-fused-rotary-pos-emb \
       --no-query-key-layer-scaling \
       --attention-softmax-in-fp32 \
       --no-masked-softmax-fusion \
       --train-iters 50000 \
       --lr-warmup-fraction 0.01 \
       --no-load-optim \
       --no-load-rng \
       --mlp-layer-fusion \
       --use-flash-attn \
       --print-input-ids \
       --bf16 | tee ./logs/ascendspeed_npu_7b-bf16_tp8pp1mbs4_16_fa_sp_cann1107cmc3${logfile}.log
```

模型的超参通常可能调整的主要有学习率，batch size，并行切分策略，学习率warm-up，模型参数，FA配置等，**用户在进行NPU精度和GPU精度比对前，需要保证两边的配置一致。**

a. 学习率：lr

b. batch size, micro batch size

batch size会影响训练速度，有时候也会影响模型精度。micro batch size会影响流水线并行中设备的计算效率。

c. 切分策略：DP、TP、PP

DP：data parallel

数据并行（data parallelism）是大规模深度学习训练中常用的并行模式，它会在每个进程(设备)或模型并行组中维护完整的模型和参数，但在每个进程上或模型并行组中处理不同的数据。因此，数据并行非常适合大数据量的训练任务。

TP：tensor parallel

张量并行也叫层内并行，通过将网络中的权重切分到不同的设备，从而降低单个设备的显存消耗，使得超大规模模型训练成为可能。张量并行不会增加设备等待时间，除了通信代价外，没有额外代价。

PP：pipeline parallel

流水线并行将模型的不同层放置到不同的计算设备，降低单个计算设备的显存消耗，从而实现超大规模模型训练。流水线并行也叫层间并行，层输入输出的依赖性使得设备需要等待前一步的输出，通过batch进一步切分成微batch， 网络层在多个设备上的特殊安排和巧妙的前向后向计算调度，可以最大程度减小设备等待（计算空泡），从而提高训练效率。

d. 学习率预热

不同的学习率调度器(决定什么阶段用多大的学习率)有不同的学习率调度相关超参，例如线性调度可以选择从一个初始学习率lr-warmup-init开始预热。可以选择多少比例的训练迭代步使用预热阶段的学习率。不同的训练框架有不同的参数命名，需要结合代码实现设置对应的参数。

e. 模型结构

配置模型结构的超参主要有num-layer、hidden-size;、seq-length等。

f. FA配置：use-flash-attn。

2. 训练脚本

   由算法迁移人员排查迁移后的NPU脚本是否存在问题，可以通过beyond compare工具比对GPU训练脚本和NPU训练脚本之间是否存在差异。例如是否GPU环境下开启了FA但是NPU上未开启FA。

3. 三方库版本比对

   大模型训练通常会使用deepspeed、megatron等三方库，需要确保这些三方库的版本一致。

4. 环境版本更新

   这一项仅在条件允许的情况下进行，根据精度问题定位经验，部分问题是由于使用了较早版本的昇腾软件版本或者非商用发布的昇腾软件版本，所以推荐在条件允许的前提下配套安装最新商发版本的昇腾开发套件CANN Toolkit、昇腾驱动以及torch_npu包。参考**昇腾商用版资源下载指导**。

5. 数据集。

   需要排查是否使用的训练数据集存在差异。

6. 初始权重。

   需要排查是否加载的初始权重有差异，建议加载相同的初始权重。

问题复现

一般场景的训练模型都是包括随机种子、数据集shuffle、网络结构dropout等操作的，目的是在网络阶段引入一定的随机性使得训练结果更加具有鲁棒性。然而在精度诊断或者对齐阶段，这些随机性会导致训练运行结果每次表现不一致，无法进行和标杆的比对。因此在训练模型复现问题时，需要固定存在随机性的步骤，保证实验可重复性。存在随机性的步骤包括模型参数初始化，数据batch加载顺序，dropout层等。部分算子的计算结果也存在不确定性，需要固定。

当前固定随机性操作可分为**工具固定**和**人工固定**两种。

1. 工具固定（seed_all）

   对于网络中随机性的固定，**ptdbg**工具提供了seed_all接口用于固定网络中的随机数。如果客户使用了工具但取用了其他随机种子，则必须使用客户的随机种子固定随机性。

   函数原型

   seed_all(seed=1234, mode=False)

表 6-2 参数说明

参数名	说明	是否必选
l seed	l 随机数种子。参数示例：seed=1000。默认值为：1234。	l 否
l mode	l 确定性计算模式。可配置True或False。参数示例：mode=True。默认为False。 l 即使在相同的硬件和输入下，API多次执行的结果也可能不同，开启确定性计算是为了保证在相同的硬件和输入下，API多次执行的结果相同。 l 确定性计算会导致API执行性能降低，通常不需要在精度问题刚开始定位时就开启，而是建议在发现模型多次执行结果不同的情况下时再开启。 l rnn类算子、ReduceSum、ReduceMean等算子可能与确定性计算存在冲突，若开启确定性计算后多次执行的结果不相同，则考虑存在这些算子。	l 否

函数示例

seed_all函数的随机数种子，取默认值即可，无须配置；第二个参数默认关闭，不开启确定性计算时也无须配置。

确定性计算是NPU的一套机制，用于保证算子的计算确定性。之所以要有这个机制，是为了在debug过程中，让所有的算子计算结果前后完全一致可复现，这是大多数精度问题分析的重要前提。因此，在精度问题定位过程中，确定性计算**不是目的，而是手段**，很多场景下我们要在确定性计算使能的情况下，进行下一步的精度问题分析定位。cuda对部分算子实现了确定性计算，但仍有部分算子无法固定。通常需要依赖确定性计算的场景是长稳问题，因为长稳问题我们需要通过多次长跑来分析Loss情况，这时候如果NPU本身计算结果不确定，就难以支撑和GPU结果的多次对比。

l 示例1：仅固定随机数，不开启确定性计算

seed_all()

l 示例2：固定随机数，开启确定性计算

seed_all(mode=True)

除此以外，还需要添加以下环境变量，固定通信算子计算的确定性：

export HCCL_DETERMINISTIC=TRUE

固定随机数范围

seed_all函数可固定随机数的范围如下表。

API	固定随机数
l os.environ['PYTHONHASHSEED'] = str(seed)	l 禁止Python中的hash随机化
l random.seed(seed)	l 设置random随机生成器的种子
l np.random.seed(seed)	l 设置numpy中随机生成器的种子

API	固定随机数
l torch.manual_seed(seed)	l 设置当前CPU的随机种子
l torch.cuda.manual_seed(seed)	l 设置当前GPU的随机种子
l torch.cuda.manual_seed_all(seed)	l 设置所有GPU的随机种子
l torch_npu.npu.manual_seed(seed)	l 设置当前NPU的随机种子
l torch_npu.npu.manual_seed_all(seed)	l 设置所有NPU的随机种子
l torch.backends.cudnn.enable=False	l 关闭cuDNN
l torch.backends.cudnn.benchmark=False	l cuDNN确定性地选择算法
l torch.backends.cudnn.deterministic=True	l cuDNN仅使用确定性的卷积算法

2. 工具固定（dropout）

   dropout的实质是以一定概率使得输入网络的数据某些维度上变为0，这样可以使得模型训练更加有效。但在精度问题的定位过程之中，我们需要避免产生这种问题，因此需要关闭drop_out。

   在使用from ptdbg_ascend import *后，工具会自动将如下接口参数p（丢弃概率）置为0。

   torch.nn.functional.dropout

   torch.nn.functional.dropout2d

   torch.nn.functional.dropout3d

   torch.nn.Dropout

   torch.nn.Dropout2d

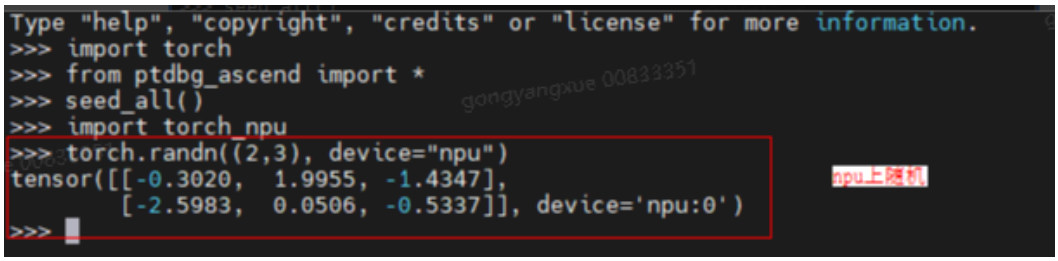
   torch.nn.Dropout3d

3. 人工固定（硬件随机差异）

工具内部对于随机的控制，是通过设定统一的随机种子进行随机性固定的。

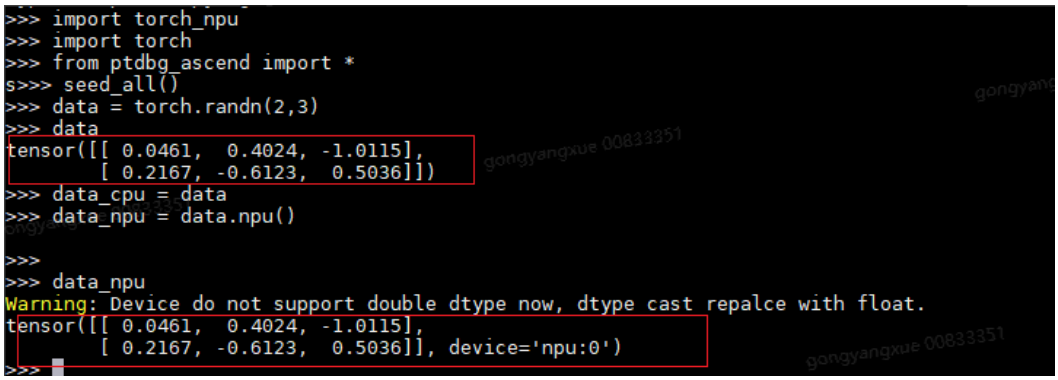
但是由于硬件的差异，会导致同样的随机种子在不同硬件上生成的随机数不同。具体可以看下面示例：

```
>>> import torch
>>> from ptdbg_ascend import *
2023-08-31 10:50:45(1025)-[INFO]For precision comparison, the probability p in the dropout method is set to 0.
>>> seed_all()
>>> torch.randn((2,3),device="cuda")
tensor([[-1.6165,  0.5685, -0.5102],
        [-0.9113, -1.1555, -0.2262]], device='cuda:0')
```



图中可见，torch.randn在GPU和NPU上固定随机种子后，仍然生成不同的随机张量。

对于上述场景，用户需要将网络中的randn在cpu上完成后再转到对应device。比如StableDiffusion中需要在forward过程中逐步生成随机噪声。



这样在host侧生成的随机张量能够保证一样，搬移到npu或者gpu设备上仍然一样。

固定随机性完成后，我们可以使用缩小的模型在单机环境进行问题复现。复现后使用下一章节介绍的工具进行问题定位。这里需要注意的是，部分模型算法本身存在固有的随机性，在使用上述方法固定随机性后，如果使用工具也未能找到出问题的API，需要分析是否由算法本身的随机性导致。

API 预检工具使用说明

对于任何问题场景都推荐先使用预检工具，检查第1个step或loss明显出现问题的step。它可以抓取模型中API输入的数值范围，根据范围随机生成输入，用相同的输入分别在npu（gpu）和cpu上执行算子，比较输出差异。预检最大的好处是，它能根据算子（API）的精度标准来比较输出结果并判定其是否有精度问题，所以不需要使用者做任何额外分析，而且基本不会出现误检的情况，使用门槛较低。预检工具使用包含以下三步：dump、run_ut以及api_precision_compare。

1）dump这一步主要是为了获取整网中每个pytorch 计算API的输入真实张量数值、shape、 dtype以及数值分布。

2）run_ut这一步可以根据dump输出数据完成NPU vs CPU高精度（标杆）或者GPU vs CPU高精度（标杆）的单API测试，并输出预检结果。

3）api_precision_compare是预检结果的比对，需要同时获取NPU和GPU环境下run_ut的结果文件进行比对，输出最终的比对结果。

该工具的使用指导请参考**api_accuracy_checker**。

精度比对工具使用说明

ptdbg_ascend是昇腾开源的用于PyTorch框架迁移训练的精度对比工具。使用时需要两组模型运行环境，一组是基于昇腾AI芯片的NPU环境，另一组是CPU/GPU环境（标杆

环境）。ptdbg_ascend通过在PyTorch训练脚本中插入dump接口，跟踪计算图中算子的前向传播与反向传播时的输入与输出，然后再compare将对比结果写出到.csv表格中。

当前支持计算Cosine（余弦相似度）、MaxAbsErr（最大绝对误差）和MaxRelativeErr（最大相对误差）这三种评价指标，通过设定相似度阈值和最大绝对偏差限来判断API运行时是否存在精度问题。

步骤1 安装ptdbg_ascend工具。

下载最新的whl包至服务器并拷贝至运行的容器环境中（**下载链接**），通过pip安装ptdbg_ascend工具。

```
#shell
pip install ./ptdbg_ascend-{version}-py3-none-any.whl
```

步骤2 获取NPU和GPU的dump数据。

在单卡场景下，PyTorch训练脚本插入dump接口方式如下：

```
# 导入ptdbg_ascend依赖包
from ptdbg_ascend import register_hook, overflow_check, seed_all, set_dump_path, set_dump_switch, acc_cmp_dump
# 在 main 函数中固定随机数
seed_all(seed=1234, mode=False)
# 设置 dump 文件保存路径
set_dump_path("./npu_dump", dump_tag='all')
# 添加hook函数和数据比对dump开关
register_hook(model, acc_cmp_dump)
# dump 开启和关闭。在一个 iter 的开始和结束位置设置
set_dump_switch("ON", mode="api_stack", filter_switch="OFF")
# ---------
# iterartion
# ---------
set_dump_switch("OFF", mode="api_stack", filter_switch="OFF")
```

以上给出的是dump整网精度数据的方式，在迁移过程中也会遇到数值溢出问题。在ptdbg中设置检查精度溢出方式如下:

```
# dump 开启和关闭。在一个 iter 的开始和结束位置设置
register_hook(model, overflow_check, overflow_nums=3)
set_overflow_check_switch("ON")
# ---------
# iterartion
# ---------
set_overflow_check_switch("OFF")
```

这里的overflow_sum用来控制溢出次数，表示多少次溢出时停止训练。

步骤3 生成精度对比表。

dump得到NPU和GPU数据的之后，会得到保存api完整输入输出Tensor的“.npy”文件以及保存API简单统计信息的“.pkl”文件。在compare对比时，需要分别指定NPU和GPU的pkl文件路径和dump数据路径（具体参数请按实际路径填写），compare.py实现如下：

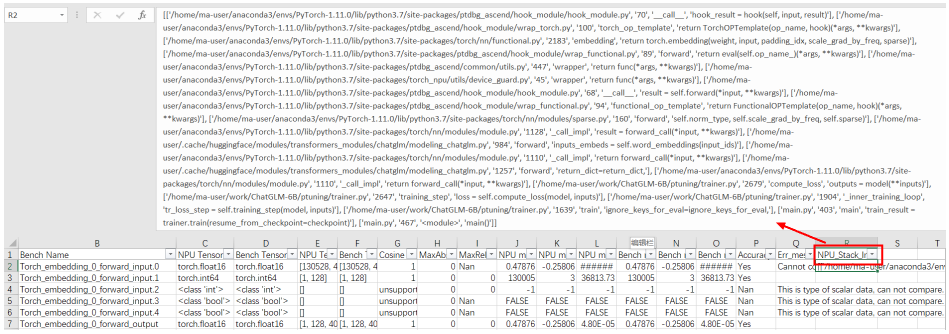
```
# compare.py
from ptdbg_ascend import compare
dump_result_param= {
"npu_pkl_path": "${dump_data_npu}/api_stack_dump.pkl",
"bench_pkl_path":"${dump_data_gpu}/api_stack_dump.pkl",
"npu_dump_data_dir": "${dump_data_npu}/api_stack_dump/",
"bench_dump_data_dir": "${dump_data_npu}/api_stack_dump/",
"is_print_compare_log": True
}
compare(dump_result_param, "./output", stack_mode=True)
```

执行comapre.py对比之后会在output目录下输出compare_result_{timestamp}.csv文件。根据此文件，可以查看网络训练过程中各个API执行的输入输出以及评价指标的信息。

步骤4 根据堆栈信息定位代码差异点。

在compare对比表文件NPU_Stack_Info列，有各个算子在执行时的堆栈信息。如下图所示，列出的是NPU下Torch_embedding_0_forward（代表torch的embedding算子在第 0 次执行forward阶段）的堆栈信息。

图 6-39 堆栈信息



部分情况下也需要查看GPU训练时的堆栈信息来排除GPU和NPU是否执行的不同逻辑代码。这种情况可以根据NPU Name在dump阶段生成的.pkl文件中查找。在ptdbg-ascend中，有支持使用API接口parse堆栈信息的方式，代码如下：

```
# compare.py
from ptdbg_ascend import parse
parse(". /dump_data /npu/rank0/api_stack_dump.pkl", "Torch_embedding_0_forward")
```

步骤5 精度对齐。

ptdbg-ascend当前支持的评判指标有Cosine、MaxAbsError（可参见**接口函数说明**）：

精度存在异常的情况包含以下三种情况：

- Cosine < 0.99且MaxAbsError > 0.001
- Cosine < 0.9
- MaxAbsError > 1

其余情况都视为达标。精度对齐时，需要根据compare表格查找精度不达标的算子进行调整优化。由于算子间可能存在前后数据传输的相关性，一般先定位第一个不达标的算子，然后结合堆栈信息进行分析和调整，调整之后重新训练dump数据再做对比，直至模型训练的loss曲线和在验证集上做测试的结果和GPU标杆结果一致为止。

----结束

6.3.6 PyTorch 迁移性能调优

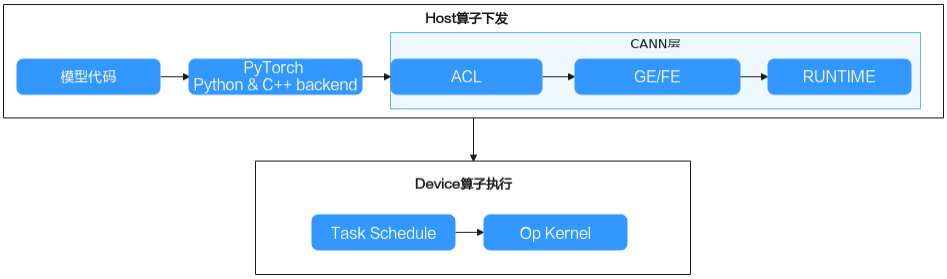
6.3.6.1 性能调优总体原则和思路

PyTorch在昇腾AI处理器的加速实现方式是以算子为粒度进行调用（OP-based），即通过Python与C++调用CANN层接口Ascend Computing Language（AscendCL）调用

一个或几个亲和算子组合的形式，代替原有GPU的实现方式，具体逻辑模型参考**此处**。

在PyTorch模型迁移后进行训练的过程中，CPU只负责算子的下发，而NPU负责算子的执行，算子下发和执行异步发生，性能瓶颈在此过程中体现。在PyTorch的动态图机制下，算子被CPU逐个下发到NPU上执行。一方面，理想情况下CPU侧算子下发会明显比NPU侧算子执行更快，此时性能瓶颈主要集中在NPU侧；另一方面，理想情况下NPU侧算子计算流水线一直执行，不会出现NPU等待CPU算子下发即NPU空转的场景，如果存在，则CPU侧算子下发存在瓶颈。

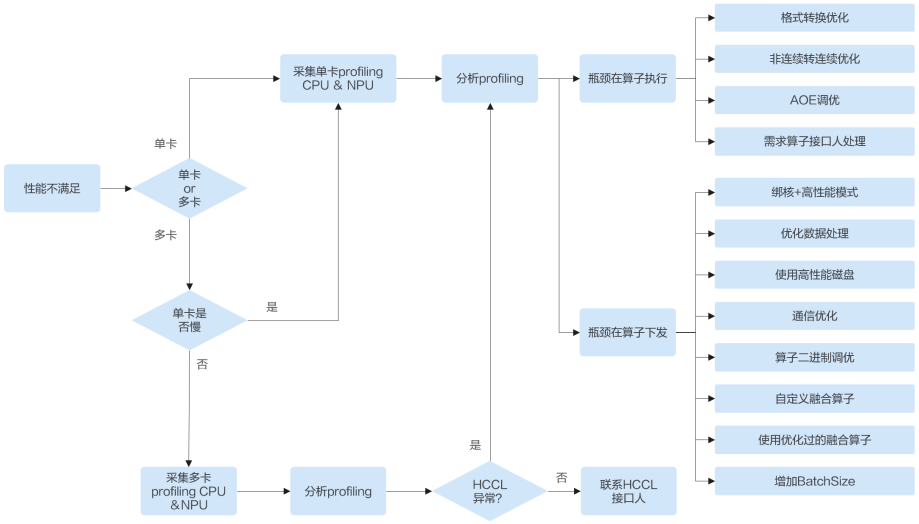
图 6-40 Host 算子下发和 Device 算子执行



综上所述，性能优化的总体原则为：减少Host算子下发时间、减少Device算子执行时间。

训练代码迁移完成后，如存在性能不达标的问题，可参考下图所示流程进行优化。建议按照单卡、单机多卡、多机多卡的流程逐步做性能调优。

图 6-41 性能调优总体思路



为了便于用户快速进行迁移调优，降低调优门槛，ModelArts提供了MA-Adivisor性能自动诊断工具，用户采集性能profiling数据后，可通过该工具自动扫描profiling数据，工具分析完数据后会给出可能的性能问题点及调优建议，用户可以根据调优建议做相应的修改适配。目前该工具对CV类模型给出的调优建议较多，LLM类建议稍少，但是总体都有性能提升，实测大约可提升10%~30%的性能，并且已经在多个迁移性能调优项目中实际应用。

6.3.6.2 自动诊断工具 MA-Advisor 使用指导

6.3.6.2.1 自动诊断工具 MA-Advisor 简介

MA-Advisor是一款昇腾迁移性能问题自动诊断工具，当前支持如下场景的自动诊断：

- 推理场景下的子图数据调优分析，给出对应融合算子的调优建议。
- 推理、训练场景下对Profiling timeline单卡数据进行调优分析，给出相关亲和API替换的调优建议。
- 推理、训练场景下对Profiling单卡数据进行调优分析，给出AICPU相关调优建议。
- 推理、训练场景下对Profiling单卡数据进行调优分析，给出block dim、operator no bound相关AOE配置以及调优建议。
- 支持对昇腾训练、推理环境进行预检，完成相关依赖配置项的提前检查，并在检测出问题时给出相关修复建议。

自动诊断工具可以有效减少人工分析profiling的耗时，降低性能调优的门槛，帮助客户快速识别性能瓶颈点并完成性能优化。推荐用户在采集profiling分析后使用自动诊断工具进行初步性能调优。更进一步的性能调优再使用Ascend-Insight工具进行数据可视化并人工分析瓶颈点。

6.3.6.2.2 MA-Advisor 使用指导

工具安装

步骤1 下载**ma-advisor**安装包至开发环境中。

步骤2 （可选）完成软件包签名校验。

a. **下载软件包签名校验文件**。

b. 安装openssl并进行软件一致性验证，具体签名校验命令如下：

```
openssl cms -verify -binary -in ma_advisor-latest-py3-none-any.whl.cms -inform DER -content ma_advisor-latest-py3-none-any.whl  -noverify > ./test
```

签名校验结果如下所示则完成软件的一致性验证。

图 6-42 一致性验证

```
(PyTorch-1.8) [ma-user work]$openssl cms -verify -binary -in ma_advisor-latest-py3-none-any.whl.cms -inform DER -content ma_advisor-latest-py3-none-any.whl  -noverify > ./test
CMS Verification successful
```

步骤3 执行安装命令。

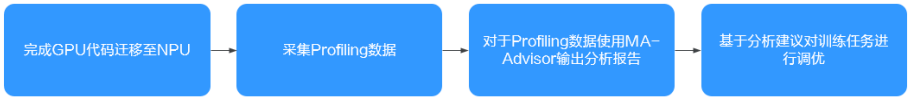
```
pip install ma_advisor-latest-py3-none-any.whl
```

----结束

工具使用流程

代码迁移性能调优流程主要如下：

图 6-43 调优流程



步骤1 基于Pytorch Adapter完成GPU代码迁移至NPU。

步骤2 参考《Ascend PyTorch Profiler数据采集与分析》采集训练的Profiling数据，采集profiling时需要保持参数with_stack=True，用于定位python侧的代码问题。

图 6-44 采集训练的 Profiling 数据

```
# 扩展参数，通过扩展配置性能分析工具常用的采集项
experimental_config = torch_npu.profiler._ExperimentalConfig(
    aic_metrics=torch_npu.profiler.AiCMetrics.PipeUtilization, profiler_level=torch_npu.profiler.ProfilerLevel.Level1, l2_
)

# 配置性能数据采集参数
with torch_npu.profiler.profile(
    activities=[
      torch_npu.profiler.ProfilerActivity.CPU,  # 采集框架侧数据开关
      torch_npu.profiler.ProfilerActivity.NPU   # 采集NPU数据开关
      ],
    schedule=torch_npu.profiler.schedule(wait=1, warmup=1, active=2, repeat=2, skip_first=10),  # 设置不同step的行
    on_trace_ready=torch_npu.profiler.tensorboard_trace_handler("./result"),  # 将采集到的性能数据导出为TensorBo

    record_shapes=True,  # 算子的InputShapes和InputTypes，Bool类型
    profile_memory=True,  # 算子的内存占用情况，Bool类型
    with_stack=True,  # 算子调用栈，Bool类型
    experimental_config=experimental_config) as prof:
      for step in range(steps):
        train_one_step(step, steps, train_loader, model, optimizer, criterion)
        prof.step()
```

步骤3 使用ma-advisor命令行工具对上述Profiling数据进行分析，会在当前工作目录下输出“ma_advisor_{timestamp}.html”和“log/ma_advisor_{timestamp}.xlsx”文件，如果识别到AOE相关调优项，会在当前工作目录下生成“operator_tuning_file.cfg”文件。

步骤4 优先根据“ma_advisor_{timestamp}.html”中的建议对训练任务进行调优，包括亲和API替换、算子调优（AOE调优、二进制调优、AI CPU分析）、异常dataloader检测等。

----结束

MA-Advisor 命令总览

MA-Advisor当前支持如下四种命令：

- analyze：根据Profiling单卡数据进行相关调优分析，并给出调优建议。
- query：根据Profiling单卡timeline数据，输入算子相关参数，查询出算子详细信息。
- env：对当前昇腾环境进行运行前预检查，分析出相关环境问题，并给出环境检查修复建议。
- update: 更新用于分析的知识库，将云端知识库同步至分析环境中。
- auto-completion：自动补全模式，在终端中自动完成MA-Advisor命令补全，支持“bash（默认）/zsh/fish”。

在执行任何命令前，若对其参数有疑问，可执行-h进行查看帮助，例如：

```
ma-advisor analyze -h
```

图 6-45 查看帮助

```
PS D:\> ma-advisor --help
Usage: ma-advisor [OPTIONS] COMMAND [ARGS]...

Options:
  -V, -v, --version  0.0.2
  -H, -h, --help     Show this message and exit.

Commands:
  analyze          Analyze profiling datasets and give performance optimization suggestion.
  query            Query operator details from timeline.
  env              Environment operation command, such as check and test.
  update           Update operation command, such as update rule and specify save path.
  auto-completion  Auto complete ma-advisor command in terminal, support "bash(default)/zsh/fish".
```

analyze 命令详解

- all：同时进行融合算子图调优、亲和API替换调优、AICPU调优、算子调优等分析，并输出相关简略建议到执行终端中，并生成“ma_advisor_**.html”文件可供用户在浏览器中进行预览：

  ```
  ma-advisor analyze all --data-dir='/temp/profiling_dir'
  ```

图 6-46 命令样例

```
+-----+----------------------+--------------------------------------+--------------------------------------+
| No. | Problem              | Description                          | Suggestion                           |
+-----+----------------------+--------------------------------------+--------------------------------------+
| 1   | Affinity training api | Found 7 apis to be replaced based on the | 1. Please replace training api according |
|     |                      | runtime env cann-7.0.0 and torch-1.11.0 | to sub table 'Affinity training api' |
+-----+----------------------+--------------------------------------+--------------------------------------+
| 2   | operator no bound    | There is no mte, cube, vector, scalar | 1. Optimize operator by AOE, such as: |
|     |                      | ratio is more than 80.00%; Top task  | 'aoe --job_type=2                    |
|     |                      | duration operators need to be tuned are | --model_path=$user_dump_path --tune_ops_ |
|     |                      | as follows:  Addcdiv, Addcmul, Axpy, | file=D:\operator_tuning_file_20240226100 |
|     |                      | Mul, ForeachNonFiniteCheckAndUnscale, | 841.cfg'                             |
|     |                      | Add,  Abs, RealDiv, Square,  Sqrt    |                                      |
+-----+----------------------+--------------------------------------+--------------------------------------+
| 3   | block dim            | some operator does not make full use of | 1. Optimize operator by AOE, such as: |
|     |                      | 20 ai core or 40 ai vector core;  Top-10 | 'aoe --job_type=2                    |
|     |                      | operator of task duration are as     | --model_path=$user_dump_path --tune_ops_ |
|     |                      | follows: BatchMatMulV2, MatMulV2,    | file=D:\operator_tuning_file_20240226100 |
|     |                      | ConcatD,  Mul, Sub, Slice,           | 841.cfg'                             |
|     |                      | LayerNormXBackpropV3, SoftmaxV2, Add, |                                      |
|     |                      | LayerNormV3                          |                                      |
+-----+----------------------+--------------------------------------+--------------------------------------+
| 4   | AICPU operator       | Some operators and task duration exceed | 1. Modify code to avoid aicpu operator |
|     |                      | 20 us, such as : UpsampleNearest3d,  |                                      |
|     |                      | UpsampleNearest3dGrad                |                                      |
+-----+----------------------+--------------------------------------+--------------------------------------+
```

命令执行后同时会生成各场景优化建议的html，相关算子问题概览会按照不同建议进行汇总。

图 6-47 生成结果

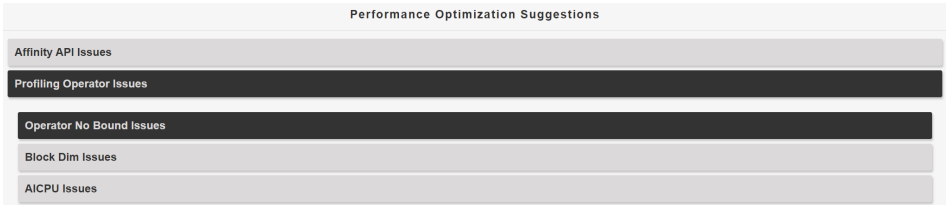
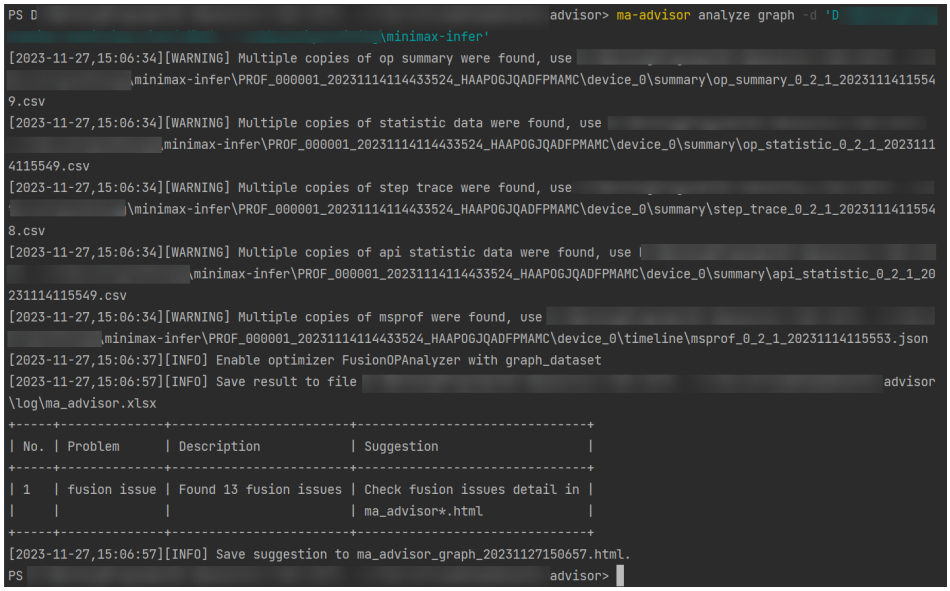


表 6-3 参数解释

参数	缩写	是否必填	说明
--data-dir	-d	必填	代表存储Profiling单卡性能数据的目录，目前暂不支持同时分析多卡Profiling目录，Profiling数据可通过如下方法获取： 在执行推理或训练程序时，请参见“**Profiling工具使用指南**”完成Profiling数据的采集、解析与导出（您可以在昇腾文档页面左上角切换版本，选择对应版本的指导文档）。数据采集时需要配置“aic-metrics”参数为“PipeUtilization”，“aicpu”参数为“on”。 MA-Advisor依赖Profiling工具解析后的timeline数据、summary数据以及info.json*文件，请确保指定的“profiling_dir”目录下存在以上文件。
--cann_version	-cv	选填	使用Profiling工具采集时对应的CANN软件版本，可通过在环境中执行如下命令获取其version字段，目前配套的兼容版本为“6.3.RC2”，“7.0.RC1”和“7.0.0”，此字段不填默认按“7.0.RC1”版本数据进行处理，其余版本采集的Profiling数据在分析时可能会导致不可知问题： `cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info`
--torch_version	-tv	选填	运行环境的torch版本，默认为1.11.0，支持torch1.11.0和torch2.1.0，当运行环境torch版本为其他版本如torch1.11.3时，可以忽略小版本号差异选择相近的torch版本如1.11.0。
--debug	-D	选填	工具执行报错时可打开此开关，将会展示详细保存堆栈信息。
--help	-h，-H	选填	在需要查询当前命令附属子命令或相关参数时，给出帮助建议。

- graph：单独对推理dump的子图数据进行调优，并在分析完成后，给出相关建议到终端中，并生成“ma_advisor_graph_**.html”文件到执行目录中，目前暂不支持同时分析多卡推理性能数据：
  ```
  ma-advisor analyze graph --data-dir='/temp/profiling_dir'
  ```

图 6-48 命令样例



命令执行后生成融合算子优化建议的html，相关融合算子问题概览会按照不同融合算子类型进行汇总。

图 6-49 生成结果

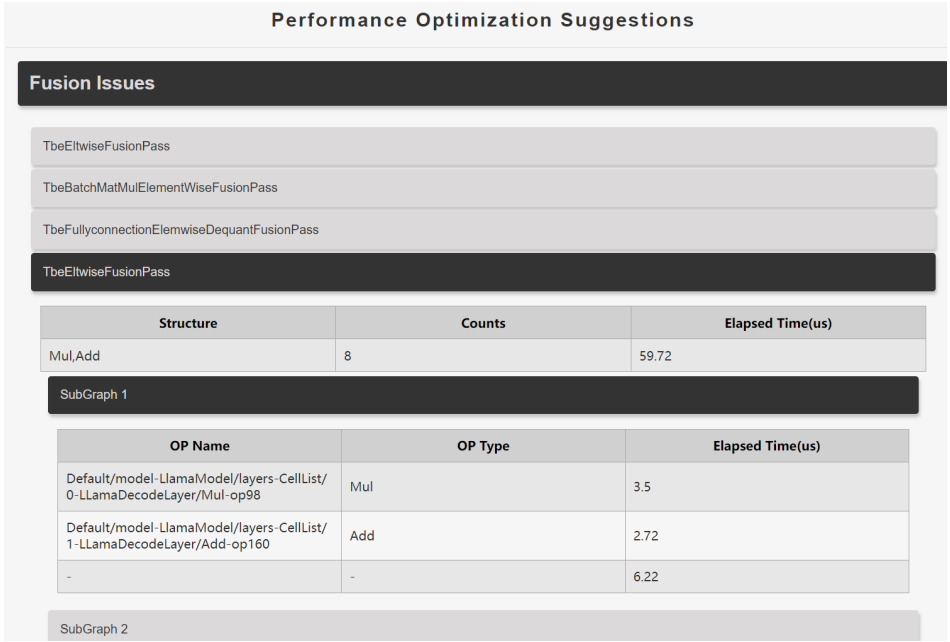


表 6-4 参数解释

参数	缩写	是否必填	说明
--data-dir	-d	必填	代表存储Profiling单卡性能数据的目录，目前暂不支持同时分析多卡Profiling目录，Profiling数据可通过如下方法获取： ● 在执行推理或训练程序时，请参见“**Profiling工具使用指南**”完成Profiling数据的采集、解析与导出（您可以在昇腾文档页面左上角切换版本，选择对应版本的指导文档）。数据采集时需要配置“aic-metrics”参数为“PipeUtilization”，“aicpu”参数为“on”，“with_stack”参数为True。 ● MA-Advisor 依赖Profiling工具解析后的timeline数据、summary数据以及info.json*文件，请确保指定的“profiling_dir”目录下存在以上文件。
--cann_version	-cv	选填	使用Profiling工具采集时对应的CANN软件版本，可通过在环境中执行如下命令获取其version字段，目前配套的兼容版本为“6.3.RC2”，“7.0.RC1”和“7.0.0”，此字段不填默认按“7.0.RC1”版本数据进行处理，其余版本采集的Profiling数据在分析时可能会导致不可知问题： `cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info`
--debug	-D	选填	工具执行报错时可打开此开关，将会展示详细保存堆栈信息。
--help	-h，-H	选填	在需要查询当前命令附属子命令或相关参数时，给出帮助建议。

- profiling：单独对推理、训练Profiling性能数据进行算子调优分析，在分析完成后，给出相关分析说明到执行终端中，并生成“ma_advisor_profiling_**.html”文件到执行目录中，目前暂不支持同时分析多卡Profiling性能数据。

```
ma-advisor analyze profiling --data-dir='/temp/profiling_dir'
```

图 6-50 命令样例

```
+-----+-------------------+-----------------------------------------+-----------------------------------------+
| No. | Problem           | Description                             | Suggestion                              |
+-----+-------------------+-----------------------------------------+-----------------------------------------+
| 1   | operator no bound | There is no mte, cube, vector, scalar   | 1. Optimize operator by AOE, such as:   |
|     |                   | ratio is more than 80.00%; Top task     | 'aoe --job_type=2                       |
|     |                   | duration operators need to be tuned are | --model_path=$user_dump_path --tune_ops_ |
|     |                   | as follows:  Addcdiv, Addcmul, Axpy,    | file=D:\operator_tuning_file_20240226102 |
|     |                   | Mul, ForeachNonFiniteCheckAndUnscale,   | 114.cfg'                                |
|     |                   | Add,  Abs, RealDiv, Square,  Sqrt       |                                         |
+-----+-------------------+-----------------------------------------+-----------------------------------------+
| 2   | block dim         | some operator does not make full use of | 1. Optimize operator by AOE, such as:   |
|     |                   | 20 ai core or 40 ai vector core;  Top-10 | 'aoe --job_type=2                       |
|     |                   | operator of task duration are as        | --model_path=$user_dump_path --tune_ops_ |
|     |                   | follows: BatchMatMulV2, MatMulV2,       | file=D:\operator_tuning_file_20240226102 |
|     |                   | ConcatD,  Mul, Sub, Slice,              | 114.cfg'                                |
|     |                   | LayerNormXBackpropV3, SoftmaxV2, Add,   |                                         |
|     |                   | LayerNormV3                             |                                         |
+-----+-------------------+-----------------------------------------+-----------------------------------------+
| 3   | AICPU operator    | Some operators and task duration exceed | 1. Modify code to avoid aicpu operator  |
|     |                   | 20 us, such as : UpsampleNearest3d,     |                                         |
|     |                   | UpsampleNearest3dGrad                   |                                         |
+-----+-------------------+-----------------------------------------+-----------------------------------------+
```

命令执行后生成AICORE算子使用AOE配置优化建议、AICPU算子优化建议的html，目前由于AOE优化不支持动态shape算子优化，因此若检测到算子均为动态shape时，将不会推荐AOE调优；除此之外，单算子问题概览会按照不同算子类型进行汇总，同时根据耗时大小进行降序显示。

图 6-51 生成结果

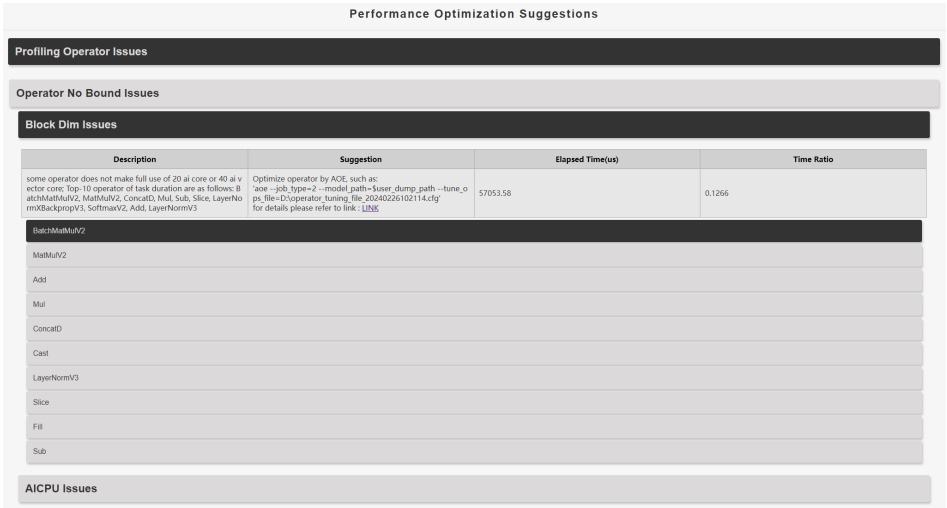


表 6-5 参数解释

参数	缩写	是否必填	说明
--data-dir	-d	必填	代表存储Profiling单卡性能数据的目录，目前暂不支持同时分析多卡Profiling目录，Profiling数据可通过如下方法获取： 在执行推理或训练程序时，请参见“**Profiling工具使用指南**”完成Profiling数据的采集、解析与导出（您可以在昇腾文档页面左上角切换版本，选择对应版本的指导文档）。数据采集时需要配置“aic-metrics”参数为“PipeUtilization”，“aicpu”参数为“on”。MA-Advisor依赖Profiling工具解析后的timeline数据、summary数据以及info.json*文件，请确保指定的“profiling_dir”目录下存在以上文件。
--cann_version	-cv	选填	使用Profiling工具采集时对应的CANN软件版本，可通过在环境中执行如下命令获取其version字段，目前配套的兼容版本为“6.3.RC2”，“7.0.RC1”和“7.0.0”，此字段不填默认按“7.0.RC1”版本数据进行处理，其余版本采集的Profiling数据在分析时可能会导致不可知问题： `cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info`
--ntework_type	-t	选填	“train”或者“infer”，不填默认为“train”。

参数	缩写	是否必填	说明
--debug	-D	选填	工具执行报错时可打开此开关，将会展示详细保存堆栈信息。
--help	-h，-H	选填	在需要查询当前命令附属子命令或相关参数时，给出帮助建议。

- timeline：单独对推理、训练timeline性能数据进行亲和API调优分析，在分析完成后，给出相关亲和API分析说明到执行终端中，并生成“ma_advisor_timeline_**.html”文件到执行目录中，目前暂不支持同时分析多卡Profiling性能数据。

```
ma-advisor analyze timeline --data-dir='/temp/profiling_dir'
```

图 6-52 命令样例

```
+-----+-----------------------+-----------------------------------------+-----------------------------------------+
| No. | Problem               | Description                             | Suggestion                              |
+-----+-----------------------+-----------------------------------------+-----------------------------------------+
| 1   | Affinity training api | Found 7 apis to be replaced based on the | 1. Please replace training api according |
|     |                       | runtime env cann-7.0.0 and torch-1.11.0  | to sub table 'Affinity training api'     |
+-----+-----------------------+-----------------------------------------+-----------------------------------------+
[2024-02-26,10:25:24][INFO] Save suggestion to ma_advisor_affinity_api_20240226102518.html.
```

命令执行后生成亲和API相关优化建议的html，将会按建议替换的亲和API进行汇总聚类，同时给出对应待替换API的堆栈信息。

图 6-53 生成结果

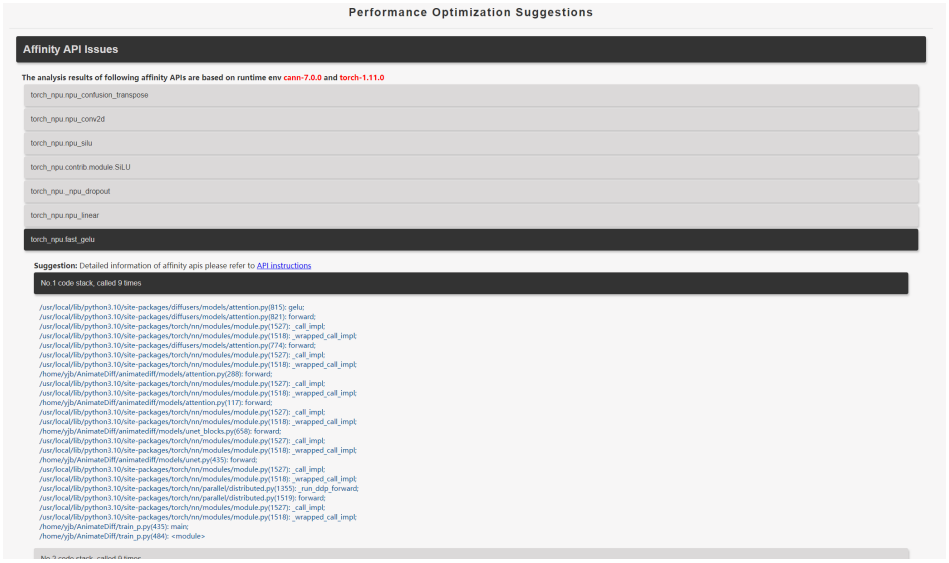


表 6-6 参数解释

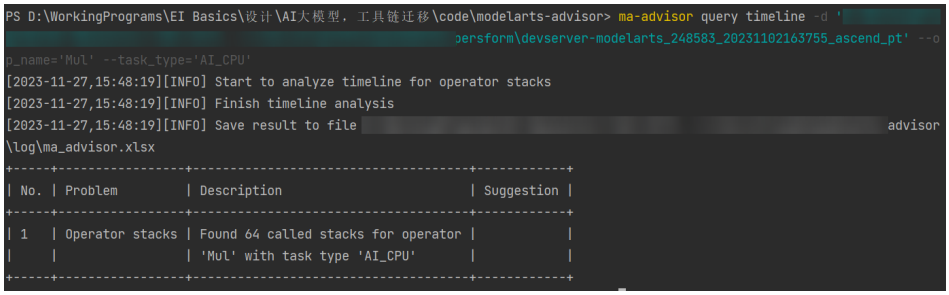
参数	缩写	是否必填	说明
--data-dir	-d	必填	代表存储Profiling单卡性能数据的目录，目前暂不支持同时分析多卡Profiling目录，Profiling数据可通过如下方法获取： • 在执行推理或训练程序时，请参见“**Profiling工具使用指南**”完成Profiling数据的采集、解析与导出（您可以在昇腾文档页面左上角切换版本，选择对应版本的指导文档）。数据采集时需要配置“aic-metrics”参数为“PipeUtilization”，“aicpu”参数为“on”，“with_stack”参数为True。 • MA-Advisor 依赖Profiling工具解析后的timeline数据、summary数据以及info.json*文件，请确保指定的“profiling_dir”目录下存在以上文件。
--cann_version	-cv	选填	使用Profiling工具采集时对应的CANN软件版本，可通过在环境中执行如下命令获取其version字段，目前配套的兼容版本为“6.3.RC2”，“7.0.RC1”和“7.0.0”，此字段不填默认按“7.0.RC1”版本数据进行处理，其余版本采集的Profiling数据在分析时可能会导致不可知问题： `cat /usr/local/Ascend/ascend-toolkit/latest/aarch64-linux/ascend_toolkit_install.info`
--debug	-D	选填	工具执行报错时可打开此开关，将会展示详细保存堆栈信息。
--help	-h，-H	选填	在需要查询当前命令附属子命令或相关参数时，给出帮助建议。

query 命令详解

timeline：单独对推理、训练timeline性能数据进行单算子详情查询，根据算子名称以及任务类型（AI_CPU|AI_CORE）进行查询，算子查询统计信息输出到运行终端，并在执行目录下的"log/ma_advisor.xlsx"文件中给出相关算子详细信息。

```
ma-advisor query timeline --data-dir='/temp/profiling_dir' --op_name='Mul' --task_type='AI_CPU'
```

图 6-54 执行命令



命令执行后生成对应算子类型查询到的详细信息的“ma-advisor*.xlsx”文件，将会给出相关算子的Taskid，以及给出对应算子的堆栈信息。

图 6-55 生成结果

Task Id	op name	op type	code stacks
59314	Mul	AI_CPU	/home/docker_home/xxk/PersFormer_3DLane/models/networks/Lane3D.py(371): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/models/PersFormer.py(128): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /usr/local/lib/python3.7/site-packages/torch_npu/utils/module.py(204): ddp_forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/experiments/runner.py(237): train; main_persformer.py(38): main; main_persformer.py(44): <module>
59320	Mul	AI_CPU	/home/docker_home/xxk/PersFormer_3DLane/models/networks/Lane3D.py(372): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/models/PersFormer.py(128): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /usr/local/lib/python3.7/site-packages/torch_npu/utils/module.py(204): ddp_forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/experiments/runner.py(237): train; main_persformer.py(38): main; main_persformer.py(44): <module>
59329	Mul	AI_CPU	/home/docker_home/xxk/PersFormer_3DLane/models/networks/Lane3D.py(371): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/models/PersFormer.py(128): forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /usr/local/lib/python3.7/site-packages/torch_npu/utils/module.py(204): ddp_forward; /usr/local/lib/python3.7/site-packages/torch/nn/modules/module.py(1110): _call_impl; /home/docker_home/xxk/PersFormer_3DLane/experiments/runner.py(237): train; main_persformer.py(38): main; main_persformer.py(44): <module>

表 6-7 参数解释

参数	缩写	是否必填	说明
--data-dir	-d	必填	代表存储Profiling单卡性能数据的目录，目录下需包含trace_view.json文件。
--op_name	-n	必填	代表待查询的算子名称，例如"Mul"。
--task_type	-t	必填	代表算子任务类型，枚举支持"AI_CPU"或"AI_CORE"。
--debug	-D	选填	工具执行报错时可打开此开关，将会展示详细保存堆栈信息。
--help	-h，-H	选填	在需要查询当前命令附属子命令或相关参数时，给出帮助建议。

auto-completion 命令详解

支持自动补全模式，在终端中自动完成ma-advisor命令补全，支持“bash（默认）/zsh/fish”。

```
ma-advisor auto-completion Bash
```

图 6-56 提示

```
Tips: please paste following shell command to your terminal to activate auto completion.

eval "$(_MA_ADVISOR_COMPLETE=bash_source ma-advisor)"
```

根据提示，在terminal中输入对应的命令即可开启在bash中对MA-Advisor相关命令自动补全功能，例如，执行如下命令后，即可在bash命令行中，后续执行ma-advisor相关命令时，使用Tab键即可自动补全：

```
eval "$(_MA_ADVISOR_COMPLETE=bash_source ma-advisor)"
```

update 命令详解

获取云端知识库至本地，可基于最新的知识库进行调优建议分析。

```
ma-advisor update rule
```

图 6-57 提示

```
PS D:\> ma-advisor update --help
Usage: ma-advisor update [OPTIONS] COMMAND [ARGS]...

  Update operation command, such as update rule and specify save path.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
  rule  Update the ma-advisor rules on the terminal. The default save path is "~/rules/cloud/". If user want to specify the save path, please use the
        environment variable "ADVISOR_RULE_PATH"
```

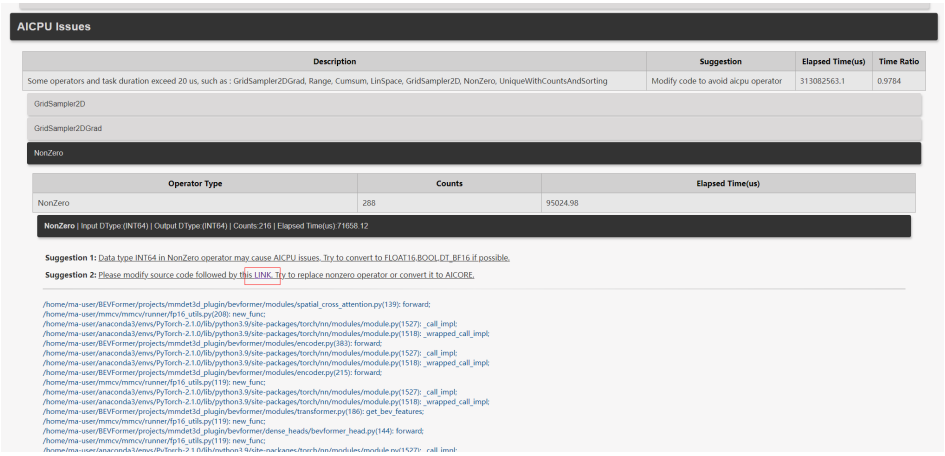
根据提示，在terminal中，可以通过“ADVISOR_RULE_PATH”环境变量设置知识库的本地路径。

工具扫描结果解读

- **AI CPU算子分析和处理**

  MA-Advisor工具分析结果的html文件中会有下述链接，提供AI CPU算子相关问题的修复指导和案例。

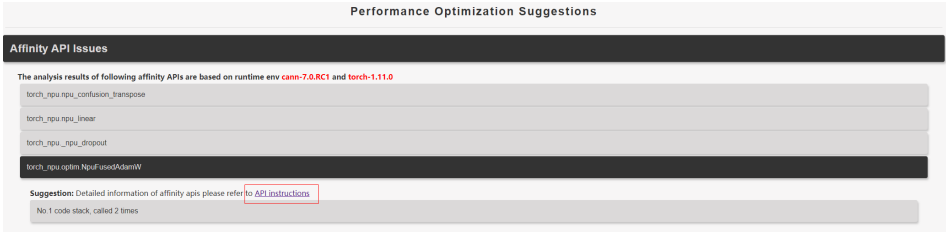
  图 6-58 AI CPU 算子分析和处理



- **亲和API替换**

  MA-Advisor工具分析结果的html文件中会有下述链接，提供亲和API替换相关问题的修复指导和代码样例。

  图 6-59 亲和 API 替换



6.3.6.2.3 昇腾迁移融合算子 API 替换样例

部分torch原生的API在下发和执行时会包括多个小算子，下发和执行耗时较长，可以通过替换成NPU API来使能融合算子，提升训练性能。

API 替换总览

- **torch_npu.optim.NpuFusedAdamW**
- **optimizer.clip_grad_norm_fused_**
- **torch_npu.npu_confusion_transpose**
- **torch_npu.npu_scaled_masked_softmax**
- **torch_npu.fast_gelu**
- **torch_npu.npu_rms_norm**
- **torch_npu.npu_swiglu**

- **torch_npu.npu_rotary_mul**
- **torch_npu.npu_fusion_attention**

上述torch_npu api的功能和参数描述见**概述**。

优化器替换

替换优化器一般都能有较大的性能受益，可以优先考虑将torch原生的优化器替换为**昇腾提供的亲和优化器**。下文以AdamW优化器为例，其他优化器的替换方式一致。

- **torch_npu.optim.NpuFusedAdamW**

torch原生代码示例如下：

```
import torch
optimizer = torch.optim.AdamW(
  model.parameters(),
  learning_rate,
  momentum=momentum,
  weight_decay=weight_decay
)
```

torch_npu代码示例如下：

```
import torch_npu
from torch_npu.contrib import transfer_to_npu

optimizer = torch_npu.optim.NpuFusedAdamW(
  model.parameters(),
  learning_rate,
  momentum=momentum,
  weight_decay=weight_decay
)
```

亲和 API 替换

- **optimizer.clip_grad_norm_fused_**

  在替换为npu亲和梯度裁剪api之前，请确保代码中已使用npu亲和优化器。

  torch原生代码示例如下：

  ```
  import torch
  optimizer = torch.optim.AdamW(model.parameters(), lr = lr)
  torch.nn.utils.clip_grad_norm_(parameters=model.parameters(), max_norm=10, norm_type=2)
  ```

  torch_npu代码示例如下：

  ```
  import torch
  import torch_npu
  from torch_npu.contrib import transfer_to_npu

  optimizer = torch_npu.optim.NpuFusedAdamW(model.parameters(), lr = lr)
  optimizer.clip_grad_norm_fused_(max_norm=10, norm_type=2)
  ```

- **torch_npu.npu_confusion_transpose**

  示例一

  torch原生代码示例如下：

  ```
  import torch

  data = torch.rand(64, 3, 64, 128).cuda()
  batch, channel, height, width = data.shape
  result = torch.permute(data, (0, 2, 1, 3)).reshape(height, batch, channel*width)
  ```

  torch_npu代码示例如下：

  ```
  import torch
  import torch_npu
  ```

```
from torch_npu.contrib import transfer_to_npu

data = torch.rand(64, 3, 64, 128).cuda()
batch, channel, height, width = data.shape
result = torch_npu.npu_confusion_transpose(data, (0, 2, 1, 3), (height, batch, channel*width),
transpose_first=True)
```

示例二

torch原生代码示例如下：

```
import torch

data = torch.rand(64, 3, 64, 128).cuda()
batch, channel, height, width = data.shape
result = dat.view(batch, height*channel*width).transpose(1, 0)
```

torch_npu代码示例如下：

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

data = torch.rand(64, 3, 64, 128).cuda()
batch, channel, height, width = data.shape
result = torch_npu.npu_confusion_transpose(data, (1, 0), (batch, height*channel*width),
transpose_first=False)
```

- **torch_npu.npu_scaled_masked_softmax**

  需要注意的，atten_mask和atten_scores张量最后一维的取值范围为32-8192，且必须为32的整数倍。

  torch原生代码示例如下：

  ```
  import torch
  x = torch.randn([64, 8, 128, 256]).cuda()
  mask = torch.randn([1, 1, 128, 256]).cuda() >= 1
  scale = 0.8

  output = torch.softmax((x * scale).masked_fill(mask, -1*torch.inf), dim=-1)
  # shape is (64, 8, 128, 256)
  ```

  torch_npu代码示例如下：

  ```
  import torch
  import torch_npu
  from torch_npu.contrib import transfer_to_npu

  x = torch.randn([64, 8, 128, 256]).cuda()
  mask = torch.randn([1, 1, 128, 256]).cuda() >= 1
  scale = 0.8

  output = torch_npu.npu_scaled_masked_softmax(x, mask, scale)
  # shape is (64, 8, 128, 256)
  ```

- **torch_npu.fast_gelu**

  示例一

  替换torch.nn.functional.fast_gelu方法，实现上有些差异，激活函数输出结果会不同。

  torch原生代码示例如下：

  ```
  import torch
  input_data = torch.rand(64, 32).cuda()
  result = torch.nn.functional.gelu(input_data)
  ```

  torch_npu代码示例如下：

  ```
  import torch
  import torch_npu
  from torch_npu.contrib import transfer_to_npu

  input_data = torch.rand(64, 32).cuda()
  result = torch_npu.fast_gelu(input_data)
  ```

示例二

继承torch.nn.GELU，基于torch_npu.fast_gelu重写forward方法。

torch原生代码示例如下：

```
import torch
input_data = torch.rand(64, 32).cuda()
gelu_module = torch.nn.GELU().cuda()
result3 = gelu_module(input_data)
```

torch_npu代码示例如下：

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

# 继承torch.nn.GELU，基于torch_npu.fast_gelu重写forward方法
class FastGelu(torch.nn.GELU):
    def forward(self, input_data):
        return torch_npu.fast_gelu(input_data)

input_data = torch.rand(64, 32).cuda()
fast_gelu_module = FastGelu().cuda()
result = fast_gelu_module(input_data)
```

- **torch_npu.npu_rms_norm**

  输入数据dtype仅支持float16、bfloat16、float。

  torch原生代码示例如下：

```
import torch

class TorchRMSNorm(torch.nn.Module):
    def __init__(self, dim: int, eps = 1e-6):
        super().__init__()
        self.eps = eps
        self.weight = nn.Parameter(torch.ones(dim)).cuda()

    def _norm(self, x):
        return x * torch.rsqrt(x.pow(2).mean(-1, keepdim=True) + self.eps)

    def forward(self, x):
        output = self._norm(x.float()).type_as(x)
        return output * self.weight

input_data = torch.randn(128, 256).cuda()
torch_rms_norm = TorchRMSNorm((128, 256))
result = torch_rms_norm(data)
```

  torch_npu代码示例如下：

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

class NpuRMSNorm(torch.nn.Module):
    def __init__(self, dim: int, eps = 1e-6):
        super().__init__()
        self.eps = eps
        self.weight = nn.Parameter(torch.ones(dim)).cuda()

    def forward(self, x):
        return torch_npu.npu_rms_norm(x, self.weight, epsilon=self.eps)[0]

input_data = torch.randn(128, 256).cuda()
npu_rms_norm = NpuRMSNorm((128, 256))
result = npu_rms_norm(data)
```

- **torch_npu.npu_swiglu**

  输入数据dtype仅支持float16、bfloat16、float。

  torch原生代码示例如下：

```
import torch
class TorchSwiGlu(torch.nn.Module):
    def __init__(self, dim = -1):
        super().__init__()
        self.dim = dim

    def _swiglu(self, x):
        x = torch.chunk(x, 2, -1)
        return torch.nn.functional.silu(x[0]) * x[1]

    def forward(self, x):
        output = self._swiglu(x)
        return output

input_data = torch.randn(128, 256).cuda()
torch_swiglu = TorchSwiGlu()
result = torch_swiglu(data)
```

torch_npu代码示例如下：

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

class NpuSwiGlu(torch.nn.Module):
    def __init__(self, dim = -1):
        super().__init__()
        self.dim = dim

    def forward(self, x):
        dim = -1
        return torch_npu.npu_swiglu(x, dim=dim)

input_data = torch.randn(128, 256).cuda()
npu_swiglu = NpuSwiGlu()
result = npu_swiglu(data)
```

- **torch_npu.npu_rotary_mul**

  torch原生代码示例如下：

  ```
  import torch

  x = torch.rand([2, 8192, 5, 128]).cuda()
  r1 = torch.rand([1, 8192, 1, 128]).cuda()
  r2 = torch.rand([1, 8192, 1, 128]).cuda()

  def torch_func(x, r1, r2):
    x1, x2 = x[..., : x.shape[-1] // 2], x[..., x.shape[-1] // 2:]
    # x1, x2 = torch.chunk(x, 2, -1)
    x_new = torch.cat((-x2, x1), dim=-1)
    output = r1 * x + r2 * x_new
    return output

  result = torch_func(x, r1, r2)
  ```

  torch_npu代码示例如下：

  ```
  import torch
  import torch_npu
  from torch_npu.contrib import transfer_to_npu

  x = torch.rand([2, 8192, 5, 128]).cuda()
  r1 = torch.rand([1, 8192, 1, 128]).cuda()
  r2 = torch.rand([1, 8192, 1, 128]).cuda()

  result = torch_npu.npu_rotary_mul(x, r1, r2)
  ```

- **torch_npu.npu_fusion_attention**

  torch原生代码示例如下：

  ```
  import torch

  class TorchFlashAttention():
  ```

```
    def supported_op_exec(self, query, key, value, atten_mask=None):
        scale = 0.099
        qk = torch.matmul(query, key.transpose(2, 3)).mul(scale)

        if atten_mask is not None:
            qk.masked_fill_(atten_mask.npu(), torch.tensor(-float('inf')).npu())
        softmax_res = torch.nn.functional.softmax(qk, dim=-1, dtype=torch.float32).to(torch.float16)
        output = torch.matmul(softmax_res, value)
        output = output.transpose(1, 2)
        output = output.reshape(output.shape[0], output.shape[1], -1)
        return output

    def custom_op_exec(self, query, key, value, atten_mask=None):
        scale = 0.099
        return torch_npu.npu_fusion_attention(
            query, key, value, head_num=32, input_layout="BSH", scale=scale, atten_mask=atten_mask)

    def trans_BNSD2BSH(self, tensor: torch.Tensor):
        tensor = torch.transpose(tensor, 1, 2)
        tensor = torch.reshape(tensor, (tensor.shape[0], tensor.shape[1], -1))
        return tensor

    def test_torch_flash_attention(self, device="npu"):
        query = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        key = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        value = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        atten_mask = torch.randn(1, 1, 128, 128, dtype=torch.float16).npu() >= 0

        q_npu = self.trans_BNSD2BSH(query).npu()
        k_npu = self.trans_BNSD2BSH(key).npu()
        v_npu = self.trans_BNSD2BSH(value).npu()

        result = self.supported_op_exec(query.npu(), key.npu(), value.npu(), atten_mask=atten_mask)
        # result shape (1, 128, 4096)
```

torch_npu代码示例如下：

```
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu


class NPUFlashAttention():

    def npu_exec(self, query, key, value, atten_mask=None):
        scale = 0.099
        return torch_npu.npu_fusion_attention(
            query, key, value, head_num=32, input_layout="BSH", scale=scale, atten_mask=atten_mask)

    def trans_BNSD2BSH(self, tensor: torch.Tensor):
        tensor = torch.transpose(tensor, 1, 2)
        tensor = torch.reshape(tensor, (tensor.shape[0], tensor.shape[1], -1))
        return tensor

    def test_npu_flash_attention(self, device="npu"):
        query = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        key = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        value = torch.randn(1, 32, 128, 128, dtype=torch.float16)
        atten_mask = torch.randn(1, 1, 128, 128, dtype=torch.float16).npu() >= 0

        q_npu = self.trans_BNSD2BSH(query).npu()
        k_npu = self.trans_BNSD2BSH(key).npu()
        v_npu = self.trans_BNSD2BSH(value).npu()

        result, softmax_max, softmax_sum, softmax_out, seed, offset, numels = self.npu_exec(q_npu,
k_npu, v_npu, atten_mask)
        # result shape (1, 128, 4096)
```

6.3.6.2.4 AI CPU 算子替换样例

部分算子因为数据输入类型问题或者算子实现问题，导致会在昇腾芯片的AI CPU上执行，没有充分利用AI CORE的资源，从而导致计算性能较差，影响训练速度。部分场景下，可以通过修改Python代码来减少这类AI CPU算子，从而提升训练性能。

当前对 AICPU 算子识别到的调优方式主要包含两种：

- PyTorch数据类型转换，将执行在AICPU上的类型算子转换为执行在AICORE单元的算子。
- 等价的算子替换。

类型转换方式

当前PyTorch支持的dtype类型如下，详见**Link**。

图 6-60 PyTorch 支持的 dtype

TENSOR ATTRIBUTES

Each `torch.Tensor` has a `torch.dtype`, `torch.device`, and `torch.layout`.

torch.dtype

CLASS torch.dtype

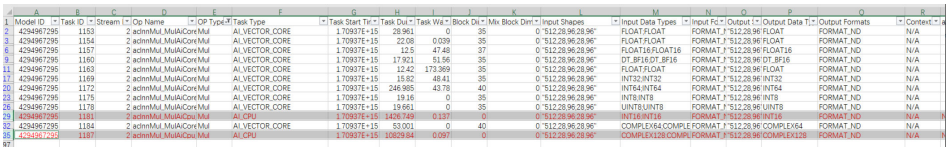
A `torch.dtype` is an object that represents the data type of a `torch.Tensor`. PyTorch has twelve different data types:

Data type	dtype	Legacy Constructors
32-bit floating point	`torch.float32` or `torch.float`	`torch.*.FloatTensor`
64-bit floating point	`torch.float64` or `torch.double`	`torch.*.DoubleTensor`
64-bit complex	`torch.complex64` or `torch.cfloat`	
128-bit complex	`torch.complex128` or `torch.cdouble`	
16-bit floating point 1	`torch.float16` or `torch.half`	`torch.*.HalfTensor`
16-bit floating point 2	`torch.bfloat16`	`torch.*.BFloat16Tensor`
8-bit integer (unsigned)	`torch.uint8`	`torch.*.ByteTensor`
8-bit integer (signed)	`torch.int8`	`torch.*.CharTensor`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.*.ShortTensor`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.*.IntTensor`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.*.LongTensor`
Boolean	`torch.bool`	`torch.*.BoolTensor`

基于此对常见的算子如MUL、EQUAL、TENSOREQUAL等做单算子测试，看有哪些类型的算子是执行在AICPU上的，然后尝试转换到支持AICORE单元的类型dtype上计算，实现效率提升的目的。

- MUL

图 6-61 Mul



AICORE支持的dtype。

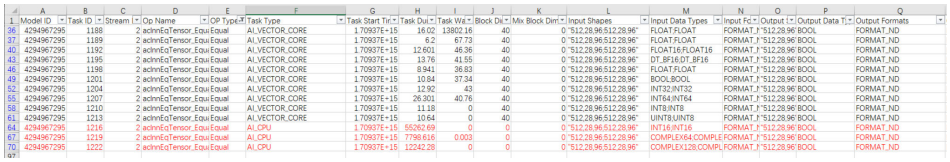
float, float32, float16, dt_bf16, float64, int32, int64, int8, uint8, complex641

AICPU 类型的 dtype。

int16, complex128

- Equal

图 6-62 Equal



AICORE支持的dtype。

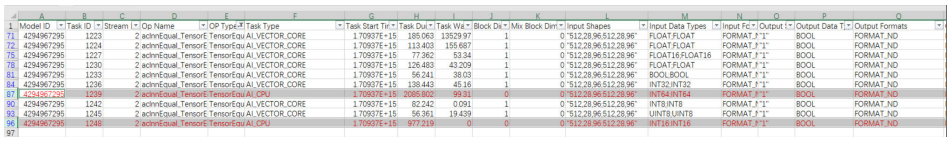
float, float32, float16, dt_bf16, float64, bool, int32, int64, int8, uint81

AICPU 类型的 dtype。

int16, complex64,complex128

- TensorEqual

图 6-63 TensorEqual



AICORE支持的dtype。

float, float32, float16, dt_bf16, float64, bool, int32, int8, uint81

AICPU 类型的 dtype。

int16, int64

算子等价替换

- Index算子替换
  - 情形一： index by index

    这种操作会造成输出的shape和输入的shape不一致，我们可以直接用index_select(gatherV2)操作替换该算子运行在aicore性能高上很多

    图 6-64 index by index

  - 情形二：index_put by index

    tensor[index] = 3

    这类操作尽量避免，没有特别好的替代方式，可以将index转化成mask，或者一开始就生成mask作为索引而不是index。

    如果要替换可以用scatter算子替换，目前发现用到这种场景时index一般比较少，所以用index方式可能性能更高。
  - 情形三：index_put by mask

    tensor_a[mask] = 3

index_put by mask可以通过where (selectV2)算子来替代。这种方式与原先语义不同的是，会返回一个新的tensor。

图 6-65 index_put by mask

index by mask或者index_put by mask相对来说对NPU和框架比较友好。关键在保持shape这样不需要contiguous，然后将必要的index抽取操作放在最后。在index比较少的情况下，index操作就比较快了，可能优于替换。

- IndexPut算子替换

  在tensor类型的赋值和切片操作时，会使用IndexPut算子执行，一般都在AICPU上执行，可以转换为等价的tensor操作转换到CUBE单元上执行。例如：

  ```
  masked_input[input_mask] = 0
  ```

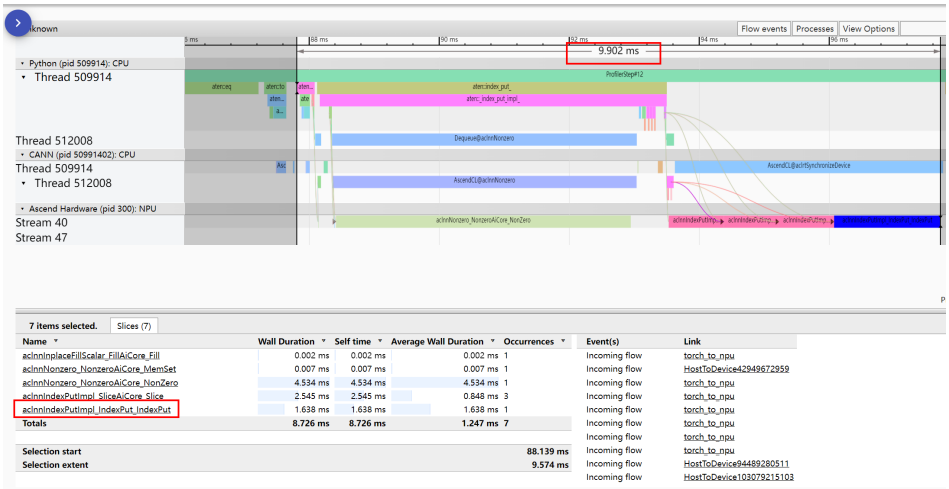
  建议替换为：

  ```
  masked_input *= ~input_mask
  ```

  此处是将IndexPut的masked_input是float类型的tensor数据，input_mask是和masked_input shape 一致的bool类型tensor或者01矩阵。由于是赋0操作，所以先对input_mask 取反后再进行乘法操作。

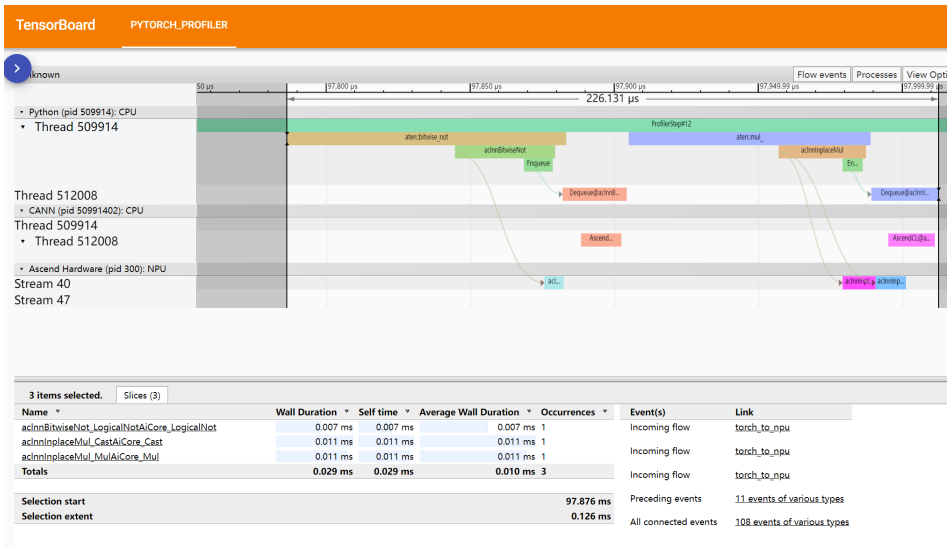
  以赋0操作为例，在shape = (512, 32, 64) 类型float32 数据上测试，替换前耗时: 9.639978408813477 ms，替换之后耗时为 0.1747608184814453 ms，如下图，替换前，总体耗时在9.902ms，Host下发到device侧执行5个算子，其中aclnnIndexPutImpl_IndexPut_IndexPut是执行在 AICPU上。

  图 6-66 替换前耗时

  替换后，总体耗时226.131us。下发三个执行算子，均执行在AI CORE上。

图 6-67 替换后耗时

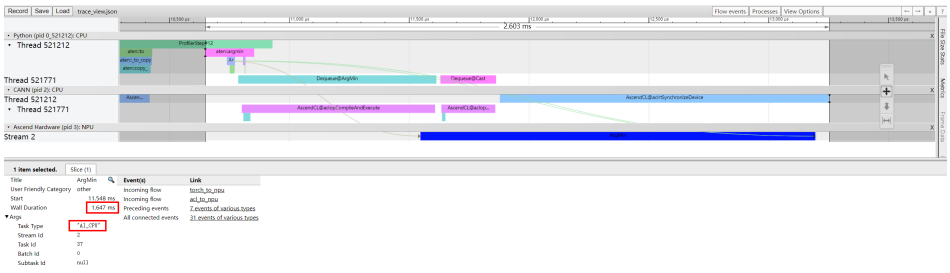


- ArgMin算子优化

  ArgMin在CANN 6.3 RC2版本上算子下发到 AICPU执行，在CANN 7.0RC1上下发到AI_CORE 上边执行。出现此类情形建议升级CANN包版本。

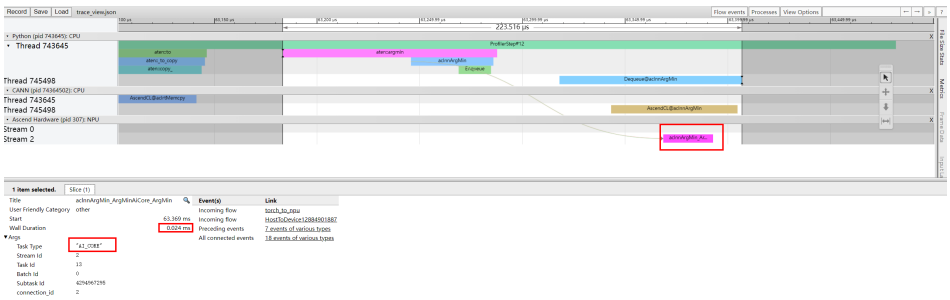
  在shape大小是 (1024, 1024) 的tensor上测试，结果如下：CANN 6.3.RC2上，单算子执行时间 2.603 ms。

图 6-68 单算子执行时间（CANN 6.3.RC2）



CANN7.0 RC1上，单算子执行时间 223.516 us。

图 6-69 单算子执行时间（CANN7.0 RC1）



- nonzero算子优化

  将mask转化为index，对于所有值大于0的tensor在某些计算中可以利用乘法替代。比如要对mask的tensor求和。tensor_a[mask].sum()就相当于(tensor_a * mask).sum()。

例如：

```
shape = (1024, )
mask= torch.randint(-1, 2, shape).npu()
tensor_a = torch.ones(shape).float().npu()
mask_inds = torch.nonzero(
        gt_inds > 0, as_tuple=False).squeeze(1)

tensor_sum = tensor_a[mask_inds].sum()
```

就相当于：

```
shape = (1024, )
mask= torch.randint(-1, 2, shape).npu()
tensor_a = torch.ones(shape).float().npu()
mask_inds = torch.nonzero( gt_inds > 0, as_tuple=False).squeeze(1)
tensor_sum2 = (tensor_a * mask_inds2).sum()
```

6.3.6.3 性能可视化工具 Ascend-Insight 使用指导

对于高阶的调优用户，可以使用可视化profiling数据查看数据详情并分析可优化点，昇腾提供了Ascend-Insight可视化工具，相比于chrometrace等工具提供了更优的功能和性能。详见昇腾**《Ascend-Insight用户指南》**。

6.3.6.4 其他性能分析工具

对于GPU和NPU性能比对、NPU多次训练之间性能比对的场景，昇腾提供了性能比对工具**compare_tools**，通过对训练耗时和内存占用的比对分析，定位到具体劣化的算子，帮助用户提升性能调优的效率。工具将训练耗时拆分为计算、通信、调度三大维度，并针对计算和通信分别进行算子级别的比对；将训练占用的总内存，拆分成算子级别的内存占用进行比对。

对于集群训练场景，昇腾提供了集群分析工具**cluster_analysis**，当前主要对基于通信域的迭代内耗时分析、通信时间分析以及通信矩阵分析为主，从而定位慢卡、慢节点以及慢链路问题。

6.3.7 训练网络迁移总结

- 确保算法在GPU训练时，持续稳定可收敛。避免在迁移过程中排查可能的算法问题，并且要有好的对比标杆。如果是NPU上全新开发的网络参考**PyTorch迁移精度调优**，排查溢出和精度问题。
- 理解GPU和NPU的构造以及运行的差别，有助于在迁移过程中分析问题并发挥NPU的优势。由于构造和运行机制的差别，整个迁移过程并非是完全平替，GPU在灵活性上是有其独特的优势的，而NPU上的执行目前还是依赖于算子的下发，对于NPU构造的理解是昇腾训练迁移中必备的知识，只有对于昇腾有基础理解，配合一些诊断工具，面对复杂问题时，才能进行进一步诊断与定位，进而发挥NPU的能力。
- 性能调优可以先将重点放在NPU不亲和的问题处理上，确保一些已知的性能问题和优化方法得到较好的应用。通用的训练任务调优、参数调优可以通过可观测数据来进行分析与优化，一般来说分段对比GPU的运行性能会有比较好的参考。算子级的调优某些情况下如果是明显的瓶颈或者性能攻坚阶段，考虑到门槛较高，可以联系华为工程师获得帮助。
- 精度问题根因和表现种类很多，会导致问题定位较为复杂，一般还是需要GPU上充分稳定的网络（包含混合精度）再到NPU上排查精度问题。常见的精度调测手段，包含使用全精度FP32，或者关闭算子融合开关等，先进行排查。对于精度问题，系统工程人员需要对算法原理有较深入的理解，仅从工程角度分析有时候会非常受限，同时也可联系华为工程师进行诊断与优化。

6.4 基于 AIGC 模型的 GPU 推理业务迁移至昇腾指导

6.4.1 场景介绍

阅读本文前建议您先了解以下内容：

- Stable Diffusion的基础知识，可参考**Stable Diffusion github**、**Stable Diffusion wikipedia**、**diffusers github**、**Stable Diffusion with diffusers**。
- 推理业务迁移到昇腾的通用流程，可参考**GPU推理业务迁移至昇腾的通用指导**。

> **说明**
>
> 由于Huggingface网站的限制，访问Stable Diffusion链接时需使用代理服务器，否则可能无法访问网站。

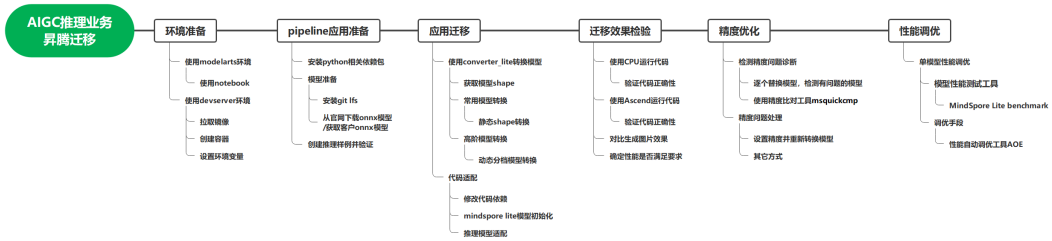
在Stable Diffusion迁移适配时，更多的时候是在适配Diffusers和Stable Diffusion WebUI，使其能够在昇腾的设备上运行。其中，Diffusers遵循了Huggingface的“**single-file policy**”的设计原则，它的三个主要模块Pipeline、Schedulers和预训练模型中，Pipeline和Schedulers都完全遵循了“single-file policy”原则。该设计原则更推荐直接复制粘贴代码，而不是进行抽象处理。因此，与模型前向运算相关的所有源代码都被直接复制粘贴到同一个文件中，而不是调用某些抽象提取出的模块化库。Diffusers的这种设计原则的好处是代码简单易用、对代码贡献者友好。然而，这种反软件结构化的设计也有明显的缺点。由于缺乏统一的模块化库，对于昇腾适配而言变得更加复杂，必须针对每个不同业务的Pipeline进行单独适配。本文以**Stable Diffusion v1.5**的图生图为例，通过可以直接执行的样例代码介绍Diffusers的昇腾迁移过程。对于其他pipeline的迁移，可以在充分理解其代码的基础上，参考本文的思路进行举一反三。Stable Diffusion WebUI的迁移不包含在本文中，具体原因详见**Stable Diffusion WebUI如何适配?** 。

AI推理应用运行在昇腾设备上一般有两种方式：

- 方式1：通过Ascend PyTorch，后端执行推理，又称在线推理。
- 方式2：通过模型静态转换后，执行推理，又称离线推理。

通常为了获取更好的推理性能，推荐使用方式2的离线推理。下文将以Diffusers img2img onx pipeline为示例来讲解如何进行离线推理模式下的昇腾迁移。迁移的整体流程如下图所示：

图 6-70 迁移流程图



6.4.2 迁移环境准备

迁移环境准备有以下两种方式：

- 方式一 ModelArts Notebook：该环境为在线调试环境，主要面向演示、体验和快速原型调试场景。
  - 优点：可快速、低成本地搭建环境，使用标准化容器镜像，官方notebook示例可直接运行。
  - 缺点：由于是容器化环境因此不如裸机方式灵活，例如不支持root权限操作、驱动更新等。
  - 环境开通指导参考：**Notebook环境创建**。
  - 样例演示可参考**Notebook样例：Stable Diffusion模型迁移到Ascend上进行推理**。
- 方式二 ModelArts Lite DevServer：该环境为裸机开发环境，主要面向深度定制化开发场景。
  - 优点：支持深度自定义环境安装，可以方便的替换驱动、固件和上层开发包，具有root权限，结合配置指导、初始化工具及容器镜像可以快速搭建昇腾开发环境。
  - 缺点：资源申请周期长，购买成本高，管理视角下资源使用效率较低。
  - 环境开通指导参考：**DevServer资源开通**
  - 环境配置指导参考：**Snt9B裸金属服务器环境配置指南**

本文基于方式二的环境进行操作，请参考方式二中的环境开通和配置指导完成裸机和容器开发初始化配置。注意业务基础镜像选择**Ascend+PyTorch**镜像。

配置好的容器环境如下图所示：

图 6-71 环境配置完成

```
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker run -itd --cap-add=SYS_PTRACE -e ASCEND_VISIBLE_DEVICES=3 -v
/home:/host_home -u=0 --name pytorch_test swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_1_11_ascend:pytorch_1.
11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-d910b-20230815141604-3685231 /bin/bash
0292be41ac1ef03a37b7c78adcff4fc999a967e1163e5f6e565edbe6a638c69b
[root@devserver-modelarts-demanager-0eaabe8f ~]# docker exec -ti 0292be41a bash
The environment has been set
[root@0292be41ac1e ma-user]# source .bashrc
The environment has been set
The environment has been set
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]# python3 -c "import torch;import torch_npu; a = torch.randn(3, 4).npu();
 print(a + a);"
tensor([[-1.0911, -0.4146,  1.6027,  1.8585],
        [ 3.2549,  0.7026,  2.9356,  0.9544],
        [ 5.1409, -0.8820, -0.3400,  0.0257]], device='npu:0')
(PyTorch-1.11.0) [root@0292be41ac1e ma-user]#
```

6.4.3 pipeline 应用准备

当前迁移路径是从ONNX模型转换到MindIR模型，再用MindSpore Lite做推理， 所以迁移前需要用户先准备好自己的ONNX pipeline。下文以官方开源的图生图的Stable Diffusion v1.5的onnx pipeline代码为例进行说明。

步骤1 进入容器环境，创建自己的工作目录，由于在**Snt9B裸金属服务器环境配置指南**的配置环境步骤中，在启动容器时将物理机的home目录挂载到容器的“/home_host”目录下，该目录可以直接使用上传到物理机“home”目录下的文件。本文中，将基于容器的“/home_host”目录创建工作目录：

```
mkdir -p /home_host/work
cd /home_host/work
```

步骤2 在迁移onnx pipeline前，首先需要确保原始的onnx pipeline能在昇腾机器的ARM CPU上正常执行。进入容器环境后，安装依赖包。

```
pip install torch==1.11.0 onnx transformers==4.27.4 accelerate onnxruntime diffusers==0.11.1
```

步骤3 下载git lfs，用于下载git仓中的大文件。由于欧拉源上没有git-lfs包，所以需要从压缩包中解压使用，在浏览器中输入如下地址下载git-lfs压缩包并上传到服务器的/home目录。

```
https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
```

步骤4 安装git lfs：

```
tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
cd git-lfs-3.2.0
sh install.sh
rm -rf git-lfs-linux-arm64-v3.2.0.tar.gz git-lfs-3.2.0
```

步骤5 通过git下载sd pytorch模型。

该模型用于获取模型shape，也可以转换生成onnx模型。后文中的modelarts-ascend仓库已经给出了模型shape，可以直接使用，onnx模型也可以单独下载。

```
# git clone sd模型
git lfs install
mkdir -p /home_host/work/runwayml
cd /home_host/work/runwayml
git clone https://huggingface.co/runwayml/stable-diffusion-v1-5/ -b main
# 将下载的文件夹重命名，以便后续脚本中引用
mv stable-diffusion-v1-5 pytorch_models
```

说明

这里由于Huggingface网站的限制以及模型文件的大小原因，很可能会下载失败。可以进到**Huggingface网站**，从浏览器下载模型后，再手动上传到物理机/home/pytorch_models目录下。

步骤6 通过git下载sd onnx模型。

```
# git clone sd模型
git lfs install
cd /home_host/work/runwayml
git clone https://huggingface.co/runwayml/stable-diffusion-v1-5 -b onnx
# 将下载的文件夹重命名，以便后续脚本中引用
mv stable-diffusion-v1-5 onnx_models
```

说明

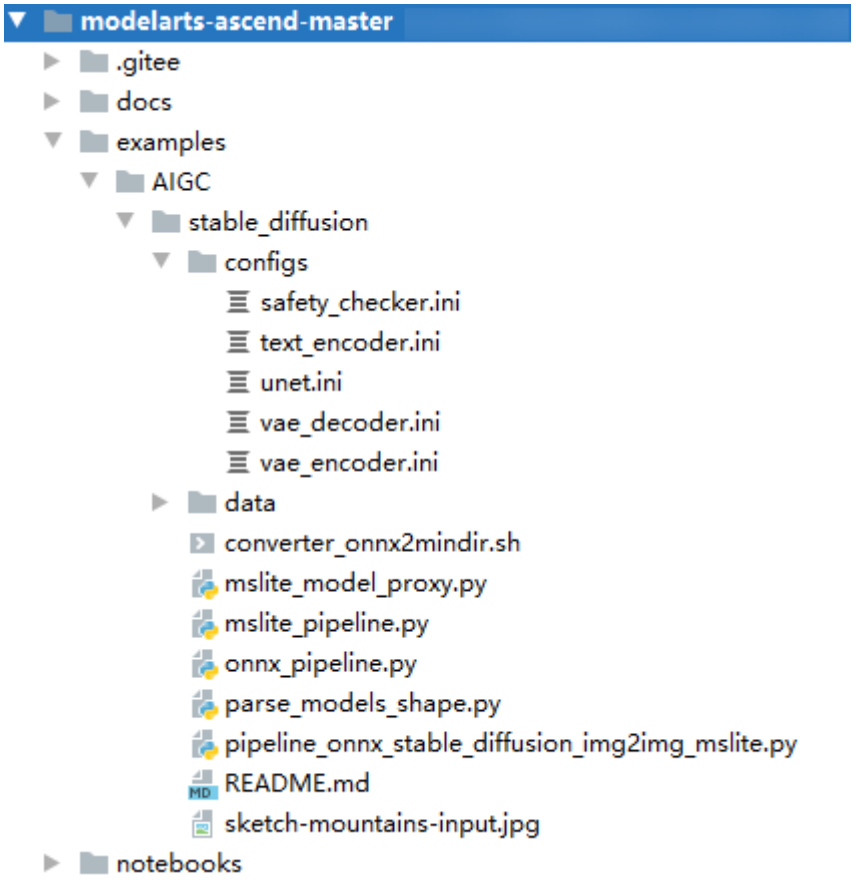
这里由于Huggingface网站的限制以及模型文件的大小原因，很可能会下载失败。可以进到**Huggingface网站**，从浏览器下载模型后，再手动上传到物理机/home/onnx_models目录下。

步骤7 下载好模型后，需要编写推理脚本。为了便于讲解，本指导中所需的代码已发布在ModelArts代码仓，可以使用如下命令下载推理脚本样例代码：

```
cd /home_host/work
git clone https://gitee.com/ModelArts/modelarts-ascend.git
ll modelarts-ascend/examples/AIGC/stable_diffusion
```

代码目录如下图所示，onnx_pipeline.py是图生图推理脚本。mslite_pipeline.py、mslite_model_proxy.py、pipeline_onnx_stable_diffusion_img2img_mslite.py是迁移后的文件，其中mslite_model_proxy.py是代理模型类，pipeline_onnx_stable_diffusion_img2img_mslite.py是从Stable Diffusion源码中的pipeline复制并修改的，这些文件在后续的章节中会使用并做进一步讲解。

图 6-72 代码目录



步骤8 将“modelarts-ascend/examples/AIGC/stable_diffusion/onnx_pipeline.py”文件中的“onnx_model_path”改为**步骤6**中下载的onnx_models地址“/home_host/work/runwayml/onnx_models”。执行推理脚本进行测试，这里使用的推理硬件是CPU，由于CPU执行较慢，验证待迁移的代码可能需要大约15分钟左右才能完成：

```
cd modelarts-ascend/examples/AIGC/stable_diffusion # 必须执行该命令，否则会报错找不到sketch-mountains-input.jpg
python onnx_pipeline.py
```

生成的图片fantasy_landscape.png会保存在当前路径下，该图片也可以作为后期精度校验的一个对比。

图 6-73 生成图片

----结束

6.4.4 应用迁移

6.4.4.1 模型适配

MindSpore Lite是华为自研的推理引擎，能够最大化地利用昇腾芯片的性能。在使用MindSpore Lite进行离线推理时，需要先将模型转换为mindir模型，再利用MindSpore Lite作为推理引擎，将转换后的模型直接运行在昇腾设备上。模型转换需要使用converter_lite工具。

Huggingface提供的onnx模型文件的输入是动态shape，而mindir不支持动态shape，只能使用静态shape或者几个固定档位的分档shape代替。使用converter_lite转换模型时，也分为静态shape和分档shape两种方式，需要根据具体的业务需求使用对应的转换方式。本次迁移使用的是静态shape方式进行模型转换。

获取模型 shape

由于在后续模型转换时需要知道待转换模型的shape信息，这里指导如何通过训练好的stable diffusion pytorch模型获取模型shape，主要有如下两种方式获取：

- 方式一：通过stable diffusion的pytorch模型获取模型shape。

- 方式二：通过查看**ModelArts-Ascend代码仓库**，根据每个模型的configs文件获取已知的shape大小。

下文主要介绍方式1如何通过stable diffusion的pytorch模型获取模型shape。

步骤1 在pipeline应用准备章节，已经下载到sd的pytorch模型（ /home_host/work/runwayml/pytorch_models ）。进入工作目录：

```
cd /home_host/work
```

步骤2 新建python脚本文件“parse_models_shape.py”用于获取shape，其中model_path是指上面下载的pytorch_models的路径。

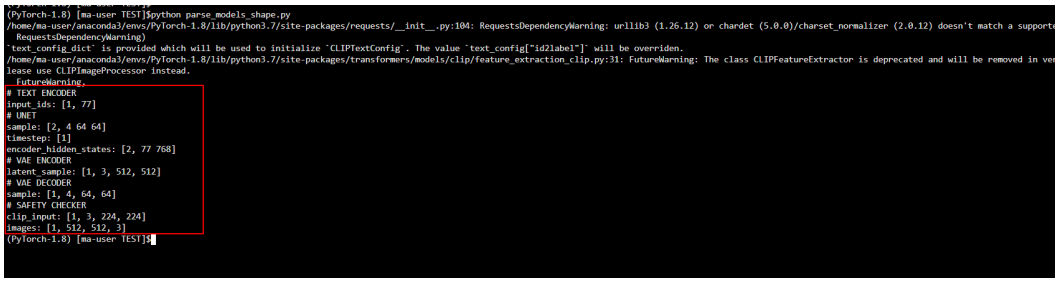
```
# parse_models_shape.py
import torch
import numpy as np
from diffusers import StableDiffusionPipeline

model_path = '/home_host/work/runwayml/pytorch_models'
pipeline = StableDiffusionPipeline.from_pretrained(model_path, torch_dtype=torch.float32)

# TEXT ENCODER
num_tokens = pipeline.text_encoder.config.max_position_embeddings
text_hidden_size = pipeline.text_encoder.config.hidden_size
text_input = pipeline.tokenizer(
    "A sample prompt",
    padding="max_length",
    max_length=pipeline.tokenizer.model_max_length,
    truncation=True,
    return_tensors="pt",
)
print("# TEXT ENCODER")
print(f"input_ids: {np.array(text_input.input_ids.shape).tolist()}")

# UNET
unet_in_channels = pipeline.unet.config.in_channels
unet_sample_size = pipeline.unet.config.sample_size
print("# UNET")
print(f"sample: [{2}, {unet_in_channels} {unet_sample_size} {unet_sample_size}]")
print(f"timestep: [{1}]")  # 此处应该是1，否则和后续的推理脚本不一致
print(f"encoder_hidden_states: [{2}, {num_tokens} {text_hidden_size}]")

# VAE ENCODER
vae_encoder = pipeline.vae
vae_in_channels = vae_encoder.config.in_channels
vae_sample_size = vae_encoder.config.sample_size
print("# VAE ENCODER")
print(f"sample: [{1}, {vae_in_channels}, {vae_sample_size}, {vae_sample_size}]")

# VAE DECODER
vae_decoder = pipeline.vae
vae_latent_channels = vae_decoder.config.latent_channels
vae_out_channels = vae_decoder.config.out_channels
print("# VAE DECODER")
print(f"latent_sample: [{1}, {vae_latent_channels}, {unet_sample_size}, {unet_sample_size}]")

# SAFETY CHECKER
safety_checker = pipeline.safety_checker
clip_num_channels = safety_checker.config.vision_config.num_channels
clip_image_size = safety_checker.config.vision_config.image_size
print("# SAFETY CHECKER")
print(f"clip_input: [{1}, {clip_num_channels}, {clip_image_size}, {clip_image_size}]")
print(f"images: [{1}, {vae_sample_size}, {vae_sample_size}, {vae_out_channels}]")
```

步骤3 执行以下命令获取shape信息。

```
python parse_models_shape.py
```

可以看到获取的shape信息如下图所示。

图 6-74 shape 信息



----结束

PyTorch 模型转换为 Onnx 模型（可选）

获取onnx模型有两种方式，方式一是使用官方提供的模型转换脚本将pytorch模型转换为onnx模型，方式二是对于提供了onnx模型的仓库，可以直接下载onnx模型。下面介绍方式一如何操作，如果采用方式二，可以跳过此步骤。

步骤1 通过git下载diffusers对应版本的源码。

```
git clone https://github.com/huggingface/diffusers.git -b v0.11.1
```

步骤2 在diffusers的script/convert_stable_diffusion_checkpoint_to_onnx.py脚本中，可以通过执行以下命令生成onnx模型，其中model_path指定pytorch的模型根目录，output_path指定生成的onnx模型目录。

```
cd /home_host/work
python diffusers/scripts/convert_stable_diffusion_checkpoint_to_onnx.py --model_path "./runwayml/
pytorch_models"  --output_path "./pytorch_to_onnx_models"
```

----结束

静态 shape 模型转换

转换静态shape模型需要在模型转换阶段固定模型的输入shape，也就是说每个输入shape是唯一的。静态shape转换主要包括两种场景：

- 第一种是待转换onnx模型的输入本身已经是静态shape，此时不需要在转换时指定输入shape也能够正常转换为和onnx模型输入shape一致的mindir模型。
- 第二种是待转换onnx模型的输入是动态shape（导出onnx模型时指定了dynamic_axes参数），此时需要在转换时明确指定输入的shape。

转换时指定输入shape可以在命令行中指定，也可以通过配置文件的形式进行指定。

- 在命令行中指定输入shape。

  命令行可以直接通过--inputShape参数指定输入的shape，格式为“input_name:input_shape”，如果有多个输入，需要使用“;”隔开，比如“input1_name:input1_shape;input2_name:input2_shape”。

  ```
  converter_lite --modelFile=./text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR --
  optimize=ascend_oriented --outputFile=./text_encoder --inputShape="input_ids:1,77"
  ```

- 在配置文件中指定输入shape。

  配置文件中通过“[ascend_context]”配置项指定input_shape，格式与命令行一致，多个输入，需要使用“;”隔开；然后在命令行中通过--configFile指定对应的配置文件路径即可。

  ```
  # text_encoder.ini
  ```

```
[ascend_context]
input_shape=input_ids:[1,77]
```

转换命令如下：

```
converter_lite --modelFile=./text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR --
optimize=ascend_oriented --outputFile=./text_encoder --configFile=./text_encoder.ini
```

在使用converter_lite工具转换时，默认是将所有算子的精度转换为fp16，如果想要将固定shape的模型精度修改为fp32进行转换，需要在配置文件中指定算子的精度模式为precision_mode，配置文件的写法如下（更多精度模式请参考**precision_mode**）：

```
# text_encoder.ini
[ascend_context]
input_shape=input_ids:[1,77]
precision_mode=enforce_fp32
```

对于本次AIGC迁移，为了方便对多个模型进行转换，可以通过批量模型转换脚本自动完成所有模型的转换。

步骤1 执行以下命令创建并进入static_shape_convert目录。

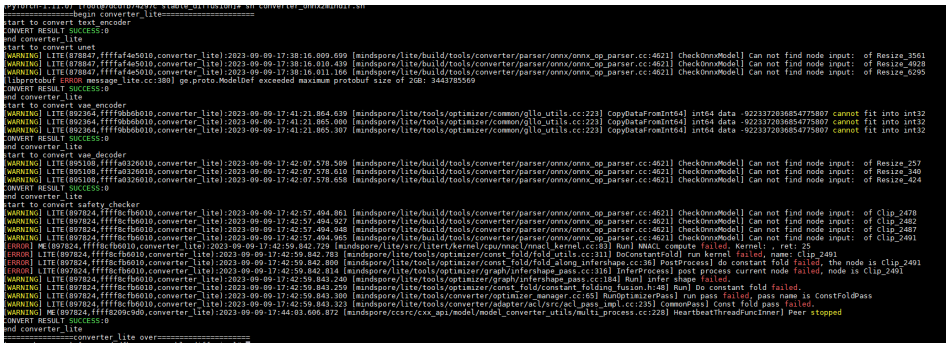
```
mkdir -p /home_host/work/static_shape_convert
cd /home_host/work/static_shape_convert
```

步骤2 在static_shape_convert目录下新建converter_onnx2mindir.sh文件并复制下面内容。其中，onnx_dir表示onnx模型的目录，mindir_dir指定要生成的mindir模型的保存目录。

```
# converter_onnx2mindir.sh
# 设置onnx模型和mindir模型目录
onnx_dir=/home_host/work/runwayml/onnx_models
mindir_dir=./mindir_models

# 指定配置文件路径
config_dir=/home_host/work/modelarts-ascend/examples/AIGC/stable_diffusion/configs

echo "================begin converter_lite====================="

sub_cmd='--fmk=ONNX --optimize=ascend_oriented --saveType=MINDIR'
mkdir -p $mindir_dir
# rm缓存,慎改
atc_data_dir=/root/atc_data/
# 通用转换方法
common_converter_model() {
    model_name=$1
    echo "start to convert $model_name"
    rm -rf $atc_data_dir
    converter_lite --modelFile="$onnx_dir/$model_name/model.onnx" \
                --outputFile="$mindir_dir/$model_name" \
                --configFile="$config_dir/$model_name.ini" \
                $sub_cmd
    printf "end converter_lite\n"
}
common_converter_model "text_encoder"
common_converter_model "unet"
common_converter_model "vae_encoder"
common_converter_model "vae_decoder"
common_converter_model "safety_checker"

echo "================converter_lite over====================="
```

转换结果如下，其中safety_checker模型转换成功了，但中间有ERROR日志，该ERROR属于常量折叠失败，不影响结果。

图 6-75 转换结果



----结束

动态分档模型转换（可选）

说明

如果迁移的模型有多个shape档位的需求，可以通过如下方式对模型进行分档转换。

动态分档是指将模型输入的某一维或者某几维设置为“动态”可变，但是需要提前设置可变维度的“档位”范围。即转换得到的模型能够在指定的动态轴上使用预设的几种shape（保证模型支持的shape），相比于静态shape更加灵活，且性能不会有劣化。

动态分档模型转换需要使用配置文件，指定输入格式为“ND”，并在config文件中配置ge.dynamicDims和input_shape使用，在input_shape中将输入shape的动态维度设为-1，并在ge.dynamicDims中指定动态维度的档位，更多配置项可以参考**官方文档**。

- 如果网络模型只有一个输入：每个档位的dim值与input_shape参数中的-1标识的参数依次对应，input_shape参数中有几个-1，则每档必须设置几个维度。

  以text_encoder模型为例，修改配置文件text_encoder.ini如下所示：

  ```
  # text_encoder.ini

  [acl_build_options]
  input_format="ND"
  input_shape="input_ids:1,-1"
  ge.dynamicDims="77;33"
  ```

  使用上述配置文件转换得到的模型，支持的输入shape为(1,77)和(1,33)。

  然后使用converter lite执行模型转换，转换命令如下：

  ```
  converter_lite --modelFile=./onnx_models/text_encoder/model.onnx --fmk=ONNX --saveType=MINDIR
  --optimize=ascend_oriented --outputFile=./mindirs --configFile=./configs/text_encoder.ini
  ```

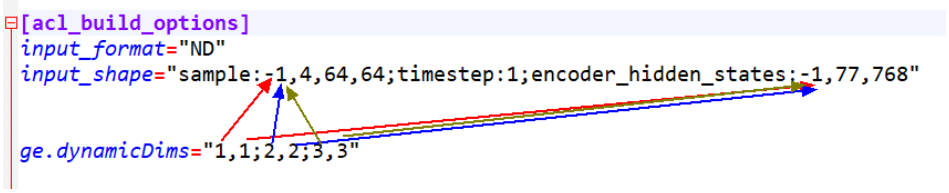
- 如果网络模型有多个输入：档位的dim值与网络模型输入参数中的-1标识的参数依次对应，网络模型输入参数中有几个-1，则每档必须设置几个维度。

  以unet模型为例，该网络模型有三个输入，分别为“sample(1,4,64,64)”、“timestep(1)”、“encoder_hidden_states(1,77,768)”，修改unet.ini配置文件如下所示：

  ```
  # unet.ini

  [acl_build_options]
  input_format="ND"
  input_shape="sample:-1,4,64,64;timestep:1;encoder_hidden_states:-1,77,768"
  ge.dynamicDims="1,1;2,2;3,3"
  ```

  转换得到的模型支持的输入dims组合档数分别为：

图 6-76 组合档数



第0档：sample(1,4,64,64) + timestep(1) + encoder_hidden_states(1,77,768)

第1档：sample(2,4,64,64) + timestep(1) + encoder_hidden_states(2,77,768)

第2档：sample(3,4,64,64) + timestep(1) + encoder_hidden_states(3,77,768)

然后使用converter lite执行模型转换，转换命令如下：

```
converter_lite --modelFile=./onnx_models/unet/model.onnx --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented --outputFile=./mindirs --configFile=./configs/unet.ini
```

说明

- 最多支持100档配置，每一档通过英文逗号分隔。
- 如果用户设置的dim数值过大或档位过多，可能会导致模型编译失败，此时建议用户减少档位或调低档位数值。
- 如果用户设置了动态维度，实际推理时，使用的输入数据的shape需要与设置的档位相匹配。

6.4.4.2 pipeline 代码适配

onnx pipeline的主要作用是将onnx模型进行一系列编排，并在onnx Runtime上按照编排顺序执行。因此，需要将转换得到的mindir模型按照相同的逻辑进行编排，并在MindSpore Lite上执行。只需要将原始onnx的pipeline中涉及到onnx模型初始化及推理的接口替换为MindSpore Lite的接口即可。

MindSpore Lite提供了Python、C++以及JAVA三种应用开发接口，此处以Python接口为例，介绍如何使用MindSpore Lite Python API构建并推理Stable Diffusion模型，更多信息请参考**MindSpore Lite应用开发**。

以官方onnx pipeline代码为例，其提供的onnx pipeline代码路径在“**${diffusers}/pipelines/stable_diffusion/pipeline_onnx_stable_diffusion_img2img.py**”，其中${diffuers}表示diffusers包的安装路径，可以通过pip进行查看。

```
# shell
pip show diffusers
```

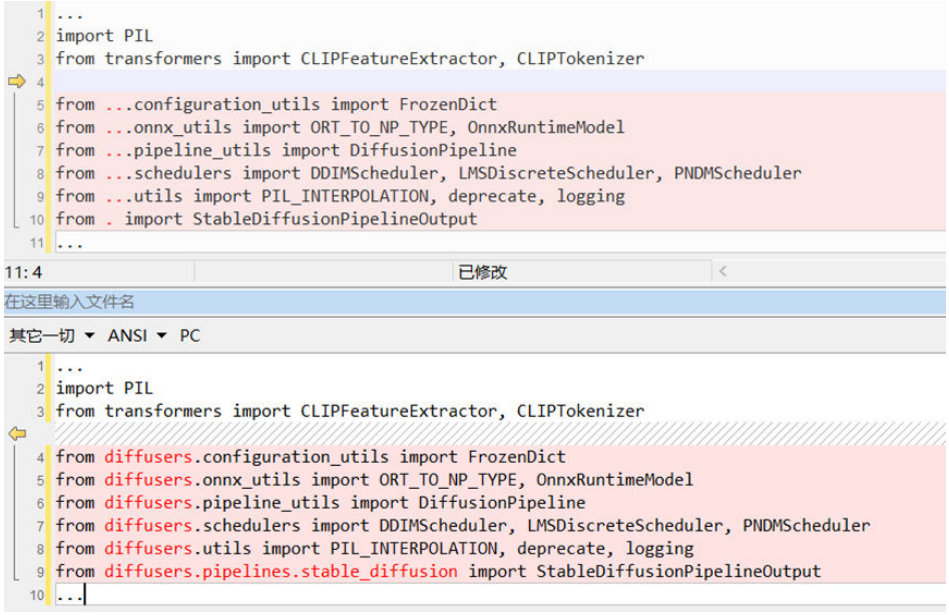
修改代码依赖

新建并进入/home_host/work/pipeline目录。

```
mkdir -p /home_host/work/pipeline
cd /home_host/work/pipeline
```

将onnx pipeline依赖的图生图源码“pipeline_onnx_stable_diffusion_img2img.py”复制到该目录下，名称改为“pipeline_onnx_stable_diffusion_img2img_mslite.py”，以便与源文件名称区分。但是这样也会导致无法正确找到源码中相对路径下的依赖，需要将对于diffusers包内的相对路径修改为绝对路径的形式。

图 6-77 代码依赖修改前与修改后



将推理代码“modelarts-ascend/examples/AIGC/stable_diffusion/onnx_pipeline.py”也复制一份到该目录，名称改为“mslite_pipeline.py”，迁移后的推理代码中的pipeline需要修改为从复制的onnx pipeline文件导入：

```
# onnx_pipeline.py
from pipeline_onnx_stable_diffusion_img2img_mslite import OnnxStableDiffusionImg2ImgPipeline
```

模型初始化

使用MindSpore Lite进行推理时一般需要先设置目标设备的上下文信息，然后构建推理模型，获取输入数据，模型预测并得到最终的结果。一个基础的推理框架写法如下所示：

```
# base_mslite_demo.py
import mindspore_lite as mslite

# 设置目标设备上下文为Ascend，指定device_id为0
context = mslite.Context()
context.target = ["ascend"]
context.ascend.device_id = 0
# 构建模型
model = mslite.Model()
model.build_from_file("./model.mindir", mslite.ModelType.MINDIR, context)

# 输入数据到Device侧，针对于多输入场景可以通过list来指定输入
in_data = [np.array(data1), np.array(data2)]
inputs = model.get_inputs()
for i, _inputs in enumerate(inputs):
    _input.set_data_from_numpy(in_data[i])
# 前向推理，并将结果从device侧传到host侧
outputs = model.predict(inputs)
outputs = [output.get_data_to_numpy() for output in outputs]
# 后处理...
```

为了同时兼容onnx模型和mindir模型都能够在适配后的pipeline中运行，需要对于Model进行封装，MsliteModel各参数模型说明已给出，根据模型初始化参数设置当前模型使用onnx模型（运行在CPU上）或mindir模型（运行在昇腾设备上），也能够方便进行精度的校验。

```
# mslite_model_proxy.py
import os
import mindspore_lite as mslite
class MsliteModel:
    def __init__(self, model_path, model_name='ms model', device_type='ascend', use_ascend=True,
                 onnx_runtime_model=None, get_shape=False, resize_shape=False) -> None:
        """
        mindir模型代理类
        Args:
            model_path: mindir文件路径
            model_name: 模型名称
            device_type: 设备类型
            use_ascend: 是否使用Ascend
            onnx_runtime_model: onnx模型对象
            get_shape: 是否获取模型shape信息、输入数据shape信息
            resize_shape: resize shape开关，分档模型需开启
        """
        print('model_path:{}'.format(model_path))
        self.model_name = model_name
        self.context = MsliteModel.init_context(device_type)
        self.model = mslite.Model()
        self.model.build_from_file(model_path, mslite.ModelType.MINDIR, self.context)
        self.ms_inputs = self.model.get_inputs()
        self.use_ascend = use_ascend
        self.onnx_runtime_model = onnx_runtime_model
        self.get_shape = get_shape
        self.resize_shape = resize_shape
    @staticmethod
    def init_context(device_type='ascend'):
        context = mslite.Context()
        context.target = [device_type]
        context.ascend.device_id = int(os.getenv('DEVICE_ID') or 0)
        context.cpu.thread_num = 1 if device_type == 'ascend' else 32
        context.cpu.thread_affinity_mode = 2
        return context
    def __call__(self, **kwargs):
        if not self.use_ascend:
            return self.onnx_runtime_model(**kwargs)
        inputs = list(kwargs.values())
        if len(inputs) <= 0:
            raise Exception('get tensor input info failed')
        ms_input = self.model.get_inputs()
        if self.get_shape:
            print(f'{self.model_name} shape info:')
            for index, val in enumerate(self.model.get_inputs()):
                print(f'{self.model_name}: param{index} shape -> {val.shape}')
            shapes = [list(input.shape) for input in inputs]
            print(f"inputs: input_shape -> {shapes}")
        if self.resize_shape:
            self.model.resize(ms_input, [list(input.shape) for input in inputs])
        for index, val in enumerate(ms_input):
            val.set_data_from_numpy(inputs[index])
        outputs = self.model.predict(ms_input)
        outputs = [output.get_data_to_numpy() for output in outputs]
        return outputs
```

适配MindSpore Lite Runtime到onnx pipeline，首先需要初始化MindSpore LiteModel对象，通过在OnnxStableDiffusionImg2ImgPipeline中增加mindir模型初始化函数，然后在pipeline类的__init__方法调用该函数，在pipeline初始化的时候直接初始化模型。可以参照如下样例，可以通过修改use_ascend去修改该模型是否使用mindir运行，也可以编写代码通过环境变量指定。

```
# pipeline_onnx_stable_diffusion_img2img_mslite.py

class OnnxStableDiffusionImg2ImgPipeline(DiffusionPipeline):
    ...
    def mslite_modules_init(self):
        self.text_encoder_ms = MsliteModel(
            model_path=os.environ['TEXT_ENCODER_PATH'],
```

```
            model_name='text_encoder', use_ascend=True,
            onnx_runtime_model=self.text_encoder)
        self.vae_encoder_ms = MsliteModel(
            model_path=os.environ['VAE_ENCODER_PATH'],
            model_name='vae_encoder', use_ascend=True,
            onnx_runtime_model=self.vae_encoder)
        self.unet_ms = MsliteModel(model_path=os.environ['UNET_PATH'],
                                   model_name='unet', use_ascend=True,
                                   onnx_runtime_model=self.unet)
        self.vae_decoder_ms = MsliteModel(model_path=os.environ['VAE_DECODER_PATH'],
                                          model_name='vae_decoder', use_ascend=True,
                                          onnx_runtime_model=self.vae_decoder)
        self.safety_checker_ms = MsliteModel(model_path=os.environ['SAFETY_CHECKER_PATH'],
                                             model_name='safety_checker', use_ascend=True,
                                             onnx_runtime_model=self.safety_checker)
    ...
```

模型推理适配

完成模型初始化后，需要将onnx模型推理的代码等价替换为对应的mindir模型推理接口。以vae_encoder模型为例，在pipeline代码中查找vae_encoder推理调用的地方，然后修改为对应的MindSpore Lite版本的推理接口模型。

- 使用MindSpore Lite Runtime接口替换onnx Runtime接口

  ```
  # pipeline_onnx_stable_diffusion_img2img_mslite.py
  ...
  # onnx模型
  # init_latents = self.vae_encoder(sample=image)[0]
  # ----------------修改点-----------------
  # mslite模型
  init_latents = self.vae_encoder_ms(sample=image)[0]
  ...
  ```

- 替换内嵌模型

  ```
  # pipeline_onnx_stable_diffusion_img2img_mslite.py
  ...
  # onnx模型
  # image = np.concatenate([self.vae_decoder(latent_sample=latents[i : i + 1])[0] for i in
  range(latents.shape[0])])
  # ----------------修改点-----------------
  # mslite模型
  image = np.concatenate([self.vae_decoder_ms(latent_sample=latents[i : i + 1])[0] for i in
  range(latents.shape[0])])
  ...
  ```

修改后的文件参考**Gitee代码库**中的如下两个文件：

- pipeline_onnx_stable_diffusion_img2img_mslite.py
- mslite_model_proxy.py

运行 pipeline 代码

pipeline代码如下：

```
# mslite_pipeline.py
import os

import requests
import torch
import numpy as np
from PIL import Image
from io import BytesIO

from pipeline_onnx_stable_diffusion_img2img_mslite import OnnxStableDiffusionImg2ImgPipeline

def setup_seed(seed):
```

```
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    torch.backends.cudnn.deterministic = True

setup_seed(0)

# 指定mindir和onnx模型路径
mindir_dir = "/home_host/work/static_shape_convert/mindir_models"
onnx_model_path = "/home_host/work/runwayml/onnx_models"

os.environ['DEVICE_ID'] = "0"
os.environ['TEXT_ENCODER_PATH'] = f"{mindir_dir}/text_encoder.mindir"
os.environ['VAE_ENCODER_PATH'] = f"{mindir_dir}/vae_encoder.mindir"
os.environ['UNET_PATH'] = f"{mindir_dir}/unet_graph.mindir"
os.environ['VAE_DECODER_PATH'] = f"{mindir_dir}/vae_decoder.mindir"
os.environ['SAFETY_CHECKER_PATH'] = f"{mindir_dir}/safety_checker.mindir"
pipe = OnnxStableDiffusionImg2ImgPipeline.from_pretrained(onnx_model_path,
      torch_dtype=torch.float32).to("cpu")
url = "https://raw.githubusercontent.com/CompVis/stable-diffusion/main/assets/stable-samples/img2img/
sketch-mountains-input.jpg"
response = requests.get(url, verify=False)
init_image = Image.open(BytesIO(response.content)).convert("RGB")
init_image = init_image.resize((512, 512))

prompt = "A fantasy landscape, trending on artstation"
images = pipe(prompt=prompt, image=init_image, strength=0.75, guidance_scale=7.5).images
images[0].save("fantasy_landscape_npu.png")
```

在运行pipeline时，默认的加速卡为0号卡，当机器有多人使用时，可能存在资源占用而无法正常运行的情况，可以通过环境变量指定加速卡ID，如指定5号卡进行执行。

```
# mslite_pipeline.py
...
os.environ['DEVICE_ID'] = "5"
...
```

最后执行python脚本进行推理：

```
#shell
python mslite_pipeline.py
```

图 6-78 执行推理脚本

图 6-79 MindSpore Lite pipeline 输出的结果图片

6.4.5 迁移效果校验

在pipeline适配完成后，需要验证适配后的效果是否满足要求，通过对比原始onnx pipeline的最终输出结果确认迁移效果。如果精度和性能都没有问题，则代表迁移完成。

对比图片生成效果

在CPU上推理onnx，将原始onnx和适配完成的MindSpore Lite pipeline输出的结果图片进行对比，在这里保证输入图片及文本提示词一致。如果差异较为明显可以进行**模型精度调优**。

确认性能是否满足要求

在推理代码开始结尾处加入时间记录，并打印出推理执行耗时。根据用户需求判断性能是否满足要求，如果不满足可以进行**性能调优**。

```
import time
start_time = time.time()
# 推理代码
end_time = time.time()
print(f"infer time {end_time - start_time}")
```

6.4.6 模型精度调优

6.4.6.1 场景介绍

本小节通过一个具体问题案例，介绍模型精度调优的过程。

如下图所示，使用MindSpore Lite生成的图像和onnx模型的输出结果有明显的差异，因此需要对MindSpore Lite pipeline进行精度诊断。

图 6-80 结果对比

onnx pipeline　　MindSpore Lite pipeline

说明

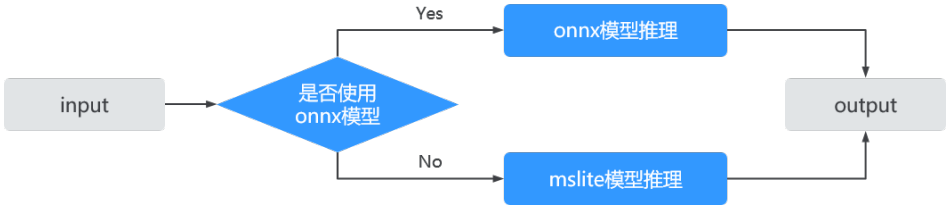
在MindSpore Lite 2.0.0版本中，Stable Diffusion的五个模型的精度都能够保证一致性，但是在最新的2.1.0版本中，会出现text_encoder模型精度不一致的情况。该问题后续会发布补丁进行修复。

6.4.6.2 精度问题诊断

逐个替换模型，检测有问题的模型

该方式主要是通过模型替换，先定位出具体哪个模型引入的误差，进一步诊断具体的模型中哪个算子或者操作导致效果问题，模型替换原理如下图所示。通过设置开关选项（是否使用onnx模型），控制模型推理时，模型使用的是onnx模型或是mindir的模型。

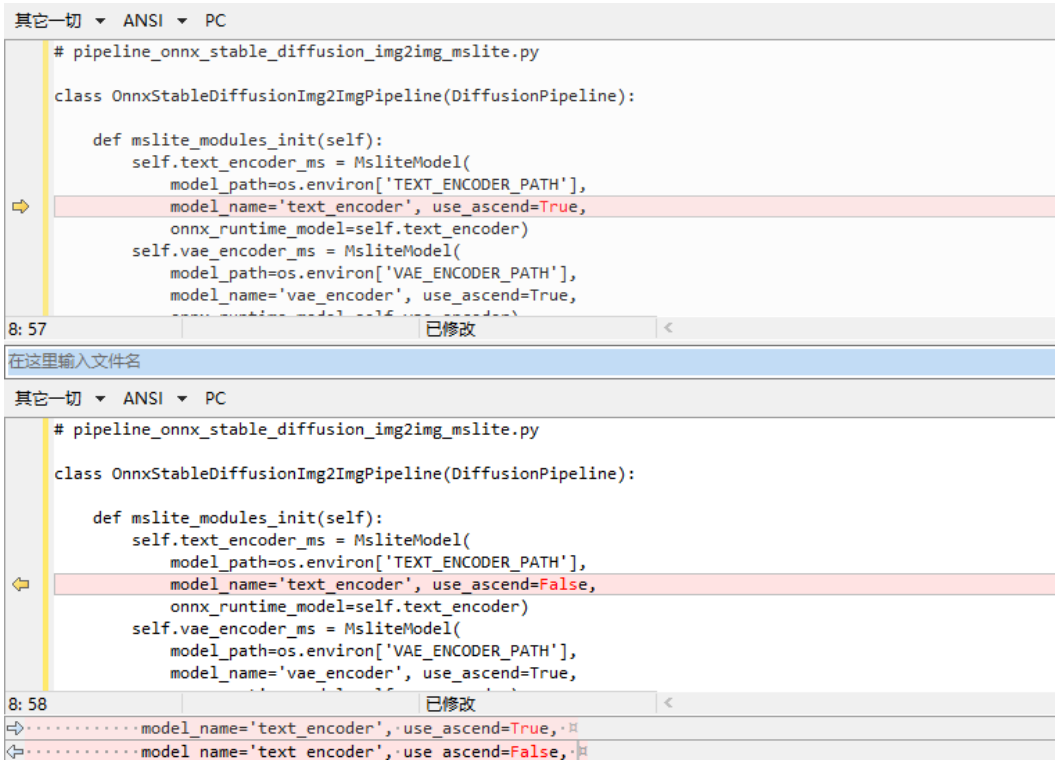
图 6-81 精度诊断流程



一般情况下，onnx模型推理的结果可以认为是标杆数据，单独替换某个onnx模型为MindSpore Lite模型，运行得到的结果再与标杆数据做对比，如果没有差异则说明pipeline的差异不是由当前替换的MindSpore Lite模型引入。

如果有差异，则说明当前模型与原始onnx的结果存在差异。依次单独替换onnx模型为对应的MindSpore Lite模型，从而定位出有差异的模型。在模型初始化的代码块已经添加了use_ascend参数，修改参考如下：

图 6-82 代码修改



以上述现象为例，通过修改use_ascend参数值对模型替换，可以发现：当text_encoder模型为onnx模型，其余模型为mindir模型时，能够得到和标杆数据相同的输出，因此可以判断出转换得到的text_encoder模型是产生pipeline精度误差的根因。通过下一小节可以进一步确认模型精度的差异。

6.4.6.3 精度问题处理

设置高精度并重新转换模型

在转换模型时，默认采用的精度模式是fp16，如果转换得到的模型和标杆数据的精度差异比较大，可以使用fp32精度模式提升模型的精度（精度模式并不总是需要使用fp32，因为相对于fp16，fp32的性能较差。因此，通常只在检测到某个模型精度存在问题时，才会考虑是否使用fp32进行尝试）。使用fp32精度模式的配置文件如下：

配置文件：

```
# config.ini
[ascend_context]
precision_mode=enforce_fp32 #使用 fp32
```

其他方式

需要实际分析算子层面的差异，需要联系华为工程师进行具体分析。

6.4.7 性能调优

6.4.7.1 单模型性能测试工具 Mindspore lite benchmark

在模型精度对齐后，针对Stable Diffusion模型性能调优，可以通过AOE工具进行自助性能调优，进一步可以通过profiling工具对于性能瓶颈进行分析，并针对性的做一些调优操作。

可以直接使用benchmark命令测试mindir模型性能，用来对比调优前后性能是否有所提升。

```
# shell
cd /home_host/work
benchmark --modelFile=diffusers/scripts/mindir_models/text_encoder.mindir --device=Ascend
```

上述命令中：modelFile指定生成的mindir模型文件；device指定运行推理的设备。其他用法参考**benchmark文档**。

测试结果如下所示：

图 6-83 测试结果

```
[root@6861cf0c12ba work]# benchmark --modelFile=diffusers/scripts/mindir_models/text_encoder.mindir --device=Ascend
ModelPath = diffusers/scripts/mindir_models/text_encoder.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 10
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 3
NumThreads = 2
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 2357.9 ms
Running warm up loops...
Running benchmark loops...
Model = text_encoder.mindir, NumThreads = 2, MinRunTime = 3.974000 ms, MaxRuntime = 3.995000 ms, AvgRunTime = 3.982000 ms
Run Benchmark text_encoder.mindir Success.
```

6.4.7.2 单模型性能调优 AOE

使用**AOE**工具可以在模型转换阶段对于模型运行和后端编译过程进行执行调优，注意AOE只适合静态shape的模型调优。在AOE调优时，容易受当前缓存的一些影响，建议分两次进行操作，以达到较好的优化效果（第一次执行生成AOE的知识库，在第二次使用时可以复用）。在该场景中，AOE对text_encoder等模型提升效果不大，性能主要瓶颈点在unet模型中，主要对unet模型做调优，整体的操作步骤如下：

步骤1 转换前先清理缓存，避免转换时的影响。

```
#shell
# 删除已有的aoe知识库，或者备份一份
rm -rf /root/Ascend/latest/data/aoe
# 删除编译缓存
rm -rf /root/atc_data/*
```

步骤2 新建并进入AOE工作目录。

```
mkdir -p /home_host/work/aoe
cd /home_host/work/aoe
```

步骤3 在配置文件中启用AOE自动调优。

配置unet.ini，开启aoe调优（ aoe_mode + op_select_impl_mode ）。

```
#unet.ini
[ascend_context]
input_shape=sample:[2,4,64,64];timestep:[1];encoder_hidden_states:[2,77,768]
input_format=NCHW

aoe_mode="subgraph tuning, operator tuning"
op_select_impl_mode=high_performance
```

配置打印ASCEND日志，其中ASCEND_GLOBAL_LOG_LEVEL的值对应的日志级别分别为：0-debug、1-info、2-warning、3-error。

```
#shell
export ASCEND_GLOBAL_LOG_LEVEL=1
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

模型转换时指定AOE调优配置文件。

```
#shell
# 模型转换时指定AOE调优配置文件并将调优日志输出到aoe_unet.log
mkdir aoe_output
converter_lite --modelFile=/home_host/work/runwayml/onnx_models/unet/model.onnx --outputFile=./
aoe_output/aoe_unet --configFile=unet.ini  --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented >
aoe_unet.log
```

启动AOE调优后，模型转换时长会延长到数小时，因为其中包含了AOE的转化过程耗时较长，也可以指定调优时间，一般情况下时间越长效果会越好，一般10h以内即可，推荐在后台执行。调优完成后，默认将AOE生成的知识库保存在“/root/Ascend/latest/data/aoe”路径下，同时会在aoe_output路径下输出对应的mindir模型，由于当前模型并没有吸收知识库信息，所以性能不佳，因此需要在保留AOE知识库的情况下，再次进行转换，以达到较优性能。

步骤4 删除编译缓存atc_data。

说明

注意相比第一次清除缓存操作，本次保留了AOE知识库。

```
#shell
# 删除编译缓存
rm -rf /root/atc_data/*
```

步骤5 再次执行模型转换命令，确保AOE能够命中知识库。

配置config.ini，关闭AOE调优：

```
# unet.ini

[ascend_context]
input_shape=sample:[2,4,64,64];timestep:[1];encoder_hidden_states:[2,77,768]
input_format=NCHW
```

再次执行模型转换命令（此次运行关闭了AOE，速度会变快）：

```
#shell
converter_lite --modelFile=/home_host/work/runwayml/onnx_models/unet/model.onnx --outputFile=./
aoe_output/aoe_unet --configFile=unet.ini  --fmk=ONNX --saveType=MINDIR --optimize=ascend_oriented >
aoe_unet2.log
```

此时，aoe_output下面会有对应的mindir模型，包含了AOE知识库信息。使用benchmark工具测试新生成的mindir模型性能，同aoe调优前的模型进行对比，可以看到模型性能有所提升。

```
#shell
调优前命令如下：
benchmark --modelFile=/home_host/work/static_shape_convert/mindir_models/unet_graph.mindir --
device=Ascend --numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --
loopCount=100
```


调优后命令如下：
benchmark --modelFile=/home_host/work/aoe/aoe_output/aoe_unet_graph.mindir --device=Ascend --numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --loopCount=100

图 6-84 调优前模型

```
[root@6861cf0c12ba aoe]# benchmark --modelFile=/home_host/work/static_shape_convert/mindir_models/unet_graph.mindir --device=Ascend --numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --loopCount=100
ModelPath = /home_host/work/static_shape_convert/mindir_models/unet_graph.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 100
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 100
NumThreads = 1
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 7786.27 ms
Running warm up loops...
Running benchmark loops...
Model = unet_graph.mindir, NumThreads = 1, MinRunTime = 72.867996 ms, MaxRuntime = 77.177002 ms, AvgRunTime = 74.184998 ms
Run Benchmark unet_graph.mindir Success.
```

图 6-85 调优后模型

```
[root@6861cf0c12ba aoe]# benchmark --modelFile=/home_host/work/aoe/aoe_output/aoe_unet_graph.mindir --device=Ascend --numThreads=1 --parallelNum=1 --workersNum=1 --warmUpLoopCount=100 --loopCount=100
ModelPath = /home_host/work/aoe/aoe_output/aoe_unet_graph.mindir
ModelType = MindIR
InDataPath =
GroupInfoFile =
ConfigFilePath =
InDataType = bin
LoopCount = 100
DeviceType = Ascend
AccuracyThreshold = 0.5
CosineDistanceThreshold = -1.1
WarmUpLoopCount = 100
NumThreads = 1
InterOpParallelNum = 1
Fp16Priority = 0
EnableParallel = 0
calibDataPath =
EnableGLTexture = 0
cpuBindMode = HIGHER_CPU
CalibDataType = FLOAT
start unified benchmark run
PrepareTime = 5907.39 ms
Running warm up loops...
Running benchmark loops...
Model = aoe_unet_graph.mindir, NumThreads = 1, MinRunTime = 44.772999 ms, MaxRuntime = 46.356998 ms, AvgRunTime = 45.570999 ms
Run Benchmark aoe_unet_graph.mindir Success.
```

AOE优化成功的mindir已经融合了优化的知识库，是一个独立可用的模型。即使AOE知识库删除，不影响该mindir的性能。可以备份这个模型优化产生的知识库，以后需要的话再使用。

----结束

6.4.8 常见问题

6.4.8.1 模型转换失败怎么办?

常见的模型转换失败原因可以通过查询转换失败错误码来确认具体导失败的原因，Stable Diffusion新推出的模型在转换中可能会遇到算子不支持的问题，可以到华为云管理页面上提交工单来寻求帮助。

6.4.8.2 图片大 Shape 性能劣化严重怎么办?

在昇腾设备上，可能由于GPU内存墙导致在大shape下遇到性能问题，MindSporeLite提供了Flash Attention编译优化机制，可以考虑升级最新版本的MidnSporeLite Convertor来进行编译期的算子优化，在大Shape场景下会有明显的改善。

6.4.8.3 同样功能的 PyTorch Pipeline，因为指导要求适配 onnx pipeline，两个 pipeline 本身功能就有差别，如何适配?

由于Diffusers社区的“single model file policy”设计原则，不同的pipeline是不同路径在独立演进的。先确保应用输出符合预期后，再进入到MindSpore Lite模型转换的过程，否则迁移昇腾后还是会遇到同样的问题。

6.4.8.4 AOE 的自动性能调优使用上完全没有效果怎么办?

在MindSpore Lite Convertor2.1版本之前可能出现的调优不生效的场景，建议直接使用MindSpore Lite Convertor2.1及以后的版本。配置文件指定选项进行AOE调优。使用转换工具配置config参数，具体如下所示，其中“subgraph tuning”表示子图调优，“operator tuning”表示算子调优。

其中，“ge.op_compiler_cache_mode”在该场景下必须设置为“force”，表示该场景下要强制刷新缓存，保证AOE调优后的知识库能够命中，实现模型调优。示例如下：

```
# config.ini

[ascend_context]
aoe_mode="subgraph tuning, operator tuning"
[acl_init_options]
ge.op_compiler_cache_mode="force"
```

6.4.8.5 迁移后应用出图效果相比 GPU 无法对齐怎么办

扩散模型在噪音和随机数上的生成，本身就有一定的随机性，GPU和NPU（Ascend）硬件由于存在一定细小的差别，很难确保完全一致，较难达成生成图片100%匹配，建议通过盲测的方式对效果进行验证。

6.4.8.6 模型精度有问题怎么办?

首先考虑通过FP16的方式进行转换和执行，再通过精度诊断工具来进行分析，更进一步可以到华为云官网上提交工单处理。

6.4.8.7 模型转换失败时如何查看日志和定位原因?

在模型转换的过程，如果出现模型转换失败，可以参考以下步骤查看日志并定位原因：

步骤1 设置DEBUG日志。

设置MindSpore日志环境变量。

```
#shell
export GLOG_v=0       # 0-DEBUG、1-INFO、2-WARNING、3-ERROR
```

设置CANN日志环境变量。

```
#shell
export ASCEND_GLOBAL_LOG_LEVEL=1          # 0：表示DEBUG、1：表示INFO、2：表示WARNING、3：表示ERROR 4: 表示NONE
export ASCEND_SLOG_PRINT_TO_STDOUT=1       # 表示日志打印
```

步骤2 设置DUMP模型转换中间图。

设置DUMP中间图环境变量。

```
#shell
export DUMP_GE_GRAPH=2              # 1：表示dump图全量内容、 2：表示不dump权重数据的基础图、 3：表示只dump节点关系的精简图
```

```
export DUMP_GRAPH_LEVEL=2          # 1：表示dump图所有图、 2：表示dump除子图外的所有图、 3：表示只dump最后一张图
```

步骤3 问题分析。

配置以上的环境变量之后，再重新转换模型，导出对应的日志和dump图进行分析：

1. 报错日志中搜到“not support onnx data type”，表示MindSpore暂不支持该算子。
2. 报错日志中搜到“Convert graph to om failed”，表示CANN模块进行图编译存在保存，需要结合CANN的报错日志和dump图进行具体分析。

----结束

6.4.8.8 Stable Diffusion WebUI 如何适配?

WebUI一般可以分为前端和后端实现两部分，后端的实现模式种类多样，并且依赖了多个的第三方库，当前在WebUI适配时，并没有特别好的方式。在对后端实现比较理解的情况下，建议针对具体的功能进行Diffusers模块的适配与替换，然后针对替换上去的Diffusers，对其pipeline进行昇腾迁移适配，进而替代原有WebUI的功能。针对很多参数以及三方加速库（如xformers）的适配，当前没有特别好的处理方案。

6.4.8.9 LoRA 适配流是怎么样的?

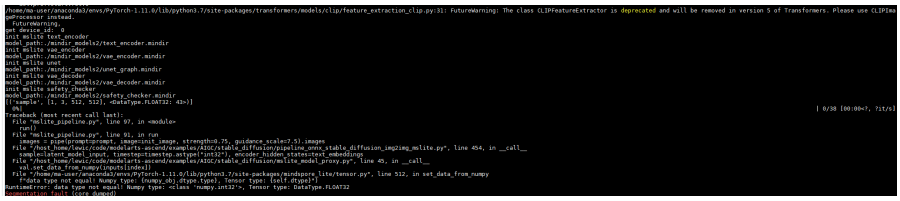
因为现在pytorch-npu推理速度比较慢（固定shape比mindir慢4倍），在现在pth-onnx-mindir的模型转换方式下，暂时只能把lora合并到unet主模型内，在每次加载模型前lora特性就被固定了（无法做到pytorch每次推理都可以动态配置的能力）。

目前临时的静态方案可参考**sd-scripts**， 使用其中的“networks/merge_lora.py”把lora模型合入unet和text-encoder模型。

6.4.8.10 数据类型不匹配问题如何处理?

报错“data type not equal”时，按照堆栈信息，将对应的行数的数据类型修改为匹配的类型。

图 6-86 报错信息



处理该问题时，pipeline_onnx_stable_diffusion_img2img_mslite.py文件的第454行修改如下：

图 6-87 修改内容

```
其它一切  ANSI  PC
            # predict the noise residual
            timestep = np.array([t], dtype=timestep_dtype)
            ################################## unet ################################

            noise_pred = self.unet_ms(
                sample=latent_model_input, timestep=timestep.astype("int32"), encoder_hidden_states=text_embeddings
            )[0]

            # perform guidance
            if do_classifier_free_guidance:
                noise_pred_uncond, noise_pred_text = np.split(noise_pred, 2)
17: 74      默认文本      已修改
在这里输入文件名
其它一切  ANSI  PC
            # predict the noise residual
            timestep = np.array([t], dtype=timestep_dtype)
            ################################## unet ################################

            noise_pred = self.unet_ms(
                sample=latent_model_input, timestep=timestep.astype("float32"), encoder_hidden_states=text_embeddings
            )[0]

            # perform guidance
            if do_classifier_free_guidance:
                noise_pred_uncond, noise_pred_text = np.split(noise_pred, 2)
                noise_pred = noise_pred_uncond + guidance_scale * (noise_pred_text - noise_pred_uncond)

            # compute the previous noisy sample x_t -> x_t-1
            scheduler_output = self.scheduler.step(
                torch.from_numpy(noise_pred), t, torch.from_numpy(latents), **extra_step_kwargs
            )
            latents = scheduler_output.prev_sample.numpy()
```

6.5 GPU 推理业务迁移至昇腾的通用指导

6.5.1 简介

场景介绍

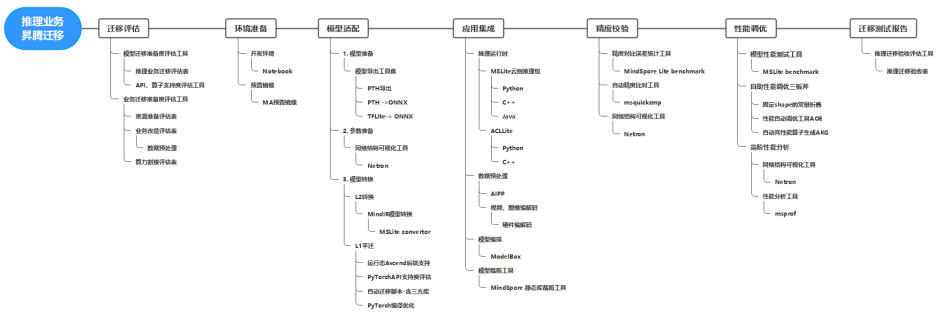
本文旨在指导客户将已有的推理业务迁移到昇腾设备上运行（单机单卡、单机多卡），并获得更好的推理性能收益。

ModelArts针对上述使用场景，在给出系统化推理业务昇腾迁移方案的基础上，提供了即开即用的云上集成开发环境，包含迁移所需要的算力资源和工具链，以及具体的Notebook代码运行示例和最佳实践，并对于实际的操作原理和迁移流程进行说明，包含迁移后的精度和性能验证、调试方法说明。

核心概念

- **推理业务昇腾迁移整体流程及工具链**

图 6-88 推理业务昇腾迁移整体路径



推理业务昇腾迁移整体分为七个大的步骤，并以完整工具链覆盖全链路：

a. 迁移评估：针对迁移可行性、工作量，以及可能的性能收益进行大致的预估。
b. 环境准备：利用ModelArts提供的开发环境一键式准备好迁移、调测需要的运行环境与工具链。
c. 模型适配：针对昇腾迁移模型必要的转换和改造。
    i. 模型准备，导出和保存确定格式的模型。
    ii. 转换参数准备，准备模型业务相关的关键参数。
    iii. 模型转换，包含模型转换、优化和量化等。
d. 应用集成。
    i. 针对转换的模型运行时应用层适配。
    ii. 数据预处理。
    iii. 模型编排。
    iv. 模型裁剪。
e. 精度校验。
    i. 精度对比误差统计工具。
    ii. 自动化精度对比工具。
    iii. 网络结构可视化工具。
f. 性能调优。
    i. 性能测试。
    ii. 性能调优三板斧。
    iii. 性能分析与诊断。
g. 迁移测试报告。
    i. 推理迁移验收表。

- **ModelArts开发环境**

  ModelArts作为华为云上的AI开发平台，提供交互式云上开发环境，包含标准化昇腾算力资源和完整的迁移工具链，帮助用户完成昇腾迁移的调测过程，进一步可在平台上将迁移的模型一键部署成为在线服务向外提供推理服务，或者运行到自己的运行环境中。

- **MindSpore Lite**

  华为自研的AI推理引擎，后端对于昇腾有充分的适配，模型转换后可以在昇腾上获得更好的性能，配合丰富的适配工具链，降低迁移成本，该工具在推理迁移工作的预置镜像已安装，可在镜像中直接使用（见**环境准备**）。关于MindSpore Lite详细介绍可参考**MindSpore Lite文档**。

迁移路线介绍

当前推理迁移时，不同的模型类型可能会采取不同的迁移技术路线。主要分为以下几类：

1. CV类小模型例如yolov5，以及部分AIGC场景的模型迁移，目前推荐使用MindSpore-Lite推理路线，可以利用MindSpore提供的图编译和自动调优能力，达到更好的模型性能。

2. LLM大语言模型场景，在GPU下通常会使用vLLM等大模型推理框架，因此迁移到昇腾时，我们推荐使用PyTorch + ascend-vllm技术路线进行迁移。

如果您使用的模型在上述案例文档中已包含，建议您直接使用案例中迁移好的模型，如果您的模型不在已提供的范围内，或者您因业务要求需要自行完成端到端的迁移，可以参考本迁移指导书介绍的步骤进行操作。

本文的迁移指导及快速入门案例均针对路线1也即MindSpore-Lite迁移路线进行介绍。使用ascend-vllm路线的迁移指导会在后续提供，您可以从上面的案例中下载相关代码并直接参考实现源码。

6.5.2 昇腾迁移快速入门案例

ModelArts提供了两个昇腾迁移案例，方便您快速了解并完成昇腾迁移过程。

约束限制

当前仅贵阳一区域支持选择本案例中的规格及镜像。

操作步骤

步骤1 ModelArts管理控制台左侧导航栏中选择“开发环境 > Notebook”，进入“Notebook”管理页面。

步骤2 单击“创建”，进入“创建Notebook”页面。

图 6-89 实例创建入口

步骤3 请参见如下说明填写参数，并单击“立即创建”。

- 镜像：选择“公共镜像”，选择“mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b”。
- 类型：Ascend。
- 规格：选择snt9b资源。
- 存储配置：云硬盘EVS。
- 磁盘规格：按照对应的存储使用情况可选择存储大小。
- SSH远程开发：如果需通过VS Code远程连接Notebook实例，可打开SSH远程开发，并选择自己的密钥对。

图 6-90 实例创建

步骤4 在Notebook列表，单击“操作列”的“打开”，打开Notebook示例。

图 6-91 运行实例

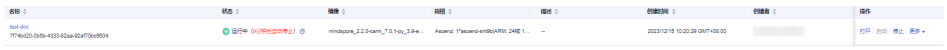
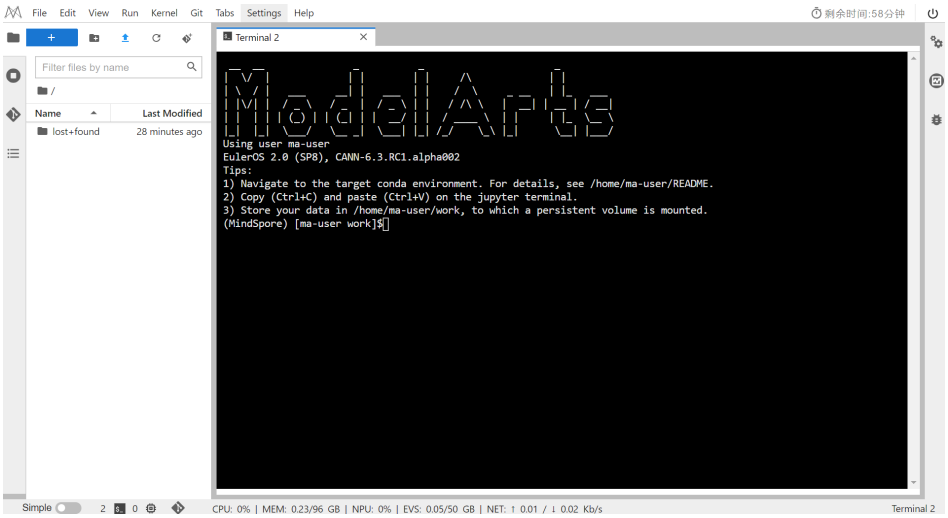


图 6-92 线上 Notebook 入口

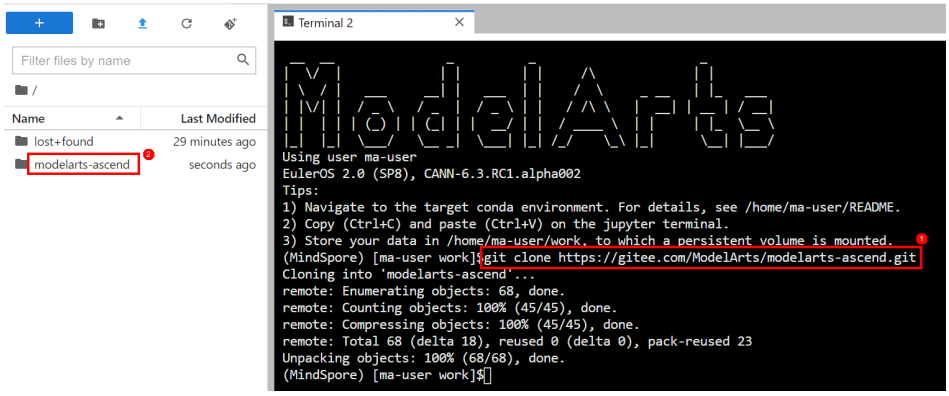


步骤5 克隆ModelArts Ascend代码库。

新建Terminal，执行下述命令将对应的repo克隆到Notebook实例。

```
git clone https://gitee.com/ModelArts/modelarts-ascend.git
```

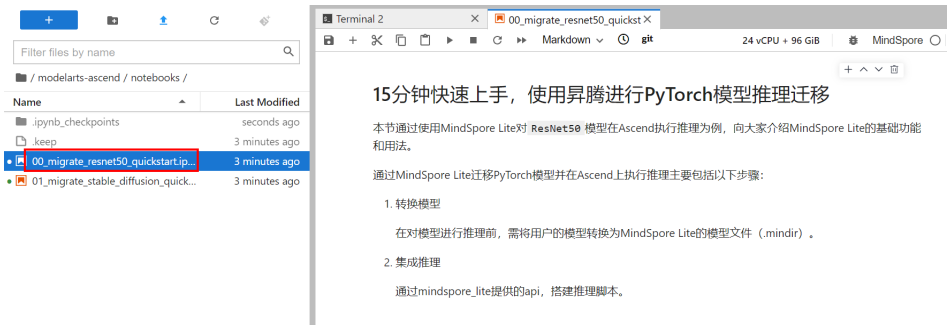
图 6-93 下载示例代码



步骤6 昇腾迁移案例在“~/work/modelarts-ascend/notebooks/”路径下，打开对应的“.ipynb”案例后运行即可。

- **ResNet50模型迁移到Ascend上进行推理**：通过使用MindSpore Lite对ResNet50模型在Ascend执行推理为例，向大家介绍MindSpore Lite的基础功能和用法。

  图 6-94 ResNet50 模型迁移到 Ascend 上进行推理

- **Stable Diffusion模型迁移到Ascend上进行推理**：介绍如何将Stable Diffusion模型通过MSLite进行转换后，迁移在昇腾设备上运行。

  图 6-95 Stable Diffusion 模型迁移到 Ascend 上进行推理

----结束

6.5.3 迁移评估

推理迁移包括模型迁移、业务迁移、精度性能调优等环节，是否能满足最终的迁移效果需要进行系统的评估。如果您仅需要了解迁移过程，可以先按照本文档的指导进行操作并熟悉迁移流程。如果您有实际的项目需要迁移，建议填写附录中的**推理业务迁移评估表**，并将该调研表提供给华为云技术支持人员进行迁移评估，以确保迁移项目能顺利实施。

6.5.4 环境准备

迁移环境简介

ModelArts开发环境针对推理昇腾迁移的场景提供了云上可以直接访问的开发环境，具有如下优点：

- 利用云服务的资源使用便利性，可以直接使用到不同规格的昇腾设备。
- 通过指定对应的运行镜像，可以直接使用预置的、在迁移过程中所需的工具集，且已经适配到最新的版本可以直接使用。
- 开发者可以通过浏览器入口一Notebook方式访问，也可以通过VSCode远程开发的模式直接接入到云上环境中完成迁移开发与调测，最终生成适配昇腾的推理应用。

当前支持以下两种迁移环境搭建方式：

- ModelArts Standard：在Notebook中，使用预置镜像进行。
- ModelArts Lite DevServer：在裸金属服务器中 ，自助配置好存储、安装固件、驱动、配置网络等。

ModelArts Standard

ModelArts上昇腾规格如下。

表 6-8 昇腾规格

规格名称	描述
Ascend 1*ascend-snt9b|ARM 24核 192GB	Snt9b单卡规格，配搭ARM处理器，适合深度学习场景下的模型训练和调测

ModelArts提供了面向推理迁移工作的预置镜像，其中包含了最新商用版驱动、昇腾软件开发库，迁移工具链等。预置镜像可以做到即开即用，用户也可以基于预置镜像构建自定义环境内容。

ModelArts支持的昇腾迁移预置镜像如下：

表 6-9 预置镜像

区域	镜像名称
贵阳一	mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b
贵阳一	mindspore_2.1.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b
贵阳一	pytorch_1.11.0-cann_6.3.2-py_3.7-euler_2.10.7-aarch64-snt9b

可通过如下方式接入Notebook开发环境进行调试。

- JupyterLab：在ModelArts管理控制台，直接打开Notebook示例的方式接入开发环境，详情请见**使用指导**。
- VS Code：利用ModelArts插件，实现VS Code远程连接Notebook示例完成远程开发，详情请见**使用指导**。

下文将介绍如何在ModelArts Standard上使用预置镜像创建Notebook实例。

步骤1 ModelArts管理控制台左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”管理页面。

步骤2 单击“创建”，进入“创建Notebook”页面。

步骤3 请参见如下说明填写参数，并单击“立即创建”。

- 镜像：选择“公共镜像”，选择“mindspore_2.2.0-cann_7.0.1-py_3.9-euler_2.10.7-aarch64-snt9b”。
- 类型：Ascend。
- 规格：选择snt9b资源。
- 存储配置：云硬盘EVS。
- 磁盘规格：按照对应的存储使用情况可选择存储大小。
- SSH远程开发：如果需通过VS Code远程连接Notebook实例，可打开SSH远程开发，并选择自己的密钥对。

图 6-96 实例创建

步骤4 在Notebook列表，单击“操作列”的“打开”，打开Notebook示例。

图 6-97 运行实例

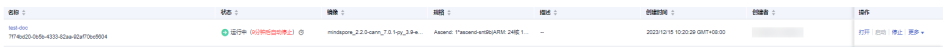
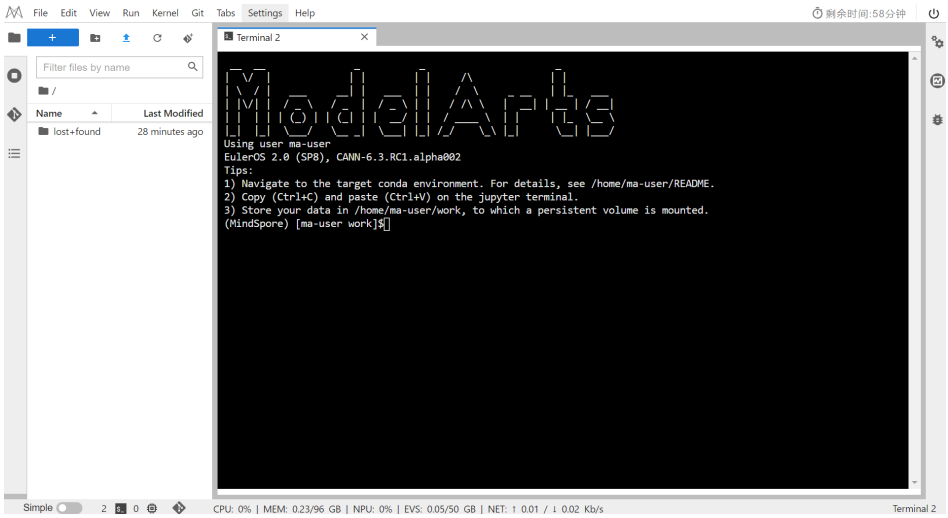


图 6-98 线上 Notebook 入口



----结束

ModelArts Lite DevServer

开通裸金属服务器资源请见**DevServer资源开通**，在裸金属服务器上搭建迁移环境请见**裸金属服务器环境配置指导**。

6.5.5 模型适配

6.5.5.1 基于 MindSpore Lite 的模型转换

迁移推理业务的整体流程如下：

- **模型准备**
- **转换关键参数准备**
- **模型转换**
- **推理应用适配**

主要通过MindSpore Lite（简称MSLite）进行模型的转换，进一步通过MindSpore Runtime支持昇腾后端的能力来将推理业务运行到昇腾设备上。

模型准备

MindSpore Lite提供的模型convertor工具可以支持主流的模型格式到MindIR的格式转换，用户需要导出对应的模型文件，推荐导出为ONNX格式。

1. 如何导出ONNX模型
   - PyTorch转ONNX，操作指导请见**此处**。
   - PyTorch导出ONNX模型样例如下：

```
import torch
import torchvision
model = torchvision.models.resnet50(pretrained=True)
# 保存模型为ONNX格式
torch.onnx.export(model, torch.randn(1, 3, 224, 224), "resnet50.onnx")
```

- TensorFlow导出ONNX模型，操作指导请见**此处**。

2. 如何导出PTH模型

   PyTorch模型导出时需要包含模型的结构信息，需要利用jit.trace方式完成模型的导出与保存。

```
# If you are instantiating the model with *from_pretrained* you can also easily set the TorchScript flag
model = BertModel.from_pretrained("bert-base-uncased", torchscript=True)

# Creating the trace
traced_model = torch.jit.trace(model, [tokens_tensor, segments_tensors])
torch.jit.save(traced_model, "traced_bert.pt")
```

转换关键参数准备

对应的模型转换成MindIR格式，通过后端绑定的编译形式来运行以达到更好的性能（类似静态图的运行模式），所以需要提前准备以下几个重点参数。

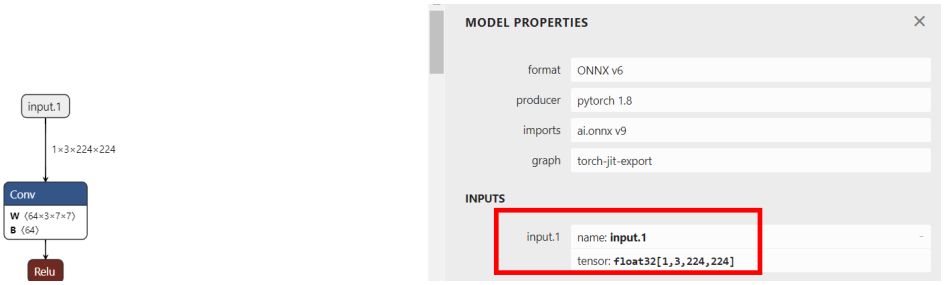
1. 输入的inputShape，包含batch信息。

   MSLite涉及到编译优化的过程，不支持完全动态的权重模式，需要在转换时确定对应的inputShape，用于模型的格式的编译与转换，可以在**netron官网**进行查看，或者对于模型结构中的输入进行shape的打印，并明确输入的batch。

   一般来说，推理时指定的inputShape是和用户的业务及推理场景是紧密相关的，可以通过原始模型推理脚本或者网络模型进行判断。需要把Notebook中的模型下载到本地后，再放入netron官网中，查看其inputShape。

   如果netron中没有显示inputShape，可能由于使用了动态shape模型导致，请确保使用的是静态shape模型，静态shape模型文件导出方法请参考**模型准备**。

   图 6-99 netron 中查看 inputShape



2. 精度选择。

   精度选择需要在模型转换阶段进行配置，执行converter_lite命令式通过--configFile参数指定配置文件路径，配置文件通过**precision_mode**参数指定精度模式。可选的参数有“enforce_fp32”，“preferred_fp32”，“enforce_fp16”，“enforce_origin”或者“preferred_optimal”，默认为“enforce_fp16”。

```
[ascend_context]
precision_mode= preferred_fp32
```

模型转换

在ModelArts开发环境中，通过对应的转换预置镜像，直接执行对应的转换过程，对应的转换和评估工具都已经预置了最新版本，详细介绍请见**使用说明**。inputShape查看方法请见**转换关键参数准备**。

```
!converter_lite --modelFile=resnet50.onnx --fmk=ONNX --outputFile=resnet50 --saveType=MINDIR --inputShape="input.1:1,3,224,224" --device=Ascend
```

为了简化用户使用，ModelArts提供了Tailor工具便于用户进行模型转换，具体使用方式参考**Tailor指导文档**。

推理应用适配

MindSpore Lite提供了JAVA/C++/Python API，进行推理业务的适配，并且在构建模型时，通过上下文的参数来确定运行时的具体配置，例如运行后端的配置等。下文以Python接口为例。

使用MindSpore Lite推理框架执行推理并使用昇腾后端主要包括以下步骤：

- 创建运行上下文：创建**Context**，保存需要的一些基本配置参数，用于指导模型编译和模型执行，在昇腾迁移时需要特别指定target为“Ascend”，以及对应的deivce_id。
```
context = mslite.Context()
context.target = ["ascend"]
context.ascend.device_id = 0
```
- 模型加载与编译：执行推理之前，需要调用**Model**的**build_from_file**接口进行模型加载和模型编译。模型加载阶段将文件缓存解析成运行时的模型。模型编译阶段会耗费较多时间所以建议Model创建一次，编译一次，多次推理。
```
model = mslite.Model()
model.build_from_file("./resnet50.mindir", mslite.ModelType.MINDIR, context)
```
- 输入数据：编译后的模型提供了predict接口用户执行模型推理任务，Inputs输入为List Tensor，这里的Tensor是MSLite的概念，具体的列表长度和tensor类型由转换时的InputShape来确定，由于后端指定了ascend，这些tensor都是在昇腾设备的显存中，用户需要在对应的tensor中填入数据，这些数据也会被搬移到显存中，进一步对于Inputs输入的内容进行处理。
```
data = convert_img(input_image)
in_data = [np.array(data)]
inputs = model.get_inputs()
for i, _input in enumerate(inputs):
_input.set_data_from_numpy(in_data[i])
```
- 执行推理：使用**Model**的**predict**进行模型推理，返回值为Outputs，也是List Tensor类型，具体的长度和类别由模型定义，对应的Tensor数据由于指定了ascend后端，Output的内容在显存中，通过tesnor的get_data_to_numpy方法来获取，并将数据读取到内存中使用。
```
outputs = model.predict(inputs)
outputs = [output.get_data_to_numpy() for output in outputs]
```
更多Python接口的高级用法与示例，请参考**Python API**。

6.5.5.2 动态 shape

在某些推理场景中，模型输入的shape可能是不固定的，因此需要支持用户指定模型的动态shape，并能够在推理中接收多种shape的输入。在CPU上进行模型转换时无需考虑动态shape问题，因为CPU算子支持动态shape；而在Ascend场景上，算子需要指定具体的shape信息，并且在模型转换的编译阶段完成对应shape的编译任务，从而能够在推理时支持多种shape的输入。

动态 batch

在模型转换阶段通过--configFile参数指定配置文件，并且在配置文件中配置input_shape及dynamic_dims动态参数。其中input_shape的-1表示动态shape所在的维度，dynamic_dims指定动态维度的取值范围，比如“[1~4],[8],[16]”表示该动态维度支持1、2、3、4、8、6共六种大小。

```
# config.ini
[ascend_context]
```

```
input_shape=input.1:[-1,3,224,224]
dynamic_dims=[1~4],[8],[16]
```

在执行convert_lite命令时，指定--configFile=config.ini即可自动编译指定的动态shape。

```
#shell
converter_lite --modelFile=resnet50.onnx --fmk=ONNX --device=Ascend --outputFile=resnet50_dynamic --saveType=MINDIR --configFile=config.ini
```

注意：推理应用开发时，需要使用模型的Resize功能，改变输入的shape。而且Resize操作需要在数据从host端复制到device端之前执行，下面是一个简单的示例，展示如何在推理应用时使用动态Shape。

```
import mindspore_lite as mslite
import numpy as np
from PIL import Image
# 设置目标设备上下文为Ascend，指定device_id为0
context = mslite.Context()
context.target = ["ascend"]
context.ascend.device_id = 0
# 构建模型
model = mslite.Model()
model.build_from_file("./resnet50_dynamic.mindir", mslite.ModelType.MINDIR, context)
data = np.random.rand(8, 3, 224, 224).astype(np.float32)
inputs = model.get_inputs()
model.resize(inputs, [list(data.shape)])
inputs[0].set_data_from_numpy(data)
# 前向推理，并将结果从device侧传到host侧
outputs = model.predict(inputs)[0].get_data_to_numpy()
print(outputs.shape)  # (8, 1000)
```

动态分辨率

动态分辨率可以用于设置输入图片的动态分辨率参数。适用于执行推理时，每次处理图片宽和高不固定的场景，该参数需要与input_shape配合使用，input_shape中-1的位置为动态分辨率所在的维度。使用方法可参考**Ascend配置文件说明**。

6.5.6 精度校验

转换模型后执行推理前，可以使用benchmark工具对MindSpore Lite云侧推理模型进行基准测试。它不仅可以对MindSpore Lite云侧推理模型前向推理执行耗时进行定量分析（性能），还可以通过指定模型输出进行可对比的误差分析（精度）。

精度测试

benchmark工具用于精度验证，主要工作原理是：固定模型的输入，通过benchmark工具进行推理，并将推理得到的输出与标杆数据进行相似度度量（余弦相似度和平均相对误差），得到模型转换后的精度偏差信息。使用benchmark进行精度比对的基本流程如下：

1. 将模型输入保存二进制文件。

   ```
   # 数据读取，预处理
   image = img_preprocess(image_path)
   image = np.array(image, dtype=np.float32)
   image = np.frombuffer(image.tobytes(), np.float32)
   # 保存网络输入为二进制文件
   image.tofile("input_data.bin")
   ```

2. 将基准模型的输出保存到文本文件。

   本例中输出节点名称为output_node_name，输出节点的shape为“(1, 1000)”，因此一共有两维，对应的输出文件为“output_node_name 2 1 1000”，再加上输出的值即可。

```
# 基于原始pth模型前向推理
output = model_inference(input_data)
# 保存网络输出节点名称、维度、shape及输出到本地文件
with open("output_data.txt", "w") as f:
f.write("output_node_name 2 1 1000\n")
f.write(" ".join([str(i) for i in output]))
```

3. 使用benchmark工具进行精度对比。

```
#shell
benchmark --modelFile=model.mindir --inputShapes=1,3,224,224 --inDataFile=input_data.bin --
device=Ascend --benchmarkDataFile=output_data.txt --accuracyThreshold=5 --
cosineDistanceThreshold=0.99
```

其中，--accuracyThreshold=5表示平均绝对误差的容忍度最大为5%，--cosineDistanceThreshold =0.99表示余弦相似度至少为99%，--inputShapes可将模型放入到**netron官网**中查看。

图 6-100 benchmark 对接结果输出示例图

```
MarkAccuracy
InData 0: -0.559551 -0.559551 -0.508177 -0.782173 -0.422553 0.211063 0.330936 -0.0458088 -0.217056 -0.251306 0.348061 0.125439 0.142564 -0.371179 0.262437 -0.525302 -0.62805 -0.542427 -0.
525302 -0.131433
================ Comparing Output data ================
Data of node 495 : 1.73535 -0.799316 0.40332 -0.526367 -2.2832 -0.32666 -1.96484 -0.309326 -0.524902 -1.40625 -2.53125 0.010025 -1.91797 -3.63281 0.98584 -3.35547 -2.19922 -3.04492 -1.166
99 -3.12891 0.322266 -1.2041 -0.265625 1.20312 -1.43555 2.54492 -2.02539 -1.69434 -0.932129 -1.88672 -2.37109 -0.712402 -1.66602 0.773438 0.3396 -0.0942383 1.9209 -0.0312347 -2.23633 -0.9
7998 -3.23047 -3.54883 -3.17383 -3.23828 -2.72656 0.18811 -2.19727 -1.21387 1.15723 1.97266
Mean bias of node/tensor 495 : 0.645113%
Mean bias of all nodes/tensors: 0.645113%
============================================================

================ Comparing Output data ================
Data of node 495 : 1.73535 -0.799316 0.40332 -0.526367 -2.2832 -0.32666 -1.96484 -0.309326 -0.524902 -1.40625 -2.53125 0.010025 -1.91797 -3.63281 0.98584 -3.35547 -2.19922 -3.04492 -1.166
99 -3.12891 0.322266 -1.2041 -0.265625 1.20312 -1.43555 2.54492 -2.02539 -1.69434 -0.932129 -1.88672 -2.37109 -0.712402 -1.66602 0.773438 0.3396 -0.0942383 1.9209 -0.0312347 -2.23633 -0.9
7998 -3.23047 -3.54883 -3.17383 -3.23828 -2.72656 0.18811 -2.19727 -1.21387 1.15723 1.97266
Mean cosine distance of node/tensor 495 : 99.9999%
Cosine distance of all nodes/tensors: 0.999999403953552
============================================================
```

为了简化用户使用，ModelArts提供了Tailor工具便于用户进行Benchmark精度测试，具体使用方式参考**Tailor指导文档**。

6.5.7 性能调优

性能测试

benchmark工具也可用于性能测试，其主要的测试指标为模型单次前向推理的耗时。在性能测试任务中，与精度测试不同，并不需要用户指定对应的输入（inDataFile）和输出的标杆数据（benchmarkDataFile），benchmark工具会随机生成一个输入进行推理，并统计推理时间。执行的示例命令行如下。

```
#shell
benchmark --modelFile=resnet50.mindir --device=Ascend
```

为了简化用户使用，ModelArts提供了Tailor工具便于用户进行Benchmark性能测试，具体使用方式参考**Tailor指导文档**。

在某些推理场景中，模型输入的shape可能是不固定的，因此需要支持用户指定模型的动态shape，并能够在推理中接收多种shape的输入。在CPU上进行模型转换时无需考虑动态shape问题，因为CPU算子支持动态shape；而在昇腾场景上，算子需要指定具体的shape信息，并且在模型转换的编译阶段完成对应shape的编译任务，从而能够在推理时支持多种shape的输入。

绝大多数情况下，昇腾芯片推理性能相比于CPU会好很多，但是也可能会遇到和CPU推理性能并无太大差别甚至出现劣化的情况。造成这种情况的原因可能有如下几种：

1. 模型中存在大量的类似于Pad或者Strided_Slice等算子，其在CPU和Ascend上的实现方法存在差异（硬件结构不同），后者在运算此类算子时涉及到数组的重排，性能较差；
2. 模型的部分算子在昇腾上不支持，或者存在Transpose操作，会导致模型切分为多个子图，整体的推理耗时随着子图数量的增多而增长；

3. 模型没有真正的调用昇腾后端，而是自动切换到了CPU上执行，这种情况可以通过输出日志来进行判断。

自助性能调优三板斧

基于上一步完成的性能测试，为了最大化模型推理性能，首先确保当前使用的CANN版本是最新版本（最新版本请见**此处**），每个迭代的CANN版本都有一定的性能收益。在此基础上，可以进行三板斧自助工具式性能调优。这些调优过程由大量的项目交付经验总结，帮助您获得模型最佳推理性能，重复**性能测试**章节可以验证对应的收益情况。

自助性能调优三板斧分别为：**通过固定shape获取更好的常量折叠**、**AOE性能自动调优**、**自动高性能算子生成工具**。

- **通过固定shape获取更好的常量折叠**

  在MindIR格式转换时（即执行converter_lite命令时），通过指定具体的静态shape，并且打开--optimize参数指定“ascend_oriented”能够获得更好的常量折叠优化效果。inputShape查看方法请见**转换关键参数准备**。

  ```
  Ascend Optimization Engine
  converter_lite --modelFile=resnet50.onnx --fmk=ONNX --outputFile=resnet50 --saveType=MINDIR --
  inputShape="input.1:1,3,224,224" --optimize=ascend_oriented
  ```

  常量折叠是编译器优化中的通用技术之一，在编译节点简化常量表达。通过多数的现代编译器不会真的产生两个乘法的指令再将结果存储下来，取而代之的是会识别出语句的结构，并在编译时期将数值计算出来而不是运行时去计算（在本例子，结果为2,048,000）。

  ```
  i = 320 * 200 * 32;
  ```

  AI编译器中，常量折叠是将计算图中预先可以确定输出值的节点替换成常量，并对计算图进行一些结构简化的操作，例如ADDN操作，以及在推理过程中的batch normalization操作等。

  以BN折叠为例，如下表示折叠后获得的性能收益。

  图 6-101 BN 折叠下前向运算性能收益

模型	CPU 前向时间	GPU 前向时间
Resnet50（合并前）	176.17ms	11.03ms
Resnet50（合并后）	161.69ms	7.3ms
提升	~8%	~34%

- **AOE性能自动调优**

  自动性能调优工具AOE(Ascend Optimization Engine)，可以对于模型的图和算子运行通过内置的知识库进行自动优化，以提升模型的运行效率。开启AOE调优后，模型转换时会自动进行性能调优操作，该过程耗时较长，可能需要数小时。

  AOE性能自动优化在模型转换阶段进行配置（即执行converter_lite命令时），通过--configFile参数指定配置文件aoe_config.ini，配置文件通过aoe_mode参数指定调优模式。可选值有：

  - “subgraph tuning”：子图调优。
  - “operator tuning”：算子调优。
  - “subgraph tuning, operator tuning”：先进行子图调优，再进行算子调优。

推荐先进行子图调优，再进行算子调优，因为先进行子图调优会生成图的切分方式，子图调优后算子已经被切分成最终的shape了，再进行算子调优时，会基于这个最终shape去做算子调优。如果优先算子调优，这时调优的算子shape不是最终切分后的算子shape，不符合实际使用场景。

本例同时指定了子图调优和算子调优，工具会先进行子图调优，再进行算子调优。

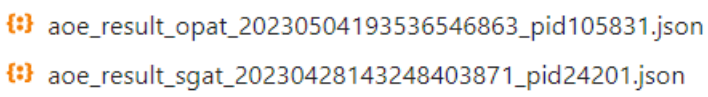
```
# aoe_config.ini
[ascend_context]
aoe_mode="subgraph tuning, operator tuning"
```

指定--configFile=aoe_config.ini即可自动进行性能优化。

```
#shell
converter_lite --modelFile=resnet50.onnx --fmk=ONNX --device=Ascend --outputFile=resnet50_aoe --saveType=MINDIR --configFile=aoe_config.ini
```

命令执行成功后，性能自动优化前后的性能对比会打印到控制台上，同时会生成更为详细的json格式调优报告。

图 6-102 自动调优输出文件



需要注意的是，并不是所有的模型使用性能自动调优都是有收益的，在本例中，ResNet50模型自动调优收益甚微（模型转换时已经做了部分针对性优化），在有些比较复杂的模型场景下可能会有较好的收益。比如VAE_ENCODER模型使用算子调优收益为11.15%。

图 6-103 VAE_ENCODER 模型使用 AOE 自动调优在屏幕上显示日志

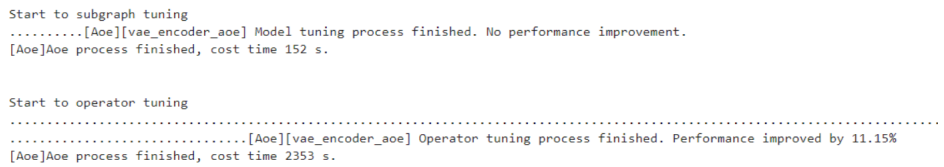
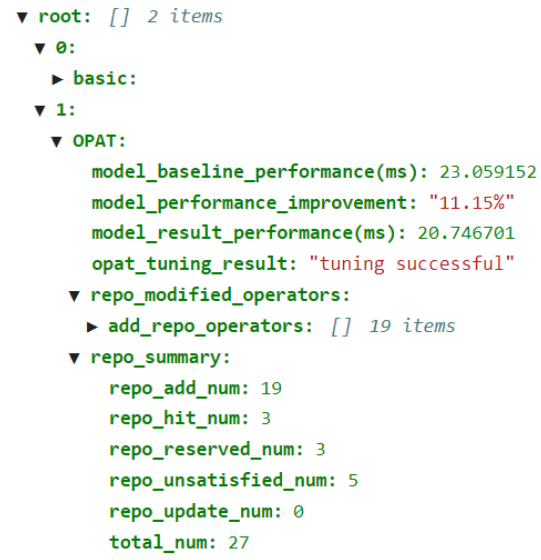


图 6-104 AOE 自动调优的输出样例



其中：

- model_baseline_performance表示调优前模型执行时间，单位为ms。
- model_performance_improvement表示调优后模型执行时间减少百分比。
- model_result_performance表示调优后模型执行时间。
- repo_summary中的信息表示调优过程中使用到的知识库算子个数或者追加到知识库的算子个数。

AOE自动调优更多介绍可参考**Ascend转换工具功能说明**。

- **自动高性能算子生成工具**

  自动高性能算子生成工具AKG(Auto Kernel Generator)，可以对深度神经网络模型中的算子进行优化，并提供特定模式下的算子自动融合功能，可提升在昇腾硬件后端上运行模型的性能。

  AKG的配置也是在模型转换阶段进行配置（即执行converter_lite命令时），通过指定对应的配置文件akg.cfg，设置对应的akg优化级别，并且在模型转换时参考样例进行对应的配置。

  ```
  # akg.cfg
  [graph_kernel_param]
  opt_level=2
  ```

  执行命令：

  ```
  # shell
  converter_lite --fmk=ONNX --modelFile=model.onnx --outputFile=model --configFile=akg.cfg --optimize=ascend_oriented
  ```

  自动高性能算子生成工具AKG更多介绍可参考**图算融合配置说明**和**MindSpore AKG**。

6.5.8 迁移过程使用工具概览

基础的开发工具在迁移的预置镜像和开发环境中都已经进行预置，用户原则上不需要重新安装和下载，如果预置的版本不满足要求，用户可以执行下载和安装与覆盖操作。

模型自动转换评估工具 Tailor

为了简化用户使用，ModelArts提供了Tailor工具，将模型转换、精度benchmark、性能benchmark和profiling采集工具集成到同一个工具中，极大简化了用户的使用流程。建议在迁移过程中使用Tailor工具替代下面列举的原始工具MS Convertor、Benchmark和msprof。使用指导详见**链接**。

模型转换工具

离线转换模型功能的工具**MSLite Convertor**，支持onnx、pth、tensorflowLite多种类型的模型转换，转换后的模型可直接运行在MindSpore运行时后端，用于昇腾推理。

精度性能检查工具

Benchmark精度检查工具，可以转换模型后执行推理前，使用其对MindSpore Lite模型进行基准测试，它不仅可以对MindSpore Lite模型前向推理执行耗时进行定量分析（性能），还可以通过指定模型输出进行可对比的误差分析（精度）。

模型自动调优工具

AOE(Ascend Optimization Engine)是一个昇腾设备上模型运行自动调优工具，作用是充分利用有限的硬件资源，以满足算子和整网的性能要求。在推理场景下使用，可以对于模型的图和算子运行内置的知识库进行自动优化，以提升模型的运行效率。

自动高性能算子生成工具 AKG

AKG(Auto Kernel Generator)对深度神经网络中的算子进行优化，并提供特定模式下的算子自动融合功能。提升在昇腾硬件后端上运行网络的性能。

AKG由三个基本的优化模块组成：规范化、自动调度和后端优化。

- 规范化： 为了解决polyhedral表达能力的局限性（只能处理静态的线性程序），需要首先对计算公式IR进行规范化。规范化模块中的优化主要包括自动运算符inline、自动循环融合和公共子表达式优化等。
- 自动调度： 自动调度模块基于polyhedral技术，主要包括自动向量化、自动切分、thread/block映射、依赖分析和数据搬移等。
- 后端优化： 后端优化模块的优化主要包括TensorCore使能、双缓冲区、内存展开和同步指令插入等。

性能分析工具

msprof命令行工具提供了采集通用命令以及AI任务运行性能数据、昇腾AI处理器系统数据、Host侧系统数据和采集和解析能力。面向推理的场景，可以对于模型的执行性能数据进行收集，可基于收集的性能数据进行性能分析。

6.5.9 常见问题

6.5.9.1 MindSpore Lite 问题定位指南

在使用MindSpore Lite过程中遇到问题时，可参考MindSpore Lite官网提供的**问题定位指南**进行问题定位。

6.5.9.2 模型转换报错如何查看日志和定位?

通过如下的配置项打开对应的模型转换日志，可以看到更底层的报错。如配置以下的环境变量之后，再重新转换模型，导出对应的日志和dump图进行分析：

1. 报错日志中搜到“not support onnx data type”，表示MindSpore暂不支持该算子。
2. 报错日志中搜到“Convert graph to om failed”，表示CANN模块进行图编译存在保存，需要结合CANN的报错日志和dump图进行具体分析。

配置方式参考如下：

1. 打开DEBUG日志。
   - 设置MindSpore日志环境变量。
     ```
     export GLOG_v=0
     # 0-DEBUG、1-INFO、2-WARNING、3-ERROR
     ```
   - 设置CANN日志环境变量。
     ```
     # 0：表示DEBUG。1：表示INFO。2：表示WARNING。3：表示ERROR。4: 表示NONE。
     export ASCEND_GLOBAL_LOG_LEVEL=1
     ```

```
# 表示日志打印
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

2. DUMP模型转换中间图。

   设置DUMP中间图环境变量。

```
# 1：表示dump图全量内容。2：表示不dump权重数据的基础图。3：表示只dump节点关系的精简图。
export DUMP_GE_GRAPH=2
# 1：表示dump图所有图。2：表示dump除子图外的所有图。3：表示只dump最后一张图。
export DUMP_GRAPH_LEVEL=2
```

6.5.9.3 日志提示 Compile graph failed

问题现象

日志提示：Compile graph failed。

图 6-105 报错提示

```
[ERROR] ME(103674,ffff8d8790b0,python):2023-04-24-11:12:26.235.526 [mindspore/lite/src/extendrt/session/single_op_session.cc:242] CompileGraph] Only support
 CustomAscend, but got Reshape, node Reshape_9
[ERROR] ME(103674,ffff8d8790b0,python):2023-04-24-11:12:26.235.617 [mindspore/lite/src/extendrt/cxx_api/model/model_impl.cc:280] BuildByBufferImpl] compile
graph failed.
```

原因分析

模型转换时未指定Ascend后端。

处理方法

需要在模型转换阶段指定“--device=Ascend”。

6.5.9.4 日志提示 Custom op has no reg_op_name attr

问题现象

日志提示：Custom op has no reg_op_name attr。

图 6-106 报错提示

```
[ERROR] GE_ADPT(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.198 [mindspore/ccsrc/transform/graph_ir/op_adapter.cc:179] GetCustomOpType] Custom op has no reg_o
p_name attr.
[ERROR] GE_ADPT(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.262 [mindspore/ccsrc/transform/graph_ir/op_adapter.cc:179] GetCustomOpType] Custom op has no reg_o
p_name attr.
[WARNING] GE_ADPT(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.292 [mindspore/ccsrc/transform/graph_ir/op_adapter.cc:202] GenerateCustomOp] Custom op node has
no input_names, op[Custom].
[ERROR] GE_ADPT(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.307 [mindspore/ccsrc/transform/graph_ir/op_adapter.cc:179] GetCustomOpType] Custom op has no reg_o
p_name attr.
[WARNING] GE_ADPT(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.324 [mindspore/ccsrc/transform/graph_ir/op_adapter.cc:206] GenerateCustomOp] Custom op node has
no output_names, op[Custom].
[ERROR] CORE(151711,ffffbf4840b0,python):2023-04-24-11:42:46.677.445 [mindspore/core/utils/log_adapter.cc:388] operator^] Runtime error for null exception handler.
[ERROR] ME(151711,ffffbf4840b0,python):2023-04-24-11:42:46.706.308 [mindspore/lite/src/extendrt/cxx_api/model/model.cc:100] Build] Catch exception: Cast failed, origi
nal value: (3, ,, 1, ,, 7, 7, ,, 7, 6, 8, ,, 2, ,, 1, ,, 7, 6, 8, ,), type: ValueTuple

----------------------------------------------------
- C++ Call Stack: (For framework developers)
----------------------------------------------------
mindspore/core/ir/anf.h:1005 GetValue
```

处理方法

定义context时无需指定：

```
context.ascend.provider = "ge"
```

6.5.10 推理业务迁移评估表

通用的推理业务及LLM推理可提供下表进行业务迁移评估：

收集项	说明	实际情况（请填写）
项目名称	项目名称，例如：XXX项目。	-
使用场景	例如： ● 使用YOLOv5算法对工地的视频流裁帧后进行安全帽检测。 ● 使用BertBase算法对用户在app上购买商品后的评论进行理解。	-
CPU架构	X86/ARM，自有软件是否支持ARM。 例如：4个推理模型在ARM上运行，6个推理模型在X86上运行。	-
当前使用的操作系统及版本	当前推理业务的操作系统及版本，如：Ubuntu 22.04。 是否使用容器化运行业务，以及容器中OS版本，HostOS中是否有业务软件以及HostOS的类型和版本。 需要评估是否愿意迁移到华为云的通用OS。	-
AI引擎及版本	当前引擎（TF/PT/LibTorch），是否接受切换MindSpore。 例如：当前使用TF 2.6，PyTorch 1.10，可以接受切换MindSpore。	-
业务编程语言、框架、版本。	C++/Python/JAVA等。 例如：业务逻辑使用JAVA，推理服务模块使用C++自定义实现推理框架，Python 3.7等。	-
CPU使用率	业务中是否有大量使用CPU的代码，以及日常运行过程中CPU的占用率（占用多少个核心），以及使用CPU计算的业务功能说明和并发机制。	-
是否有Linux内核驱动	是否有业务相关的Linux内核驱动代码。	-
依赖第三方组件列表	当前业务依赖的第三方软件列表（自行编译的第三方软件列表）。 例如：Faiss等。	-
推理框架	TensorRT/Triton/MSLite等。 例如： ● 2个推理模型使用TensorRT框架，5个使用Triton框架。 ● 通过stable-diffusion的WebUI提供AIGC推理服务。	-

收集项	说明	实际情况（请填写）
GPU卡的类型	Vnt1/Ant1/Ant03/Tnt004等。 例如： 20卡Ant1，运行Bert Large推理。 10卡Tnt004运行YOLOv5。	-
Backbone类型	ResNet/DarkNet/Transformer等。 例如： ● 5个模型使用ResNet Backbone，应用与监控。 ● 3个模型使用Transformer，应用于自然语言处理xxx。 ● 使用stable-diffusion的典型模型：TextEncoder、VaeEncoder、unet、VaeDecoder、SafetyChecker，没有使用LoRA等动态加载的诉求。	-
模型训练方式	关于推理业务中使用的模型，填写该模型训练时使用的框架以及套件。 例如：模型使用PyTorch+Megatron+DeepSpeed进行训练。	-
自定义算子	是否有自定义算子，CPU还是CUDA，复杂程度。 例如：有5个CUDA自定义算子。1个高复杂度算子，基于C++开发2000行代码。4个中等复杂度算子，基于C++开发，平均每个自定义算子约500行代码。	-
动态shape	是否需要支持动态shape。 例如：需要动态Shape，需要动态Shape的模型有ResNet-50、YOLOv5。	-
参数类型（FP32/FP16）	FP32还是FP16混合，判断精度调优难度。 例如：ResNet-50、YOLOv5模型使用FP16。BertLarge使用FP32。	-
模型变更频率	模型变更场景如下： ● 数据增量，模型算子未变更。 ● 数据增量，模型算子变化，例如： – 网络结构变化。 – AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
是否使用华为MDC产品	如果使用华为MDC产品，请填写MDC版本号，如果没有可以不填。 例如：使用了C83版本。	-

收集项	说明	实际情况（请填写）
性能指标与预期	● 例1： 模型：YOLOv5 运行环境：Vnt1 单卡 性能指标：QPS 100/s （两进程） 性能约束：单次请求最大可以接受时延需小于100ms 性能预期：QPS 130/s ● 例2： 模型：OCR 运行环境：6348（单核48U超线程） 性能指标：QPS 10/s（四进程） 性能约束：单次请求最大可以接受时延需小于1s 性能预期：QPS 20/s	-
业务访问方式	推理业务访问：“客户端 -> 云服务” 或 “云客户端 -> 云服务”。 推理业务时延要求，客户端到云服务端到端可接受时延。 例如：当前是“客户端 -> 云服务”模式，客户端请求应答可接受的最长时延为2秒。	-
模型参数规模，是否涉及分布式推理	10B/100B，单机多卡推理。	-
能否提供实际模型、网络验证的代码和数据等信息	提供实际模型、网络验证的代码和数据。 提供与业务类型类似的开源模型，例如GPT3 10B/13B。 提供测试模型以及对应的Demo代码路径（开源或共享）。 可以提前的完成POC评估，例如框架、算子支持度，以及可能的一些性能指标。	-

如果是AIGC场景的业务例如Stable Diffusion，请在上表的基础上，再提供以下信息：

收集项	说明	实际情况（请填写）
使用场景	例如： 1. 业务是文生图，图生图等。 2. 业务是否需要频繁更新模型，或者需要动态加载Lora。	-

收集项	说明	实际情况（请填写）
stable-diffusion套件	1. 使用differusers （ https://github.com/huggingface/diffusers ）。 2. stable-diffusion-webui （ https://github.com/AUTOMATIC1111/stable-diffusion-webui ）。 3. 如果是基于其他开源，需要附带开源代码仓地址。	-
具体使用库	例如： 1. 使用了哪个pipeline （ 例如lpw_stable_diffusion.py ）。 2. 使用了哪个huggingface的模型 （ 例如digiplay/majicMIX_realistic_v6 ）。 3. 如果有预处理，后处理，对应的模型是什么（ 例如后处理的超分模型 ）。	-
Lora/TextInversion	1. 是否有动态加载Lora的需求，可否接受把Lora固定到模型内。 2. 是否使用了TextInversion，是否需要动态加载。	-
动态shape	是否可接受分档shape（ 固定n个挡位的shape ）。	-
模型变更频率	模型变更场景如下： 1. 数据增量，模型算子未变更。 2. 数据增量，模型算子变化，例如： ● 网络结构变化。 ● AI框架版本升级，使用了新版本算子。 例如：每半年对模型进行一次变更，变更的内容包含模型结构，并升级AI框架。	-
尺寸要求	超分前产生的图片尺寸要求： 1. 512*512 2. 720*720 3. 1080 *1080 4. 1920*1920 （ shape过大可能导致性能下降 ）	-

7 Standard 权限管理

7.1 ModelArts 权限管理基本概念

ModelArts作为一个完备的AI开发平台，支持用户对其进行细粒度的权限配置，以达到精细化资源、权限管理之目的。这类特性在大型企业用户的使用场景下很常见，但对个人用户则显得复杂而意义不足，所以建议个人用户在使用ModelArts时，参照**个人用户快速配置ModelArts访问权限**来进行初始权限设置。

说明

您是否需要阅读本文档?

如果下述问题您的任何一个回答为“是”，则需要阅读此文档

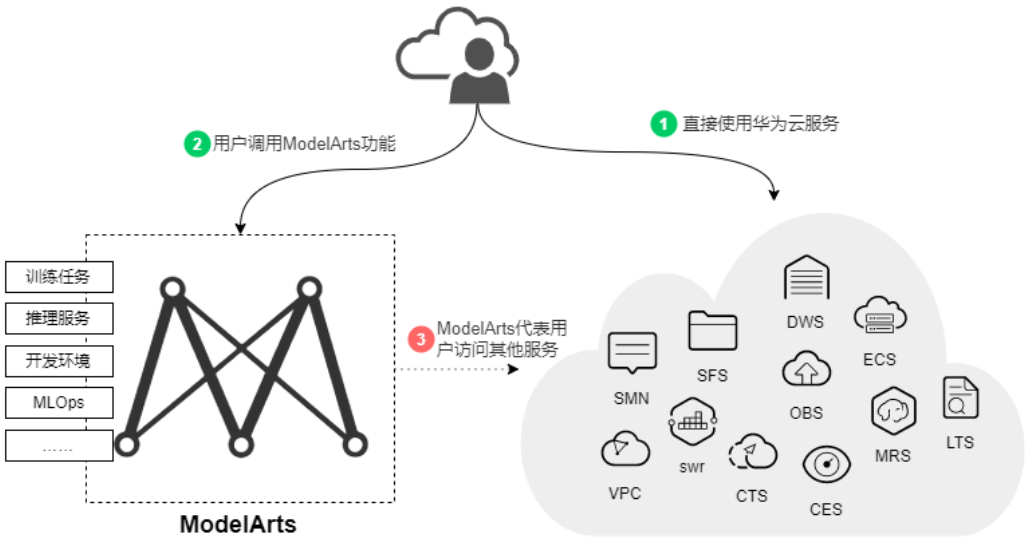
- 您是企业用户，且
  - 存在多个部门，且需要限定不同部门的用户只能访问其专属资源、功能
  - 存在多种角色（如管理员、算法开发者、应用运维），希望限制不同角色只能使用特定功能
  - 逻辑上存在多套“环境”且相互隔离（如开发环境、预生产环境、生产环境），并限定不同用户在不同环境上的操作权限
  - 其他任何需要对特定子用户（组）做出特定权限限制的情况
- 您是个人用户，但已经在IAM创建多个子用户，且期望限定不同子用户所能使用的ModelArts功能、资源不同。
- 希望了解ModelArts的权限控制能力细节，期望理解其概念和实操方法。

ModelArts的大部分权限管理能力均基于统一身份认证服务（Identity and Access Management，简称IAM）来实现，在您继续往下阅读之前，强烈建议您先行熟悉**IAM基本概念**，如果能完整理解IAM的所有概念，将更加有助于您理解本文档。

为了支持用户对ModelArts的权限做精细化控制，提供了3个方面的能力来支撑，分别是：权限、委托和工作空间。下面分别讲解。

理解 ModelArts 的权限与委托

图 7-1 权限管理抽象



ModelArts与其他服务类似，对外暴露的每个功能，都通过IAM的权限来进行控制。比如，用户（此处指IAM子用户，而非租户）希望在ModelArts创建训练作业，则该用户必须拥有 "modelarts:trainJob:create" 的权限才可以完成操作（无论界面操作还是API调用）。关于如何给一个用户赋权（准确讲是需要先将用户加入用户组，再面向用户组赋权），可以参考IAM的文档《**权限管理**》。

而ModelArts还有一个特殊的地方在于，为了完成AI计算的各种操作，AI平台在任务执行过程中需要访问用户的其他服务，典型的就是训练过程中，需要访问OBS读取用户的训练数据。在这个过程中，就出现了ModelArts“代表”用户去访问其他云服务的情形。从安全角度出发，ModelArts代表用户访问任何云服务之前，均需要先获得用户的授权，而这个动作就是一个“委托”的过程。用户授权ModelArts再代表自己访问特定的云服务，以完成其在ModelArts平台上执行的AI计算任务。

综上，对于**图1 权限管理抽象**可以做如下解读：

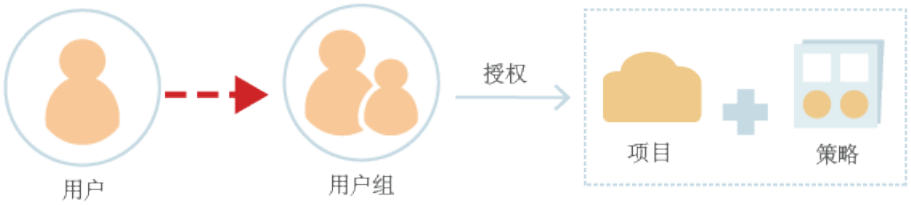
- 用户访问任何云服务，均是通过标准的IAM权限体系进行访问控制。用户首先需要具备相关云服务的权限（根据您具体使用的功能不同，所需的相关服务权限多寡亦有差异）。
- **权限**：用户使用ModelArts的任何功能，亦需要通过IAM权限体系进行正确权限授权。
- **委托**：ModelArts上的AI计算任务执行过程中需要访问其他云服务，此动作需要获得用户的委托授权。

ModelArts 权限管理

默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于授予的权限对云服务进行操作。

⚠ 注意

- ModelArts部署时通过物理区域划分，为项目级服务，授权时“选择授权范围方案”可以选择“指定区域项目资源”，如果授权时指定了区域（如华北-北京4）对应的项目（cn-north-4），则该权限仅对此项目生效；简单的做法是直接选择“所有资源”。
- ModelArts也支持企业项目，所以选择授权范围方案时，也可以指定企业项目。具体操作参见《**创建用户组并授权**》。



IAM在对用户组授权的时候，并不是直接将具体的某个权限进行赋权，而是需要先将权限加入到“策略”当中，再把策略赋给用户组。为了方便用户的权限管理，各个云服务都提供了一些预置的“系统策略”供用户直接使用。如果预置的策略不能满足您的细粒度权限控制要求，则可以通过“自定义策略”来进行精细控制。

表7-1列出了ModelArts的所有预置系统策略。

表 7-1 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限	系统策略

通常来讲，只给管理员开通“ModelArts FullAccess”，如果不需要太精细的控制，直接给所有用户开通“ModelArts CommonOperations”即可满足大多数小团队的开发场景诉求。如果您希望通过自定义策略做深入细致的权限控制，请阅读**ModelArts的IAM权限控制详解**。

说明

ModelArts的权限不会凌驾于其他服务的权限之上，当您给用户进行ModelArts赋权时，系统不会自动对其他相关服务的相关权限进行赋权。这样做的好处是更加安全，不会出现预期外的“越权”，但缺点是，您必须同时给用户赋予不同服务的权限，才能确保用户可以顺利完成某些ModelArts操作。

举例，如果用户需要用OBS中的数据进行训练，当已经为IAM用户配置ModelArts训练权限时，仍需同时为其配置对应的OBS权限（读、写、列表），才可以正常使用。其中OBS的列表权限用于支持用户从ModelArts界面上选择要进行训练的数据路径；读权限主要用于数据的预览以及训练任务执行时的数据读取；写权限则是为了保存训练结果和日志。

- 对于个人用户或小型组织，一个简单做法是为IAM用户配置“作用范围”为“全局级服务”的“Tenant Administrator”策略，这会使用户获得除了IAM以外的所有用户权限。在获得便利的同时，由于用户的权限较大，会存在相对较大的安全风险，需谨慎使用。（对于个人用户，其默认IAM账号就已经属于admin用户组，且具备Tenant Administrator权限，无需额外操作）
- 当您需要限制用户操作，仅为ModelArts用户配置OBS相关的最小化权限项，具体操作请参见**OBS权限管理**。对于其他云服务，也可以进行精细化权限控制，具体请参考对应的云服务文档。

ModelArts 委托授权

前文已经介绍，ModelArts在执行AI计算任务过程中，需要“代表”用户去访问其他云服务，而此动作需要提前获得用户的授权。在IAM权限体系下，此类授权动作是通过“委托”来完成。

关于委托的基本概念及操作可以参考对应的IAM文档《**委托其他云服务管理资源**》。

为了简化用户的委托授权操作，ModelArts增加了自动配置委托授权的支持，用户仅需在ModelArts控制台的“权限管理”页面中，为自己或特定用户配置委托即可。

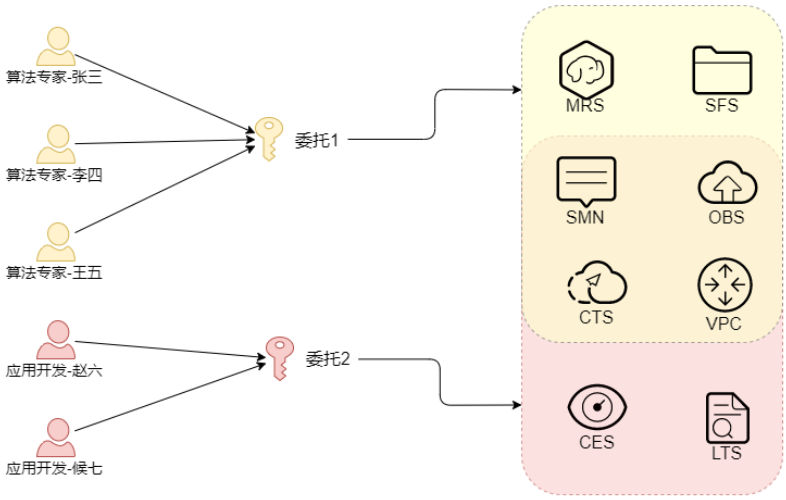
说明

- 只有具备IAM委托管理权限的用户才可以进行此项操作，通常是IAM admin用户组的成员才具备此权限。
- 目前ModelArts的委托授权操作是分区域操作的，这意味着您需要在每个您所用到的区域均执行委托授权操作。

在ModelArts控制台的“权限管理”页面，单击“添加授权”后，系统会引导您为特定用户或所有用户进行委托配置，通常默认会创建一个名为“modelarts_agency_<用户名>_随机ID”的委托条目。在权限配置的区域，您可以选择ModelArts提供的预置配置，也可以自定义选择您所授权的策略。当然如果这两种形态对于您的诉求均过于粗犷，您也可以直接在IAM管理页面里创建完全由您进行精细化配置的委托（需要委托给ModelArts服务），然后在此页面的委托选择里使用“已有委托”“”（而非“新增委托”）。

至此，您应该已经发现了一个细节，ModelArts在使用委托时，是将其与用户进行关联的，用户与委托的关系是多对1的关系。这意味着，如果两个用户需要配置的委托一致，那么不需要为每个用户都创建一个独立的委托项，只需要将两个用户都“指向”同一个委托项即可。

图 7-2 用户与委托对应关系



说明

每个用户必须关联委托才可以使用ModelArts，但即使委托所赋之权限不足，在API调用之初也不会报错，只有到系统具体使用到该功能时，才会发生问题。例如，用户在创建训练任务时打开了“消息通知”，该功能依赖SMN委托授权，但只有训练任务运行过程中，真正需要发送消息时，系统才会“出错”，而有些错误系统会选择“忽略”，另一些错误则可能导致任务直接失败。当您做深入的“权限最小化”限制时，请确保您在ModelArts上将要执行的操作仍旧有足够的权限。

严格授权模式

严格授权模式是指在IAM中创建的子用户必须由账号管理员显式在IAM中授权，才能访问ModelArts服务，管理员用户可以通过授权策略为普通用户精确添加所需使用的ModelArts功能的权限。

相对的，在非严格授权模式下，子用户不需要显式授权就可以使用ModelArts，管理员需要在IAM上为子用户配置Deny策略来禁止子用户使用ModelArts的某些功能。

账号的管理员用户可以在“权限管理”页面修改授权模式。

须知

如无特殊情况，建议优先使用严格授权模式。在严格授权模式下，子用户要使用ModelArts的功能都需经过授权，可以更精确的控制子用户的权限范围，达成权限最小化的安全策略。

用工作空间限制资源访问

工作空间是ModelArts面向企业用户提供的一个高阶功能，用于进一步将用户的资源划分在多个**逻辑隔离**的空间中，并支持以空间维度进行访问的权限限定。目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持经理申请开通。

在开通工作空间后，系统会默认为您创建一个“default”空间，您之前所创建的所有资源，均在该空间下。当您创建新的工作空间之后，相当于您拥有了一个新的

“ModelArts分身”，您可以通过菜单栏的左上角进行工作空间的切换，不同工作空间中的工作互不影响。

创建工作空间时，必须绑定一个企业项目。多个工作空间可以绑定到同一个企业项目，但一个工作空间**不可以**绑定多个企业项目。借助工作空间，您可以对不同用户的资源访问和权限做更加细致的约束，具体为如下两种约束：

- 只有被授权的用户才能访问特定的工作空间（在创建、管理工作空间的页面进行配置），这意味着，像数据集、算法等AI资产，均可以借助工作空间做访问的限制。
- 在前文提到的权限授权操作中，如果“选择授权范围方案”时设定为“指定企业项目资源”，那么该授权仅对绑定至该企业项目的工作空间生效。

说明

- 工作空间的约束与权限授权的约束是叠加生效的，意味着对于一个用户，必须同时拥有工作空间的访问权和训练任务的创建权限（且该权限覆盖至当前的工作空间），他才可以在这个空间里提交训练任务。
- 对于已经开通企业项目但没有开通工作空间的用户，其所有操作均相当于在“default”企业项目里进行，请确保对应权限已覆盖了名为default的企业项目。
- 对于未开通企业项目的用户，不受上述约束限制。

本章小结

对于ModelArts的权限管理，总结了如下几条关键点：

- 如果您是个人用户，则不需要考虑细粒度权限问题，您的账户默认具备使用ModelArts的所有权限。
- ModelArts平台的所有功能均通过IAM体系进行了权限管控，您可以通过标准的IAM**授权**动作，来对特定用户进行精细化的权限管控。
- 对于所有用户（包括个人用户），需要完成对ModelArts的**委托**授权（ModelArts > 权限管理 > 添加授权），才能使用特定的功能，否则会造成您的操作出现不可预期的错误。
- 对于开通了企业项目的用户，可以进一步申请开通ModelArts的**工作空间**，通过组合使用基础授权和工作空间，来达成更加复杂的权限控制目的。

7.2 权限控制方式

7.2.1 IAM

介绍ModelArts所有功能涉及到的IAM权限配置。

IAM 权限简介

如果您需要为企业中的员工设置不同的权限访问ModelArts资源，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制云服务资源的访问。如果华为账号已经能满足您的要求，不需要通过IAM对用户进行权限管理，您可以跳过本章节，不影响您使用ModelArts服务的其他功能。

IAM是提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。

通过IAM，您可以通过授权控制他们对服务资源的访问范围。例如您的员工中有负责软件开发的人员，您希望他们拥有ModelArts的使用权限，但是不希望他们拥有删除ModelArts等高危操作的权限，那么您可以使用IAM进行权限分配，通过授予用户仅能使用ModelArts，但是不允许删除ModelArts的权限，控制他们对ModelArts资源的使用范围。

关于IAM的详细介绍，请参见**IAM产品介绍**。

角色与策略权限管理

ModelArts服务支持角色与策略授权。默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

ModelArts部署时通过物理区域划分，为项目级服务。授权时，“授权范围”需要选择“指定区域项目资源”，然后在指定区域（如华北-北京1）对应的项目（cn-north-1）中设置相关权限，并且该权限仅对此项目生效；如果“授权范围”选择“所有资源”，则该权限在所有区域项目中都生效。访问ModelArts时，需要先切换至授权区域。

如**表7-2**所示，包括了ModelArts的所有系统策略权限。如果系统预置的ModelArts权限，不满足您的授权要求，可以创建自定义策略，可参考**策略JSON格式字段介绍**。

表 7-2 ModelArts 系统策略

策略名称	描述	类型
ModelArts FullAccess	ModelArts管理员用户，拥有所有ModelArts服务的权限。	系统策略
ModelArts CommonOperations	ModelArts操作用户，拥有所有ModelArts服务操作权限除了管理专属资源池的权限。	系统策略
ModelArts Dependency Access	ModelArts服务的常用依赖服务的权限。	系统策略

ModelArts对其他云服务有依赖关系，因此在ModelArts控制台的各项功能需要配置相应的服务权限后才能正常查看或使用，依赖服务及其预置的权限如下。

表 7-3 ModelArts 控制台依赖服务的角色或策略

控制台功能	依赖服务	需配置角色/策略
数据管理	对象存储服务OBS	OBS Administrator
	数据湖探索DLI	DLI FullAccess
	MapReduce服务MRS	MRS Administrator

控制台功能	依赖服务	需配置角色/策略
	数据仓库服务 GaussDB(DWS)	DWS Administrator
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
开发环境	对象存储服务OBS	OBS Administrator
	凭据管理服务CSMS	CSMS ReadOnlyAccess
	云审计服务CTS	CTS Administrator
	弹性云服务器ECS	ECS FullAccess
	容器镜像服务SWR	SWR Administrator
	弹性文件服务SFS	SFS Turbo FullAccess
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
训练管理	对象存储服务OBS	OBS Administrator
	消息通知服务SMN	SMN Administrator
	云审计服务CTS	CTS Administrator
	弹性文件服务SFS Turbo	SFS Turbo ReadOnlyAccess
	容器镜像服务SWR	SWR Administrator
	应用运维管理服务 AOM	AOM FullAccess
	密钥管理服务KMS	KMS CMKFullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Workflow	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
自动学习	对象存储服务OBS	OBS Administrator

控制台功能	依赖服务	需配置角色/策略
	云审计服务CTS	CTS Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI应用管理	对象存储服务OBS	OBS Administrator
	企业项目管理服务EPS	EPS FullAccess
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
部署上线	对象存储服务OBS	OBS Administrator
	云监控服务CES	CES ReadOnlyAccess
	消息通知服务SMN	SMN Administrator
	企业项目管理服务EPS	EPS FullAccess
	云审计服务CTS	CTS Administrator
	云日志服务LTS	LTS FullAccess
	虚拟私有云VPC	VPC FullAccess
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI Gallery	对象存储服务OBS	OBS Administrator
	云审计服务CTS	CTS Administrator
	容器镜像服务SWR	SWR Administrator
	AI开发平台ModelArts	ModelArts CommonOperations ModelArts Dependency Access
专属资源池	云审计服务CTS	CTS Administrator
	云容器引擎CCE	CCE Administrator
	裸金属服务器BMS	BMS FullAccess
	镜像服务IMS	IMS FullAccess
	数据加密服务DEW	DEW KeypairReadOnlyAccess
	虚拟私有云VPC	VPC FullAccess
	弹性云服务器ECS	ECS FullAccess
	弹性文件服务SFS	SFS Turbo FullAccess

控制台功能	依赖服务	需配置角色/策略
	对象存储服务OBS	OBS Administrator
	应用运维管理服务 AOM	AOM FullAccess
	标签管理服务TMS	TMS FullAccess
	AI开发平台ModelArts	ModelArtsFullAccess
	费用中心	BSS Administrator

如果系统预置的权限，不满足您的授权要求，可以创建自定义策略。自定义策略中可以添加的授权项（Action）请参考**ModelArts资源权限项**。

目前支持以下两种方式创建自定义策略：

- 可视化视图创建自定义策略：无需了解策略语法，按可视化视图导航栏选择云服务、操作、资源、条件等策略内容，可自动生成策略。
- JSON视图创建自定义策略：可以在选择策略模板后，根据具体需求编辑策略内容；也可以直接在编辑框内编写JSON格式的策略内容。

具体创建步骤请参见：**创建自定义策略**。下面为您介绍常用的ModelArts自定义策略样例。

- 示例1：授权镜像管理的权限。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- 示例2：拒绝用户创建、更新、删除专属资源池。

  拒绝策略需要同时配合其他策略使用，否则没有实际作用。用户被授予的策略中，一个授权项的作用如果同时存在Allow和Deny，则遵循**Deny优先原则**。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "modelarts:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "swr:*:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
```

```
            "smn:*:*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "modelarts:pool:create",
            "modelarts:pool:update",
            "modelarts:pool:delete"
        ],
        "Effect": "Deny"
    }
  ]
}
```

- 示例3：多个授权项策略。

  一个自定义策略中可以包含多个授权项，且除了可以包含本服务的授权项外，还可以包含其他服务的授权项，可以包含的其他服务必须跟本服务同属性，即都是项目级服务或都是全局级服务。多个授权语句策略描述如下：

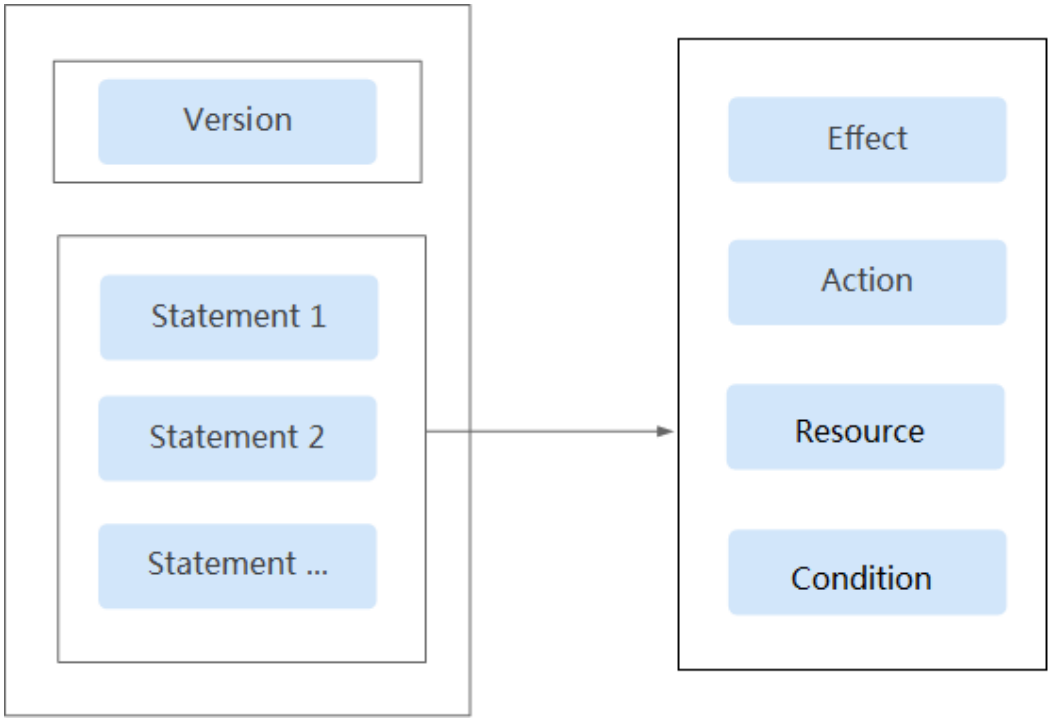
```
{
  "Version": "1.1",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "modelarts:service:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lts:logs:list"
        ]
    }
  ]
}
```

策略 JSON 格式字段介绍

策略结构

策略结构包括Version（策略版本号）和Statement（策略权限语句）两部分，其中Statement可以有多个，表示不同的授权项。

图 7-3 策略结构



策略参数

下面介绍策略参数详细说明。了解策略参数后，您可以根据场景自定义策略。具体可以参考文档**自定义策略使用样例**。

表 7-4 策略参数说明

参数		含义	值
Version		策略的版本。	1.1：代表基于策略的访问控制。
Statement：策略的授权语句	Effect：作用	定义Action中的操作权限是否允许执行。	• Allow：允许执行。 • Deny：不允许执行。 **说明** 当同一个Action的Effect既有Allow又有Deny时，遵循Deny优先的原则。
	Action：授权项	操作权限。	格式为“服务名:资源类型:操作”。授权项支持通配符号*，通配符号*表示所有。 示例： "modelarts:notebook:list"：表示查看Notebook实例列表权限，其中modelarts为服务名，notebook为资源类型，list为操作。 您可以在对应服务“API参考”资料中查看该服务所有授权项。

参数		含义	值
	Condition：条件	使策略生效的特定条件，包括**条件键**和**运算符**。	格式为“条件运算符:{条件键：[条件值1,条件值2]}”。 如果您设置多个条件，同时满足所有条件时，该策略才生效。 示例: "StringEndWithIfExists":{"g:UserName":["specialCharacter"]}：表示当用户输入的用户名以"specialCharacter"结尾时该条statement生效。
	Resource: 资源类型	策略所作用的资源。	格式为“服务名:<region>:<account-id>:资源类型:资源路径”，资源类型支持通配符号*，通配符号*表示所有。 **说明** ModelArts的授权不支持指定具体资源路径。

ModelArts 资源类型

管理员可以按ModelArts的资源类型选择授权范围。ModelArts支持的资源类型如下表：

表 7-5 ModelArts 资源类型（角色与策略授权）

资源类型	说明
notebook	开发环境的Notebook实例
exemlProject	自动学习项目
exemlProjectInf	自动学习项目的在线推理服务
exemlProjectTrain	自动学习项目的训练作业
exemlProjectVersion	自动学习项目的版本
workflow	Workflow项目
pool	专属资源池
network	专属资源池网络连接
trainJob	训练作业
trainJobLog	训练作业的运行日志
trainJobInnerModel	系统预置模型
model	模型
service	在线服务
nodeservice	边缘服务

资源类型	说明
workspace	工作空间
dataset	数据集
dataAnnotation	数据集的标注信息
aiAlgorithm	训练算法
image	镜像
devserver	弹性裸金属

ModelArts 资源权限项

参考《 ModelArts API参考 》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**
- **工作空间管理权限**

7.2.2 依赖和委托

功能依赖

功能依赖策略项

您在使用ModelArts的过程中，需要和其他云服务交互，比如需要在提交训练作业时选择指定数据集OBS路径和日志存储OBS路径。因此管理员在为用户配置细粒度授权策略时，需要同时配置依赖的权限项，用户才能使用完整的功能。

说明

- 如果您使用根用户（与账户同名的缺省子用户）使用ModelArts，根用户默认拥有所有权限，不再需要单独授权。
- 请用户确保当前用户具备委托授权中包含的依赖策略项权限。例如，用户给ModelArts的委托需要授权SWR Admin权限，需要保证用户本身具备SWR Admin权限。

表 7-6 基本配置

业务场景	依赖的服务	依赖策略项	支持的功能
全局配置	IAM	iam:users:listUsers	查询用户列表（仅管理员需要）
基本功能	IAM	iam:tokens:assume	使用委托获取用户临时认证凭据（必需）

业务场景	依赖的服务	依赖策略项	支持的功能
基本功能	BSS	bss:balance:view	在ModelArts控制台创建资源后，页面展示账号当前余额

表 7-7 管理工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
工作空间	IAM	iam:users:listUsers	按用户进行工作空间授权
	ModelArts	modelarts:*:delete*	删除工作空间时，同时清理空间内的资源

表 7-8 管理开发环境 Notebook

业务场景	依赖的服务	依赖策略项	支持的功能
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get	实例的启动、停止、创建、删除、更新等依赖的权限。
	AOM	aom:metric:get aom:metric:list aom:alarm:list	
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	

业务场景	依赖的服务	依赖策略项	支持的功能
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list	
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子用户对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子用户创建的。
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。
	IAM	iam:users:listUsers	

业务场景	依赖的服务	依赖策略项	支持的功能
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。

表 7-9 管理训练作业

业务场景	依赖的服务	依赖策略项	支持的功能
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。
		modelarts:workspace:getQuota	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。
	SWR	SWR Administrator	使用自定义镜像运行训练作业。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。

表 7-10 使用 Workflow

业务场景	依赖的服务	依赖策略项	支持的功能
使用数据集	ModelArts	modelarts:dataset:getDataset modelarts:dataset:createDataset modelarts:dataset:createDatasetVersion modelarts:dataset:createImportTask modelarts:dataset:updateDataset modelarts:processTask:createProcessTask modelarts:processTask:getProcessTask modelarts:dataset:listDatasets	在工作流中使用ModelArts数据集

业务场景	依赖的服务	依赖策略项	支持的功能
管理AI应用	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	在工作流中管理ModelArts AI应用
部署上线	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	在工作流中管理ModelArts在线服务
训练作业	ModelArts	modelarts:trainJob:get modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJobVersion:list modelarts:trainJobVersion:create modelarts:trainJob:delete modelarts:trainJobVersion:delete modelarts:trainJobVersion:stop	在工作流中管理ModelArts训练作业
工作空间	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	在工作流中使用ModelArts工作空间

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	在工作流中使用OBS数据
工作流运行	IAM	iam:users:listUsers（查询用户列表） iam:agencies:getAgency（查询指定委托详情） iam:tokens:assume（获取委托Token）	在工作流运行时，调用ModelArts其他服务
集成DLI	DLI	dli:jobs:get（查询作业详情） dli:jobs:list_all（查询作业列表） dli:jobs:create（创建新作业）	在工作流中集成DLI

业务场景	依赖的服务	依赖策略项	支持的功能
集成MRS	MRS	mrs:job:get（查询作业详情） mrs:job:submit（创建并执行作业） mrs:job:list（查询作业列表） mrs:job:stop（停止作业） mrs:job:batchDelete（批量删除作业） mrs:file:list（查询文件列表）	在工作流中集成MRS

表 7-11 管理 AI 应用

业务场景	依赖的服务	依赖策略项	支持的功能
管理AI应用	SWR	swr:repository:deleteRepository swr:repository:deleteTag swr:repository:getRepository swr:repository:listTags	从自定义镜像导入 从OBS导入时使用自定义引擎

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:ListAllMybuckets（获取桶列表） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:ListBucket（列举桶内对象） obs:bucket:GetBucketLocation（获取桶区域位置） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:GetObjectVersion（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:object:DeleteObjectVersion（删除对象、批量删除对象） obs:object:ListMultipartUploadParts（列举已上传的段） obs:object:AbortMultipartUpload（取消多段上传任务） obs:object:GetObjectAcl（获取对象ACL） obs:object:GetObjectVersionAcl（获取对象ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:object:PutObjectAcl（设置对象ACL）	从OBS导入模型 模型转换指定OBS路径

表 7-12 管理部署上线

业务场景	依赖的服务	依赖策略项	支持的功能
在线服务	LTS	lts:logs:list（查询日志列表）	查询和展示LTS日志

业务场景	依赖的服务	依赖策略项	支持的功能
	OBS	obs:bucket:GetBucketPolicy （获取桶策略） obs:bucket:HeadBucket （获取桶元数据） obs:bucket:ListAllMyBuckets （获取桶列表） obs:bucket:PutBucketPolicy （设置桶策略） obs:bucket:DeleteBucketPolicy （删除桶策略）	服务运行时容器挂载外部存储卷
批量服务	OBS	obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:bucket:CreateBucket（创建桶） obs:bucket:ListBucket（列举桶内对象） obs:bucket:ListAllMyBuckets（获取桶列表）	创建批量服务，批量推理。
边缘服务	CES	ces:metricData:list（查询指标数据）	查看服务的监控指标
	IEF	ief:deployment:delete（删除应用部署）	管理边缘服务

表 7-13 管理数据集

业务场景	依赖的服务	依赖策略项	支持的功能
管理数据集和标注	OBS	obs:bucket:ListBucket（列举桶内对象） obs:object:GetObject（获取对象内容、获取对象元数据） obs:object:PutObject（PUT上传、POST上传、复制对象、追加写对象、初始化上传段任务、上传段、合并段） obs:object:DeleteObject（删除对象、批量删除对象） obs:bucket:HeadBucket（获取桶元数据） obs:bucket:GetBucketAcl（获取桶ACL） obs:bucket:PutBucketAcl（设置桶ACL） obs:bucket:GetBucketPolicy（获取桶策略） obs:bucket:PutBucketPolicy（设置桶策略） obs:bucket:DeleteBucketPolicy（删除桶策略） obs:bucket:PutBucketCORS（设置桶的CORS配置、删除桶的CORS配置） obs:bucket:GetBucketCORS（获取桶的CORS配置） obs:object:PutObjectAcl（设置对象ACL）	管理OBS中的数据集 标注OBS数据 创建数据管理作业
管理表格数据集	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describe_table	在数据集中管理DLI数据
管理表格数据集	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail	在数据集中管理DWS数据
管理表格数据集	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	在数据集中管理MRS数据

业务场景	依赖的服务	依赖策略项	支持的功能
智能标注	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	使用智能标注
团队标注	IAM	iam:projects:listProjects（查询租户项目） iam:users:listUsers（查询用户列表） iam:agencies:createAgency（创建委托） iam:quotas:listQuotasForProject（查询指定项目的配额）	管理标注团队

表 7-14 资源管理

业务场景	依赖的服务	依赖策略项	支持的功能
资源池管理	BSS	bss:coupon:view bss:order:view bss:balance:view bss:discount:view bss:renewal:view bss:bill:view bss:contract:update bss:order:pay bss:unsubscribe:update bss:renewal:update bss:order:update	资源池的创建、续费、退订等与计费相关的功能。依赖权限需要配置在IAM项目视图中。
	CCE	cce:cluster:list cce:cluster:get	获取CCE集群列表、集群详情、集群证书等信息。依赖权限需要配置在IAM项目视图中。

业务场景	依赖的服务	依赖策略项	支持的功能
	KMS	kms:cmk:list kms:cmk:getMaterial	获取用户创建的密钥对列表信息。依赖权限需要配置在IAM项目视图中。
	AOM	aom:metric:get	获取资源池的监控数据。依赖权限需要配置在IAM项目视图中。
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion	获取AI诊断日志。依赖权限需要配置在IAM项目视图中。
	ECS	ecs:availabilityZones:list	查询可用区列表。依赖权限需要配置在IAM项目视图中。

业务场景	依赖的服务	依赖策略项	支持的功能
网络管理	VPC	vpc:routes:create vpc:routes:list vpc:routes:get vpc:routes:delete vpc:peerings:create vpc:peerings:accept vpc:peerings:get vpc:peerings:delete vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:vpcs:create vpc:vpcs:list vpc:vpcs:get vpc:vpcs:delete vpc:subnets:create vpc:subnets:get vpc:subnets:delete vpcep:endpoints:list vpcep:endpoints:create vpcep:endpoints:delete vpcep:endpoints:get vpc:ports:create vpc:ports:get vpc:ports:update vpc:ports:delete vpc:networks:create vpc:networks:get vpc:networks:update vpc:networks:delete	ModelArts网络资源创建和删除、VPC网络打通。依赖权限需要配置在IAM项目视图中。
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	用户的网络和SFS Turbo资源打通。依赖权限需要配置在IAM项目视图中。

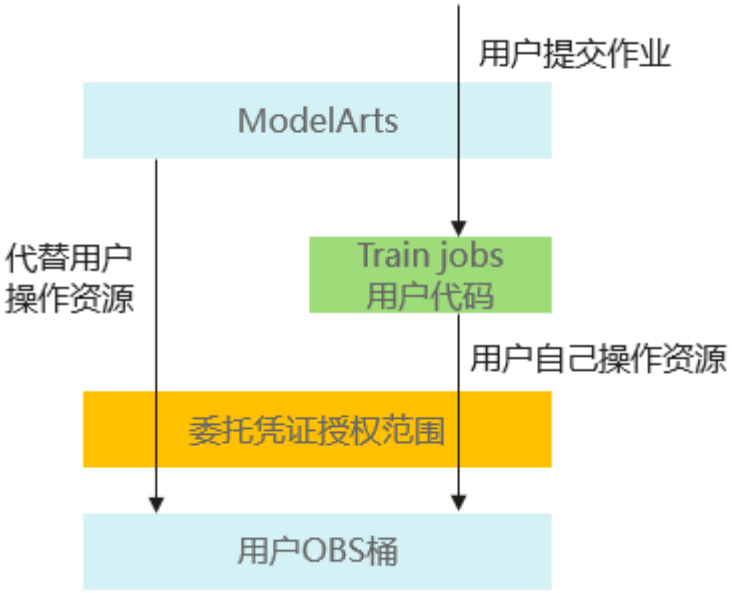
业务场景	依赖的服务	依赖策略项	支持的功能
边缘资源池	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	边缘池增删改查管理

委托授权

用户在使用ModelArts服务的过程中，为了简化用户的操作，ModelArts后台可以代替用户完成一些工作，如训练作业启动前自动下载用户OBS桶中的数据集到作业空间、自动转储训练作业日志到用户OBS桶中。

ModelArts服务不会保存用户的Token认证凭据，在后台异步作业中操作用户的资源（如OBS桶）前，需要用户通过IAM委托向ModelArts显式授权，ModelArts在需要时使用用户的委托获取临时认证凭据用于操作用户资源，见“**添加授权**”。

图 7-4 委托授权



如图7-4所示，用户向ModelArts授权后，ModelArts使用委托授权的临时凭证访问和操作用户资源，协助用户自动化一些繁琐和耗时的操作。同时，委托凭证会同步到用户的作业中（Notebook实例和训练作业），客户在作业中可以使用委托凭证自行访问自己的资源。

在ModelArts服务中委托授权有两种方式：

1、一键式委托授权

ModelArts提供了一键式自动授权功能，用户可以在ModelArts的权限管理功能中，快速完成委托授权，由ModelArts为用户自动创建委托并配置到ModelArts服务中。

这种方式为保证使用业务过程中有足够的权限，基于依赖服务的预置系统策略指定授权范围，创建的委托的权限比较大，基本覆盖了依赖服务的全部权限。如果您需要对委托授权的权限范围进行精确控制，请使用第二种方式。

2、定制化委托授权

管理员在IAM中为不同用户创建不同的委托授权策略，再到ModelArts中为用户配置已创建好的委托。管理员在IAM中为用户创建委托时，根据用户的实际权限范围为委托指定最小权限范围，控制用户在使用ModelArts过程中可访问的资源内容。具体参考**配置ModelArts基本使用权限**。

委托授权的越权风险

可以看到用户的委托授权是独立的，理论上用户的委托授权范围是可以超出用户自身用户组的授权策略的授权范围，如果配置不当就会出现用户越权的问题。

为了控制委托授权的越权风险，ModelArts服务的权限管理功能要求只有租户管理员才能为用户配置委托，由管理员保证委托授权的安全性。

委托授权的最小化

管理员在配置委托授权时，应严格控制授权的范围。

ModelArts为用户异步自动化完成作业的准备、清理等操作，所需的委托授权内容是基础授权范围。如果用户只使用ModelArts的部分功能，管理员可以依据委托授权表格的说明屏蔽不使用的基础权限项。相反地，如果用户需要在作业中使用基础授权范围外的资源权限，管理员也可以为用户在委托授权中增加新的权限项。总之，委托授权的范围应该基于实际业务场景所需权限范围来进行定制，保持委托授权范围的最小化。

委托基础授权范围

当您需要定制委托授权的权限列表时，请参考下面表格，根据实际业务选择授权项。

表 7-15 开发环境基础委托授权

业务场景	依赖的服务	委托授权项	说明
Notebook实例中操作OBS数据。	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	您可通过以下方式在Notebook实例中操作OBS中的数据： • 通过ModelArts SDK操作OBS数据。 • 通过Notebook文件上传功能操作OBS数据。 • 通过在Console页面添加OBS桶到Notebook实例的/data目录下，以文件方式操作OBS数据。
Notebook实例事件上报。	AOM	aom:alarm:put	在Notebook实例的生命周期中，部分事件会上报到用户AOM账号下。
VPC与Notebook实例网络互联。	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnet:get	Notebook实例中新增一个可以与用户指定VPC的子网的网卡，用于与用户VPC下的服务进行网络互联。
VS Code一键连接Notebook。	ModelArts	modelarts:notebook:get	用于管理Notebook实例信息，点击VS Code插件时，获取实例详情信息，以方便将SSH配置信息写入本地VS Code。
停止Notebook实例。	ModelArts	modelarts:notebook:stop	用于停止运行中的Notebook实例。
更新Notebook实例自动停止时间。	ModelArts	modelarts:notebook:updateStopPolicy	用于更新Notebook实例的自动停止时间。

业务场景	依赖的服务	委托授权项	说明
OBS并行文件系统场景下使用MindInsight/TensorBoard可视化工具。	ModelArts	modelarts:notebook:umountStorage modelarts:notebook:getMountedStorage modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage	在开发环境Notebook实例中开启MindInsight/TensorBoard可视化工具，且需要访问的是OBS并行文件系统时，需要配置左侧的权限。

表 7-16 训练作业基础委托授权

业务场景	依赖的服务	委托授权项	说明
训练作业访问OBS文件。	OBS	obs:bucket:HeadBucket obs:bucket:GetBucketLocation obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:GetObject obs:object:GetObjectVersion obs:object:GetObjectAcl obs:object:GetObjectVersionAcl	训练作业配置代码目录、输入、输出和日志的OBS桶路径时，需要OBS服务相关操作权限，用于OBS对象路径的合法性校验。
训练作业以自定义容器镜像方式启动。	SWR	SWR Administrator	训练作业以自定义容器镜像方式启动时，需要获取用户SWR容器镜像的临时登录指令，用于下载容器镜像。SWR共享版不支持细粒度权限项，因此需要配置Administrator权限。
训练作业状态变化通知。	SMN	smn:template:list smn:template:create smn:topic:list smn:topic:publish	若要配置训练作业状态变化通知，需要SMN服务相关操作权限，用于发送模板化的消息通知。
训练作业配置挂载SFS Turbo。	SFS Turbo	SFS Turbo ReadOnlyAccess	训练作业配置挂载SFS Turbo时，需要SFS Turbo读权限，以通过SFS Turbo ID获取其详情。

业务场景	依赖的服务	委托授权项	说明
审计日志上报。	CTS	CTS Administrator	配置CTS服务权限，用于上报事件。CTS服务当前上报事件功能未支持细粒度权限项，因此需要配置Administrator权限（**CTS细粒度权限说明**）。

表 7-17 推理部署基础委托授权

业务场景	依赖的服务	委托授权项	说明
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	建议配置，在线服务配置LTS日志上报。
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	使用批量服务时必须配置。
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	使用边缘服务时必须配置，通过IEF部署边缘服务。
从OBS导入AI应用。	OBS	obs:object:DeleteObject obs:object:GetObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	必须配置。若有使用并行文件系统，则需额外配置obs:bucket:HeadBucket。
从容器镜像中导入AI应用。	SWR	swr:repository:getRepository swr:repository:updateRepository swr:instance:listDomainNames swr:repository:getNamespace	必须配置。

业务场景	依赖的服务	委托授权项	说明
使用ModelArts Edge功能。	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	可选配置，如果使用ModelArts Edge功能需要配置。
AOM指标告警事件	AOM	aom:log:get aom:alarm:get aom:metric:put aom:alarm:put aom:event:put aom:event:list aom:event:get	建议配置，若需要AOM查看告警事件则需要配置。
监控指标上报CES	CES	CES ReadOnlyAccess ces:metricMeta:create	建议配置，监控指标上报CES。
企业项目	EPS	EPS ReadOnlyAccess	可选配置，如果企业项目则需要配置。
消息订阅推送	SMN	smn:topic:list smn:topic:publish smn:application:publish	可选配置，如果使用消息订阅推送功能需要配置。

表 7-18 数据管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
使用数据标注、数据处理功能。	ModelArts	modelarts:trainJob:create modelarts:trainJob:update modelarts:trainJob:delete modelarts:trainJob:get modelarts:trainJob:list modelarts:trainJob:previewLog modelarts:aiAlgorithm:get modelarts:aiAlgorithm:get modelarts:model:get modelarts:service:list modelarts:model:create modelarts:workspace:list modelarts:workspace:get modelarts:trainJobInnerModel:list	必须配置，使用数据标注、数据处理功能时会进行创建训练作业、查询训练作业、算法查询等操作。
访问OBS数据	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:bucket:HeadBucket obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:GetBucketPolicy obs:bucket:PutBucketPolicy obs:bucket:DeleteBucketPolicy obs:bucket:PutBucketCORS obs:bucket:GetBucketCORS obs:object:PutObjectAcl	必须配置，在OBS中存储、查询、删除数据。

业务场景	依赖的服务	委托授权项	说明
访问DLI数据。	DLI	dli:queue:createQueue dli:queue:dropQueue dli:queue:scaleQueue dli:queue:submitJob dli:database:displayDatabase dli:database:displayAllTables dli:table:describeTable dli:table:showPrivileges dli:jobs:grantPrivilege dli:table:dropTable	可选配置，如果访问DLI数据需要配置。
访问MRS数据。	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:file:list	可选配置，如果访问MRS数据需要配置。
访问DWS数据。	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail	可选配置，如果访问DWS数据需要配置。

表 7-19 专属资源池管理基础委托授权

业务场景	依赖的服务	委托授权项	说明
通过关联sfsturbo功能实现专属资源池和SFS Turbo资源打通。	SFS Turbo	sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic	使用该特性时需要配置。
用户的ModelArts网络和VPC进行打通，同时添加相关路由。	VPC	vpc:vpcs:get vpc:subnets:get vpc:peerings:accept vpc:routers:create vpc:routes:delete vpc:routes:get vpc:routeTables:update vpc:routeTables:get	使用该特性时需要配置。

业务场景	依赖的服务	委托授权项	说明
使用ModelArts Lite Cluster资源池。	CCE APM	cce:cluster:get cce:node:get cce:node:list cce:job:get cce:node:create cce:node:delete cce:node:remove cce:addonInstance:get cce:addonInstance:list cce:addonInstance:create cce:addonInstance:update cce:addonInstance:delete apm:icmgr:create	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式管理用户的CCE集群，同步集群信息、纳管节点等。
	ECS BMS EVS DEW	ecs:cloudServers:create ecs:cloudServers:delete ecs:cloudServers:get ecs:cloudServers:start ecs:cloudServers:stop ecs:cloudServers:reboot ecs:cloudServers:redeploy ecs:cloudServers:listServerInterfaces ecs:cloudServers:changeVpc ecs:cloudServerFlavors:get ecs:quotas:get ecs:cloudServers:batchSetServerTags bms:servers:create bms:serverFlavors:get evs:types:get evs:volumes:list evs:quotas:get kps:domainKeypairs:get	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts通过委托的方式对用户的BMS/ECS节点进行生命周期的管理。

业务场景	依赖的服务	委托授权项	说明
	IMS	ims:images:get ims:images:share	使用ModelArts Lite Cluster资源池时必须配置。 ModelArts Lite Cluster专属资源池节点创建在用户账号下，创建前需要将节点系统镜像共享给用户账号。

表 7-20 Workflow 基础委托授权

业务场景	依赖的服务	委托授权项	说明
对ModelArts数据管理、训练、推理等服务操作。	ModelArts	modelarts:workspace:getQuotas modelarts:service:get modelarts:app:get modelarts:pool:get modelarts:model:get modelarts:trainJob:get modelarts:dataset:get modelarts:dataAnnotation:get modelarts:workspace:get modelarts:aiAlgorithm:get modelarts:autoAnnotation:get modelarts:trainJobVersion:delete modelarts:dataset:delete modelarts:workspace:delete modelarts:trainJobVersion:create modelarts:dataset:import modelarts:trainJob:delete modelarts:service:delete modelarts:dataset:publishVersion modelarts:app:create modelarts:service:create modelarts:service:update modelarts:aiAlgorithm:create modelarts:trainJobVersion:stop modelarts:workspace:update modelarts:dataset:deleteVersion modelarts:trainJob:create modelarts:model:delete modelarts:model:create modelarts:dataset:create modelarts:app:delete modelarts:aiAlgorithm:delete modelarts:service:action modelarts:app:update modelarts:trainJobVersion:list modelarts:model:list modelarts:app:list	用于在Workflow中调用MA的数据管理、训练、推理等服务，创建相应的实例。 建议在配置委托权限时，直接配置ModelArts CommonOperations权限即可。

业务场景	依赖的服务	委托授权项	说明
		modelarts:service:list modelarts:dataset:list modelarts:workspace:list modelarts:trainJob:list modelarts:aiAlgorithm:list	
Workflow中对OBS数据操作。	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:PutObjectVersionAcl obs:object:PutObjectAcl obs:object:GetObjectAcl obs:object:DeleteObject obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketPolicy obs:bucket:DeleteBucketPolicy obs:bucket:GetBucketPolicy	可选配置，在工作流中使用OBS数据时需要配置。
Workflow中操作DLI相关的任务。	DLI	dli:jobs:create dli:jobs:delete dli:jobs:get dli:jobs:listAll dli:jobs:stop	可选配置，如果在Workflow中涉及DLI相关节点需要配置。
Workflow中操作MRS相关的任务	MRS	mrs:job:get mrs:cluster:get mrs:file:list mrs:cluster:list mrs:job:stop mrs:job:submit mrs:job:delete	可选配置，如果在Workflow中涉及MRS相关节点需要配置。

业务场景	依赖的服务	委托授权项	说明
Workflow中使用SMN消息订阅功能。	SMN	smn:topic:list smn:topic:publish	可选配置，使用SMN消息订阅功能时必须配置。

7.2.3 工作空间

ModelArts的用户需要为不同的业务目标开发算法、管理和部署模型，此时可以创建多个工作空间，把不同应用开发过程的输出内容划分到不同工作空间中，便于管理和使用。

工作空间支持3种访问控制：

- PUBLIC：租户（主账号和所有子账号）内部公开访问。
- PRIVATE：仅创建者和主账号可访问。
- INTERNAL：创建者、主账号、指定IAM子账号可访问当授权类型为INTERNAL时需要指定可访问的子账号的账号名，可选择多个。

每个账号每个IAM项目都会分配1个默认工作空间，默认工作空间的访问控制为PUBLIC。

通过工作空间的访问控制能力，可限制仅允许部分人访问对应的工作空间。通过此功能可实现类似如下场景：

- **教育场景**：老师可给每个学生分配1个INTERNAL的工作空间并且限制该工作空间被指定学生访问，这样可使得学生可独立完成在ModelArts上的实验。
- **企业场景**：管理者可创建用于生产任务的工作空间并限制仅让运维人员使用，用于日常调试的工作空间并限制仅让开发人员使用。通过这种方式让不同的企业角色只能在指定工作空间下使用资源。

目前工作空间功能是“受邀开通”状态，作为企业用户您可以通过您对口的技术支持申请开通。

7.3 典型场景配置实践

7.3.1 个人用户快速配置 ModelArts 访问权限

ModelArts使用过程中涉及到OBS、SWR等服务交互，需要用户配置委托授权，允许ModelArts访问这些依赖服务。如果没有授权，ModelArts的部分功能将不能正常使用。

约束与限制

- 只有主账号可以使用委托授权，可以为当前账号授权，也可以为当前账号下的所有IAM用户授权。

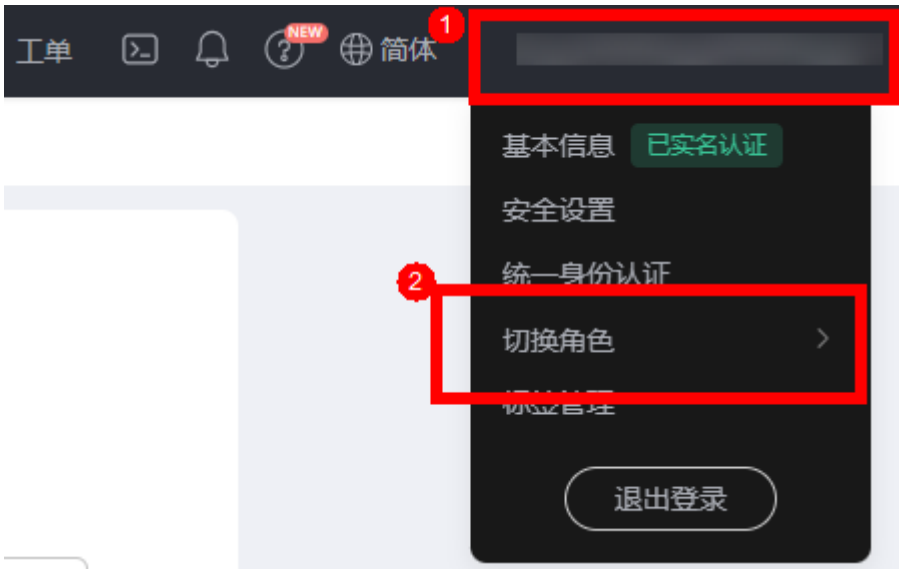
- 多个IAM用户或账号，可使用同一个委托。
- 一个账号下，最多可创建50个委托。
- 对于首次使用ModelArts新用户，请直接新增委托即可。一般用户新增普通用户权限即可满足使用要求。如果有精细化权限管理的需求，可以自定义权限按需设置。
- 如果未获得委托授权，当打开“访问授权”页面时，ModelArts会提醒您当前用户未配置授权，需联系此IAM用户的管理员账号进行委托授权。

添加授权

1. 登录ModelArts管理控制台，在左侧导航栏选择“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”，进入“访问授权”配置页面，根据参数说明进行配置。

表 7-21 参数说明

参数	说明
“授权对象类型”	包括IAM子用户、联邦用户、委托用户和所有用户。 ● IAM子用户：由主账号在IAM中创建的用户，是服务的使用人员，具有独立的身份凭证（密码和访问密钥），根据账号授予的权限使用资源。IAM子用户相关介绍请参见**IAM用户介绍**。 ● 联邦用户：又称企业虚拟用户。联邦用户相关介绍请参见**联邦身份认证**。 ● 委托用户：IAM中创建的一个委托。IAM创建委托相关介绍请参见**创建委托**。 ● 所有用户：该选项表示会将委托的权限授权到当前账号下的所有子账号、包括未来创建的子账号，授权范围较大，需谨慎使用。个人用户选择“所有用户”即可。

参数	说明
“授权对象”	“授权对象类型”选择“所有用户”时不涉及此参数。 • IAM子用户：选择指定的IAM子用户，给指定的IAM子用户配置委托授权。 **图 7-5** 选择 IAM 子用户  • 联邦用户：输入联邦用户的用户名或用户ID。 **图 7-6** 选择联邦用户  • 委托用户：选择委托名称。使用账号A创建一个权限委托，在此处将该委托授权给账号B拥有的委托。在使用账号B登录控制台时，可以在控制台右上角的个人账号切换角色到账号A，使用账号A的委托权限。 **图 7-7** 委托用户切换角色 
“委托选择”	• 已有委托：列表中如果已有委托选项，则直接选择一个可用的委托为上述选择的用户授权。单击委托名称查看该委托的权限详情。 • 新增委托：如果没有委托可选，可以在新增委托中创建委托权限。对于首次使用ModelArts的用户，需要新增委托。
“新增委托 > 委托名称”	系统自动创建委托名称，用户可以手动修改。

参数	说明
“新增委托 > 授权方式”	• 角色授权：IAM最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。由于华为云各服务之间存在业务依赖关系，因此给用户授予角色时，可能需要一并授予依赖的其他角色，才能正确完成业务。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。 • 策略授权：IAM最新提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。 角色与策略相关介绍请参考**权限基本概念**。
“新增委托 > 权限配置 > 普通用户”	普通用户包括用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。 可以单击“查看权限列表”，查看普通用户权限。
“新增委托 > 权限配置 > 自定义”	如用户有精细化权限管理的需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需要在权限列表中勾选要配置的权限。

3. 然后勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，即可完成委托配置。

查看授权的权限列表

用户可以在“权限管理”页面的授权列表中，查看已经配置的委托授权内容。单击授权内容列的“查看权限”，可以查看该授权的权限详情。

图 7-8 查看权限

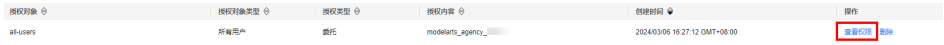
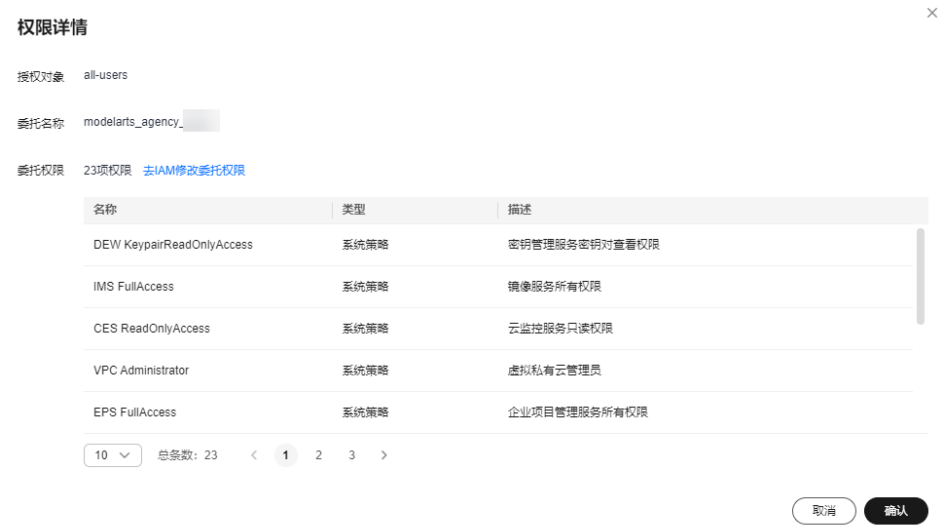


图 7-9 普通用户权限列表



7.3.2 配置 ModelArts 基本使用权限

7.3.2.1 场景描述

ModelArts作为顶层服务，其部分功能依赖于其他服务的访问权限。本章节主要介绍对于IAM子用户使用ModelArts时，如何根据需要开通的功能配置子用户相应权限。

权限列表

子用户的权限，由主用户来控制，主用户通过IAM的权限配置功能设置用户组的权限，从而控制用户组内的子用户的权限。此处的授权列表均按照ModelArts和其他服务的系统预置策略来举例。

表 7-22 服务授权列表

待授权的服务	授权说明	IAM权限设置	是否必选
ModelArts	授予子用户使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。	ModelArts CommonOperations	必选
	如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。	ModelArts FullAccess	可选 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
OBS对象存储服务	授予子用户使用OBS服务的权限。ModelArts的数据管理、开发环境、训练作业、模型推理部署均需要**通过OBS进行数据中转**。	OBS OperateAccess	必选
SWR容器镜像仓库	授予子用户使用SWR服务权限。ModelArts的**自定义镜像功能**依赖镜像服务SWR FullAccess权限。	SWR OperateAccess	必选
密钥管理服务	当子用户使用ModelArts **Notebook的SSH远程功能**时，需要配置子用户密钥管理服务的使用权限。	KMS CMKFullAccess	可选

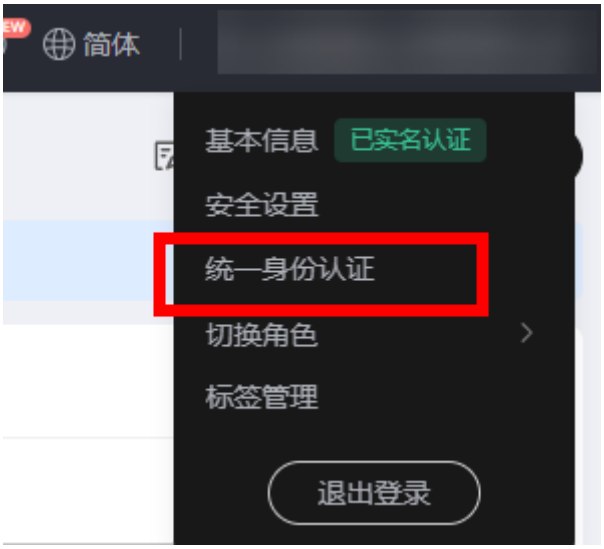
待授权的服务	授权说明	IAM权限设置	是否必选
IEF智能边缘平台	授予子用户智能边缘平台使用权限，ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。	Tenant Administrator	可选
CES云监控	授予子用户使用CES云监控服务的权限。通过CES云监控可以查看ModelArts的在线服务和对应模型负载运行状态的整体情况，并设置监控告警。	CES FullAccess	可选
SMN消息服务	授予子用户使用SMN消息服务的权限。SMN消息通知服务配合CES监控告警功能一起使用。	SMN FullAccess	可选
VPC虚拟私有云	子用户在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。	VPC FullAccess	可选
SFS弹性文件服务	授予子用户使用SFS服务的权限，ModelArts的专属资源池中可以挂载SFS系统作为开发环境或训练的存储。	SFS Turbo FullAccess SFS FullAccess	可选

7.3.2.2 Step1 创建用户组并加入用户

主用户账号下面可以创建多个子用户，并对子用户的权限进行分组管理。此步骤介绍如何创建用户组、子用户、并将子用户加入用户组中。

1. 主用户登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

图 7-10 统一身份认证



2. 创建用户组。在左侧菜单栏中，选择“用户组”。单击右上角“创建用户组”，在“用户组名称”中填入“用户组02”，然后单击“确定”完成用户组创建。

   创建完成后，返回用户组列表。通过用户组管理，将已有子用户加入到用户组中。如果没有子用户账号，可以创建子用户并加入用户组。

3. 创建子用户账号并加入用户组。在IAM左侧菜单栏中，选择“用户”，单击右上角“创建用户”，在“创建用户”页面中，添加多个用户。

   请根据界面提示，填写必选参数，然后单击“下一步”。

4. 在“加入用户组”步骤中，选择“用户组02”，然后单击“创建用户”。

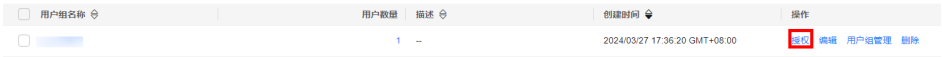
   系统将逐步创建好前面设置的2个用户。

7.3.2.3 Step2 为用户配置云服务使用权限

主用户为子用户授予ModelArts、OBS等云服务的使用权限后，子用户才可以使用这些云服务。此步骤介绍如何为用户组中的所有子用户授予使用ModelArts、OBS、SWR等各类云服务的权限。

1. 主用户在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。

   图 7-11 为用户组授权



2. 配置授权前，请先了解ModelArts各模块使用到的最小权限要求，如表7-22所示。

3. 配置ModelArts使用权限。在搜索框搜索ModelArts。ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。

   选择说明如下：

   - ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。
   - 如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。

4. 配置OBS使用权限。搜索OBS，勾选“ OBS Administrator”。ModelArts训练作业中需要依赖OBS作为数据中转站，需要配置OBS的使用权限。
5. 配置SWR使用权限。搜索SWR，勾选“SWR FullAccess”。ModelArts的自定义镜像功能依赖镜像服务SWR FullAccess权限。
6. （可选）配置密钥管理权限。如果需要使用ModelArts Notebook的SSH访问功能，依赖密钥管理权限。搜索DEW，勾选“DEW KeypairFullAccess”。

   此处需要注意以下Region配置的是DEW密钥管理权限：华北-北京一、华北-北京四、华东-上海一、华东-上海二、华南-广州、西南-贵阳一、中国-香港、亚太-新加坡。其他Region配置的是KMS密钥管理权限。本示例中使用“华南-广州”Region举例，所以需要配置DEW密钥管理权限。
7. （可选）配置智能边缘平台使用权限。ModelArts的边缘服务依赖智能边缘平台，要求配置Tenant Administrator权限。

   注意：Tenant Administrator权限比较大，包含全部云服务的管理权限，而不仅是使用ModelArts服务。请谨慎配置。
8. （可选） 配置CES云监控和SMN消息通知使用权限。ModelArts推理部署的在线服务详情页面内有调用次数详情，单击可查看该在线服务的调用次数随时间详细分布的情况。如果想进一步通过CES云监控查看ModelArts的在线服务和对应模型负载运行状态的整体情况，需要给子用户授予CES权限。

   如果只是查看监控，给子用户授予CES ReadOnlyAccess权限即可。

   如果还需要在CES上设置监控告警，则需要再加上CES FullAccess权限，以及SMN消息通知权限。
9. （可选）配置VPC权限。如果用户在创建专属资源池过程中，需要开启自定义网络配置，此处需要授予用户VPC权限。
10. （可选）配置SFS和SFS Turbo权限。如果用户在专属资源池中挂载SFS系统作为开发环境或训练的存储时，需要授予使用权限。
11. 单击左上角的“查看已选”，确认已勾选的权限。
12. 再单击“下一步”，设置最小授权范围。单击“指定区域项目资源”，勾选待授权使用的区域，单击“确定”。
13. 提示授权成功，查看授权信息，单击“完成”。此处的授权生效需要15-30分钟。

7.3.2.4 Step3 为用户配置 ModelArts 的委托访问授权

配置完IAM权限之后，需要在ModelArts页面为子用户设置ModelArts访问授权，允许ModelArts访问OBS、SWR、IEF等依赖服务。

此方式只允许主用户为子用户进行配置。因此，本示例中，管理员账号需为所有用户完成访问授权的配置。

1. 使用主用户的账号登录ModelArts服务管理控制台。请注意选择左上角的区域，例如“华南-广州”。
2. 在左侧导航栏单击“权限管理”，进入“权限管理”页面。
3. 单击“添加授权”。在“授权”页面，在“授权对象类型”下面选择“所有用户”，选择“新增委托 ”，为该主用户下面的所有子用户配置委托访问授权。
   - 普通用户：普通用户的委托权限包括了用户使用ModelArts完成AI开发的所有必要功能权限，如数据的访问、训练任务的创建和管理等。一般用户选择此项即可。
   - 自定义：如果对用户有更精细化的权限管理需求，可使用自定义模式灵活按需配置ModelArts创建的委托权限。可以根据实际需在权限列表中勾选要配置的权限。

4. 勾选“我已经详细阅读并同意《ModelArts服务声明》”，单击“创建”，完成委托授权配置。

7.3.2.5 Step4 测试用户权限

由于4中的权限需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。

2. 验证ModelArts权限。

   a. 在左上角选择区域，区域需与授权配置中的区域相同。

   b. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，界面未提示权限不足，表明ModelArts的使用权限和委托授权配置成功。

      如果提示“需获取依赖服务的授权”，说明未配置ModelArts委托访问授权，请参考**Step3 为用户配置ModelArts的委托访问授权**，使用主用户为子用户配置ModelArts委托访问授权。

   c. 在ModelArts左侧菜单栏中，选择“开发环境>Notebook”，单击“创建”，如果可以正常打开创建页面，说明具备ModelArts的操作权限。

      您也可以尝试其他功能，例如“训练管理>训练作业”等，如能正常打开创建页面，即可正常使用ModelArts。

3. 验证OBS权限。

   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。

   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。

4. 验证SWR权限。

   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。

   b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。

5. 依次验证其他可选权限。

6. 验证结束，当前用户同时具备ModelArts部分功能的操作权限，可正常开始使用ModelArts服务。

7.3.3 给子用户配置开发环境基本使用权限

场景描述

本文介绍开发环境场景下子用户所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子用户使用Notebook进行调试，数据和代码存放在并行文件系统。以下内容需使用管理账号进行配置。

权限清单

- 权限

表 7-23 开发环境所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
开发环境实例生命周期管理	ModelArts	modelarts:notebook:create modelarts:notebook:list modelarts:notebook:get modelarts:notebook:update modelarts:notebook:delete modelarts:notebook:start modelarts:notebook:stop modelarts:notebook:updateStopPolicy modelarts:image:delete modelarts:image:list modelarts:image:create modelarts:image:get modelarts:pool:list modelarts:tag:list modelarts:network:get aom:metric:get aom:metric:list aom:alarm:list	实例的启动、停止、创建、删除、更新等依赖的权限。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
动态挂载存储配置	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	动态挂载存储配置。	按需配置。
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket		
镜像管理	ModelArts	modelarts:image:register modelarts:image:listGroup	在镜像管理中注册和查看镜像。	按需配置。
保存镜像	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 开发环境运行的实例，保存成镜像。 • 使用自定义镜像创建开发环境Notebook实例。	按需配置。
使用SSH功能	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	为开发环境Notebook实例配置登录密钥。	按需配置。
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list		

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
挂载SFS Turbo盘	SFS Turbo	SFS Turbo FullAccess	子用户对SFS目录的读写操作权限。专属池Notebook实例挂载SFS（公共池不支持），且挂载的SFS不是当前子用户创建的。	按需配置。
查看所有实例	ModelArts	modelarts:notebook:listAllNotebooks	ModelArts开发环境界面上，查询所有用户的实例列表，适用于给开发环境的实例管理员配置该权限。	按需配置。
	IAM	iam:users:listUsers		
VSCode插件（本地）/ PyCharm Toolkit（本地）	ModelArts	modelarts:notebook:listAllNotebooks modelarts:trainJob:create modelarts:trainJob:list modelarts:trainJob:update modelarts:trainJobVersion:delete modelarts:trainJob:get modelarts:trainJob:logExport modelarts:workspace:getQuotas（如果开通了**工作空间**功能，则需要配置此权限。）	从本地VSCode连接云上的Notebook实例、提交训练作业等。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData		
	IAM	iam:projects:listProjects	从本地PyCharm查询IAM项目列表，完成连接配置。	
VPC接入	VPC	VPC ReadOnlyAccess	实例能够挂载在用户的VPC下，实现多网络平面接入。	按需配置。

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。

- 委托

表 7-24 开发环境所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
JupyterLab	OBS	obs:object:DeleteObject obs:object:GetObject obs:object:GetObjectVersion obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets obs:object:PutObject obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:PutBucketCORS	通过ModelArts的Notebook，在JupyterLab中使用OBS上传下载数据。	建议配置。
开发环境监控功能	AOM	aom:alarm:put	调用AOM的接口，获取Notebook相关的监控数据和事件，展示在ModelArts的Notebook中。	建议配置。
VPC接入	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnet:get	实例能够挂载在用户的VPC下，实现多网络平面接入。	按需配置。

操作步骤

本案例场景为**单机单卡场景下使用Notebook进行代码调试**，数据和代码存储在OBS服务的并行文件系统下，调试完成过后可保存镜像。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加开发环境使用权限和依赖服务SWR权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

1. 添加开发环境使用权限。
   - “策略名称”：设置自定义策略名称，例如：notebook。
   - “策略配置方式”：选择JSON视图。

- “策略内容”：填入如下内容。

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "modelarts:notebook:create",
            "modelarts:notebook:list",
            "modelarts:notebook:get",
            "modelarts:notebook:update",
            "modelarts:notebook:delete",
            "modelarts:notebook:start",
            "modelarts:notebook:stop",
            "modelarts:notebook:updateStopPolicy",
            "modelarts:notebook:listMountedStorages",
            "modelarts:notebook:mountStorage",
            "modelarts:notebook:getMountedStorage",
            "modelarts:notebook:umountStorage",
            "modelarts:image:delete",
            "modelarts:image:list",
            "modelarts:image:create",
            "modelarts:image:get",
            "modelarts:pool:list",
            "modelarts:tag:list",
            "modelarts:network:get",
            "aom:metric:get",
            "aom:metric:list",
            "aom:alarm:list"
         ]
      }
   ]
}
```

步骤3 添加依赖服务OBS权限。

在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：notebook-obs。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "obs:bucket:ListAllMyBuckets",
            "obs:bucket:ListBucket"
         ]
      }
   ]
}
```

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“trainJob”为项目级云服务、“trainJobobs”为全局级云服务。**了解更多**

步骤4 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤5 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。勾选“notebook”、“notebook-obs”、“SWR Admin”策略。单击“下一步”和“确定”。

图 7-12 给用户组授权策略

步骤6 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：ma_agency_obs。
   - “策略配置方式”：JSON视图。
   - “策略内容”：填入如下内容。

   ```
   {
      "Version": "1.1",
      "Statement": [
         {
            "Effect": "Allow",
            "Action": [
               "obs:object:GetObject",
               "obs:object:DeleteObject",
               "obs:bucket:PutBucketAcl",
               "obs:object:PutObject",
               "obs:bucket:CreateBucket",
               "obs:bucket:GetBucketAcl",
               "obs:bucket:PutBucketCORS",
               "obs:bucket:ListAllMyBuckets",
               "obs:bucket:ListBucket",
               "obs:object:GetObjectVersion"
            ]
         }
      ]
   }
   ```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_notebook。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子用户配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子用户，在已有委托中选择新建的委托，然后单击“创建”。

步骤7 验证权限是否配置成功。

登录子用户账号，如果用户能在控制台上成功**创建Notebook实例**、**挂载OBS文件系统**（OBS桶需由管理员创建）、**保存镜像**，则表示权限配置成功。

----结束

7.3.4 给子用户配置训练作业基本使用权限

场景描述

本文介绍训练作业场景下子用户所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子用户使用自定义镜像训练，数据和代码存放在OBS桶中。以下内容需使用管理账号进行配置。

权限清单

- 权限

表 7-25 训练作业所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
训练管理	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	创建训练作业和查看训练日志。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
		modelarts:workspace:getQuotas	查询工作空间配额。如果开通了**工作空间**功能，则需要配置此权限。	按需配置。
		modelarts:tag:list	在训练作业中使用标签管理服务TMS。	按需配置。
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	使用配置的委托授权项。	按需配置。
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	在训练作业中使用SFS Turbo。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	SWR	swr:repository:listTags swr:repository:getRepository swr:repository:listRepositories 若为企业SWR用户，还需要增加以下权限： swr:repository:getTag swr:instance:createTempCredential swr:repository:listTags swr:repository:getRepository swr:repository:listRepositories	使用自定义镜像运行训练作业。	按需配置。
	SMN	smn:topic:publish smn:topic:list	通过SMN通知训练作业状态变化事件。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:ModifyObjectMetaData	使用OBS桶中的数据集运行训练作业。	按需配置。

- 委托

表 7-26 训练作业所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
训练作业	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	训练作业启动前下载数据、模型、代码。 训练作业运行中上传日志、模型。	建议配置。

操作步骤

本案例场景为**单机单卡场景下创建训练作业**，数据和代码存储在OBS服务的并行文件系统下，创建自定义镜像训练作业。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加训练作业使用权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：trainJob。
- “策略配置方式”：选择JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:trainJob:*",
        "modelarts:trainJobLog:*",
        "modelarts:aiAlgorithm:*",
        "modelarts:image:list",
        "modelarts:pool:list",
        "swr:repository:listTags",
        "swr:repository:getRepository",
        "swr:repository:listRepositories"
      ]
    }
  ]
}
```

步骤3 添加依赖服务OBS权限。

在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：trainJob-obs。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "obs:object:GetObject",
        "obs:object:DeleteObjectVersion",
        "obs:bucket:GetBucketLocation",
        "obs:object:AbortMultipartUpload",
        "obs:object:PutObjectAcl",
        "obs:object:DeleteObject",
        "obs:bucket:HeadBucket",
        "obs:bucket:PutBucketAcl",
        "obs:object:PutObject",
        "obs:object:GetObjectVersionAcl",
        "obs:bucket:ListAllMyBuckets",
        "obs:object:ListMultipartUploadParts",
        "obs:object:ModifyObjectMetaData",
        "obs:bucket:ListBucket",
        "obs:object:GetObjectVersion",
        "obs:object:GetObjectAcl"
      ]
    }
  ]
}
```

> **说明**
>
> 创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“trainJob”为项目级云服务、“trainJobobs”为全局级云服务。**了解更多**

步骤4 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤5 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。勾选“trainJob”和“trainJob-obs”策略。单击“下一步”和“确定”。

步骤6 为子用户添加镜像组织管理授权。

登录容器镜像服务控制台。在左侧菜单栏选择“组织管理”，单击组织名称。在“用户”页签下单击“添加授权”，在弹出的窗口中为子用户添加“编辑”权限，然后单击“确定”。

步骤7 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：ma_agency_obs。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：填入如下内容。

   ```
   {
     "Version": "1.1",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "obs:object:GetObject",
           "obs:object:PutObject",
           "obs:bucket:ListBucket"
         ]
       }
     ]
   }
   ```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_trainJob。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子用户配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子用户，在已有委托中选择新建的委托，然后单击“创建”。

步骤8 验证权限是否配置成功。

登录子用户账号，如果用户能在控制台上成功创建使用自定义镜像创建训练作业（如**单机单卡场景下创建训练作业**），则表示权限配置成功。

----结束

7.3.5 给子用户配置部署上线基本使用权限

场景描述

本文介绍部署上线场景下子用户所需的基本使用权限，您可参考**权限清单**新增对应业务场景的权限。示例场景为授权子用户权限，使其能够在开发环境Notebook中使用基础镜像构建一个新的推理镜像，并完成AI应用的创建，部署为在线服务。

权限清单

- 权限

表 7-27 管理 AI 应用所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
管理AI应用	ModelArts	modelarts:model:*	创建、删除、查看、导入AI模型。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
	SWR	SWR Admin	SWR Admin为SWR最大权限，用于： • 从自定义镜像导入。 • 从OBS导入时使用自定义引擎。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
	OBS	obs:bucket:ListAllMybuckets obs:bucket:HeadBucket obs:bucket:ListBucket obs:bucket:GetBucketLocation obs:object:GetObject obs:object:GetObjectVersion obs:object:PutObject obs:object:DeleteObject obs:object:DeleteObjectVersion obs:object:ListMultipartUploadParts obs:object:AbortMultipartUpload obs:object:GetObjectAcl obs:object:GetObjectVersionAcl obs:bucket:PutBucketAcl obs:object:PutObjectAcl	从OBS导入模型。 模型转换指定OBS路径。	按需配置。

表 7-28 部署上线所需权限

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
部署服务	ModelArts	modelarts:service:*	部署、启动、查新、更新模型服务。	建议配置。 仅在**严格授权模式**开启后，需要显式配置左侧权限。
	LTS	lts:logs:list	查询和展示LTS日志。	按需配置。

业务场景	依赖的服务	依赖策略项	支持的功能	配置建议
批量服务	OBS	obs:object:GetObject obs:object:PutObject obs:bucket:CreateBucket obs:bucket:ListBucket obs:bucket:ListAllMyBuckets	创建批量服务。	按需配置。
边缘服务	CES	ces:metricData:list	查看服务的监控指标。	按需配置。
	IEF	IEF Administrator	管理边缘服务。	按需配置。

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。

- 委托

表 7-29 部署上线所需委托

业务场景	依赖的服务	委托授权项	说明	配置建议
在线服务	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	在线服务配置LTS日志上报。	按需配置。
批量服务	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	运行批量服务。	按需配置。
边缘服务	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	通过IEF部署边缘服务。	按需配置。

操作步骤

本案例场景为**在开发环境中构建并调试推理镜像**，在Notebook中制作自定义镜像，然后将调试完成的镜像导入ModelArts的AI应用管理中，并部署上线。

步骤1 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 添加部署上线使用权限。在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

添加部署上线使用权限。

- “策略名称”：设置自定义策略名称，例如：service。
- “策略配置方式”：选择JSON视图。
- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "modelarts:service:*",
        "modelarts:model:*",
        "modelarts:notebook:create",
        "modelarts:notebook:list",
        "modelarts:notebook:get",
        "modelarts:notebook:update",
        "modelarts:notebook:delete",
        "modelarts:notebook:start",
        "modelarts:notebook:stop",
        "modelarts:notebook:updateStopPolicy",
        "modelarts:image:delete",
        "modelarts:image:list",
        "modelarts:image:create",
        "modelarts:image:get",
        "modelarts:image:register",
        "modelarts:image:listGroup",
        "modelarts:pool:list",
        "modelarts:tag:list",
        "aom:metric:get",
        "aom:metric:list",
        "aom:alarm:list"
      ]
    }
  ]
}
```

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。

在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。勾选“service”、“SWR Admin”策略。单击“下一步”和“确定”。

步骤5 添加ModelArts委托授权。

1. 新建委托授权策略。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

   - “策略名称”：设置自定义策略名称，例如：service_agency。
   - “策略配置方式”：JSON视图。

- “策略内容”：填入如下内容。

```
{
  "Version": "1.1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lts:groups:create",
        "lts:groups:list",
        "lts:topics:create",
        "lts:topics:delete",
        "lts:topics:list"
      ]
    }
  ]
}
```

2. 创建委托。

   在统一身份认证服务页面的左侧导航选择“权限管理 > 委托”，单击右上角的“创建委托”，设置策略。填写委托信息并单击“下一步”。

   - 委托名称：可自定义委托名称，例如：ma_agency_service。
   - 委托类型：选择“云服务”。
   - 云服务：选择“ModelArts”。
   - 持续时间：选择“永久”。

   勾选新建的委托策略，然后单击“下一步”。设置最小授权范围选择“所有资源”，然后单击“确定”。

3. 为子用户配置ModelArts委托权限。

   在ModelArts服务页面的左侧导航选择“权限管理 ”，单击“添加授权”。授权对象选择子用户，在已有委托中选择新建的委托，然后单击“创建”。

步骤6 验证权限是否配置成功。

登录子用户账号，如果用户能跑通**在开发环境中构建并调试推理镜像**的案例，在Notebook中制作自定义镜像，然后将调试完成的镜像导入ModelArts的AI应用管理中，并部署上线，则表示权限配置成功。

----结束

7.3.6 管理员和开发者权限分离

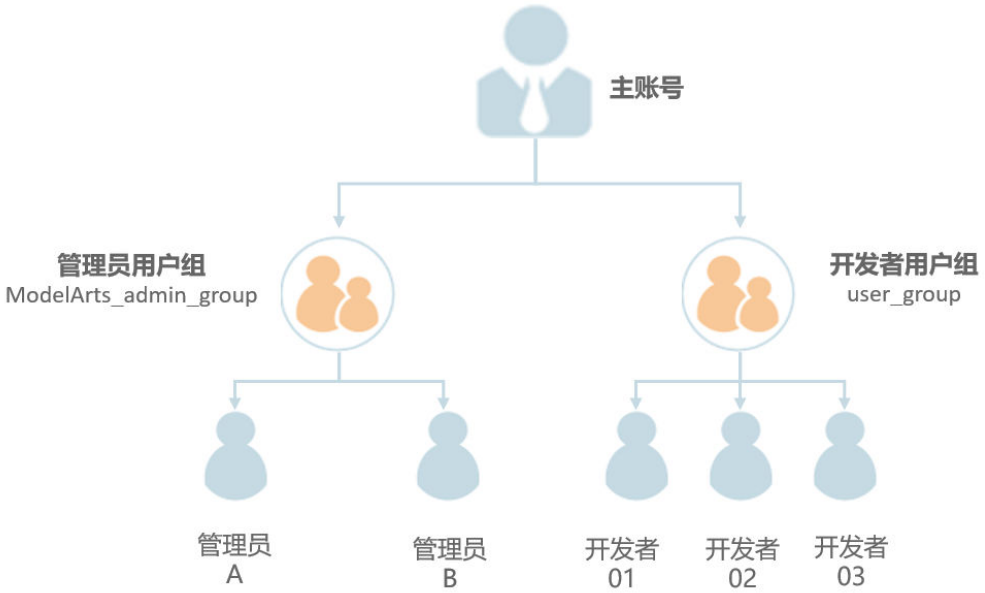
对于中小规模团队，管理员希望对ModelArts资源进行主导分配，全局控制，而对于普通开发者只需关注自己实例的生命周期控制。对于开发者账号，一般不会具有te_admin的权限，相应的权限也需要主账号进行统一配置。本章节以使用Notebook进行项目开发为例，通过自定义策略配置实现管理员和开发者分离。

场景描述

以使用Notebook进行项目开发为例，管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。

普通开发者使用开发环境，只需关注对自己Notebook实例的操作权限，包括对自己实例的创建、启动、停止、删除等权限以及周边依赖服务的权限。普通开发者不需要ModelArts专属资源池的操作权限，也不需要查看其他用户的Notebook实例。

图 7-13 账号关系示意图



配置管理员权限

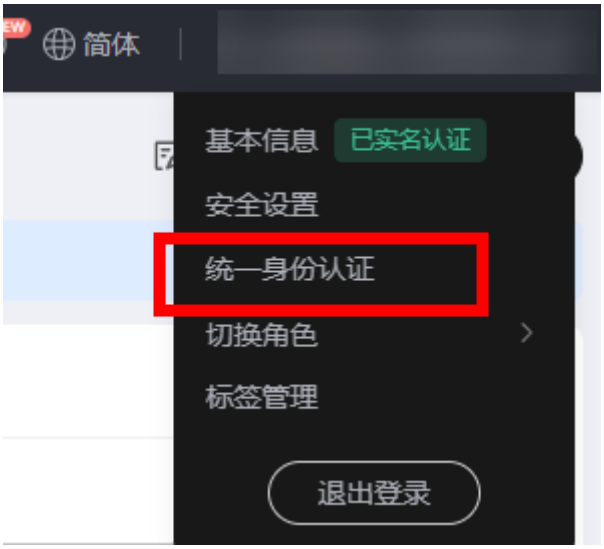
管理员账号需要拥有ModelArts专属资源池的完全控制权限，以及Notebook所有实例的访问和操作权限。可以通过以下配置流程实现管理员权限配置。

步骤1 使用主账号创建一个管理员用户组ModelArts_admin_group，将管理员账号加入用户组ModelArts_admin_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用管理员账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

   图 7-14 登录控制台

2. 创建自定义策略1，赋予用户IAM和OBS服务权限。在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，在

“策略名称”中填入“Policy1_IAM_OBS”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

自定义策略“Policy1_IAM_OBS”的具体内容如下，赋予用户IAM和OBS操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:users:listUsers",
                "iam:projects:listProjects",
                "obs:object:PutObject",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:bucket:HeadBucket",
                "obs:object:DeleteObject",
                "obs:bucket:CreateBucket",
                "obs:bucket:ListBucket"
                ]
        }
    ]
}
```

3. 重复**步骤2.2**创建自定义策略2，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。“策略名称”为“Policy2_AllowOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy2_AllowOperation”的具体内容如下，赋予用户依赖服务ECS、SWR、MRS和SMN的操作权限，ModelArts的操作权限。可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecs:serverKeypairs:list",
                "ecs:serverKeypairs:get",
                "ecs:serverKeypairs:delete",
                "ecs:serverKeypairs:create",
                "swr:repository:getNamespace",
                "swr:repository:listNamespaces",
                "swr:repository:deleteTag",
                "swr:repository:getRepository",
                "swr:repository:listTags",
                "swr:instance:createTempCredential",
                "mrs:cluster:get",
                "modelarts:*:*"
            ]
        }
    ]
}
```

步骤3 将**步骤2**创建的自定义策略授权给管理员用户组ModelArts_admin_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称ModelArts_admin_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”和“Policy2_AllowOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给管理员用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面。

2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择管理员的账号，选择“新增委托”，“权限配置”选择“普通用户”。管理员不做权限控制，此处默认使用普通用户委托即可。
3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。

步骤5 测试管理员用户权限。

1. 使用管理员用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts控制台的左侧导航栏中，选择“专属资源池”，单击创建，未提示权限不足，表明管理员用户的权限配置成功。

----结束

配置开发者权限

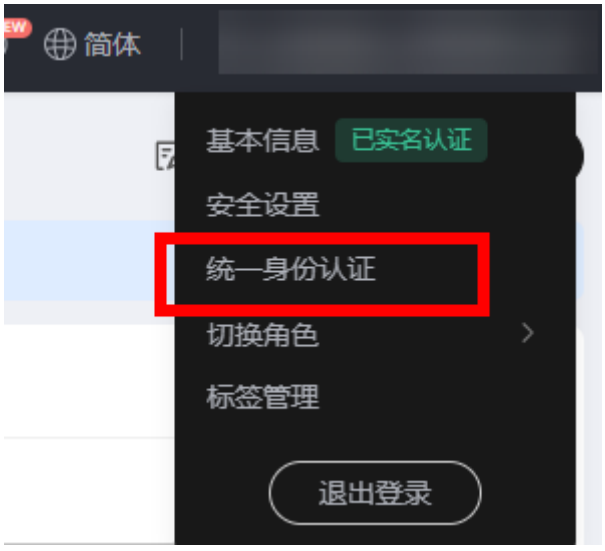
开发者权限需要通过IAM的细粒度授权控制实现，可以通过以下配置流程实现开发者权限配置。

步骤1 使用主账号创建一个开发者用户组user_group，将开发者账号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。

步骤2 创建自定义策略。

1. 使用主账号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

   图 7-15 登录控制台



2. 创建自定义策略3，拒绝用户操作ModelArts专属资源池并拒用户查看其他用户的Notebook。

   在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy3_DenyOperation”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

   自定义策略“Policy3_DenyOperation”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
```

```
  "Statement": [
        {
         "Effect": "deny",
         "Action": [
             "modelarts:pool:create",
             "modelarts:pool:update",
             "modelarts:pool:delete",
      "modelarts:notebook:listAllNotebooks"
          ]

      }
   ]
}
```

步骤3 将自定义策略授权给开发者用户组user_group。

1. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1_IAM_OBS”、“Policy2_AllowOperation”和“Policy3_DenyOperation”。单击“下一步”。
2. 选择授权范围方案为所有资源，单击“确定”。

步骤4 给开发者用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用主账号登录ModelArts的管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面。
2. 单击“添加授权”。在“访问授权”页面，在“授权对象类型”下面选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“新增委托”，“委托名称”设置为“ma_agency_develop_user”，“权限配置”选择“自定义”，“权限名称”勾选“OBS Administrator”。开发者用户只需要配置OBS的委托授权即可，允许开发者用户在使用Notebook时，与OBS服务交互。
3. 勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，单击“创建”。
4. 在“权限管理”页面，再次单击“添加授权”，进入“访问授权”页面，为其他开发者用户配置委托。

   “授权对象类型”选择“IAM子用户”，“授权对象”选择开发者的账号，“委托选择”选择“已有委托”，“委托名称”勾选上一步创建的“ma_agency_develop_user”，

步骤5 测试开发者用户权限。

1. 使用user_group用户组中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 在ModelArts左侧菜单栏中，选择“专属资源池”，单击创建，界面未提示权限不足，表明开发者用户的权限配置成功。

----结束

7.3.7 查找 Notebook 实例

查找实例

Notebook页面展示了所有创建的实例。如果需要查找特定的实例，可根据筛选条件快速查找。

- 参考**给子账号配置查看所有Notebook实例的权限**后，进入“开发空间>Notebook”页面，打开“查看所有”开关，可以看到IAM项目下所有子用户创建的Notebook实例。
- 按实例名称、实例ID、实例状态、使用的镜像、实例规格、实例描述、创建时间等单个筛选或组合筛选。

给子账号配置查看所有 Notebook 实例的权限

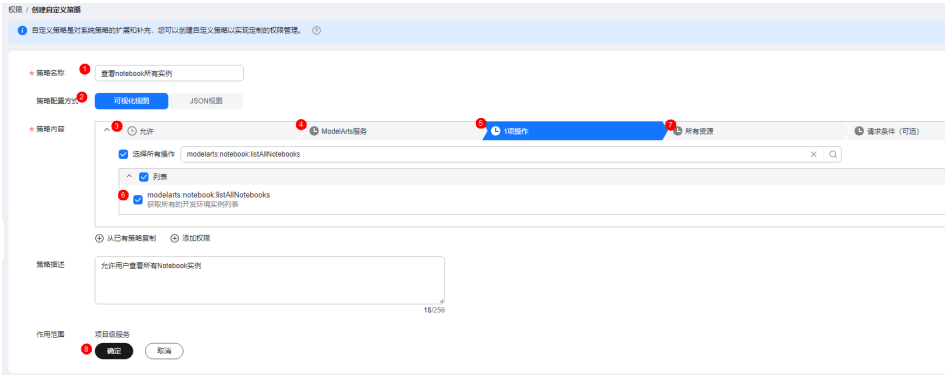
当子用户被授予“listAllNotebooks”和“listUsers”权限时，在Notebook页面上，单击“查看所有”，可以看到IAM项目下所有子用户创建的Notebook实例 。配置该权限后，也可以在Notebook中访问子用户的OBS、SWR等。

1. 使用主用户账号登录ModelArts管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，需要设置两条策略。

   策略1：设置查看Notebook所有实例，如**图7-16**所示，单击“确定”。

   - “策略名称”：设置自定义策略名称，例如：查看Notebook所有实例。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:notebook:listAllNotebooks并选中，所有资源选择默认值。

   图 7-16 创建自定义策略

   策略2：设置查看Notebook实例创建者信息的策略。

   - “策略名称”：设置自定义策略名称，例如：查看所有子用户信息。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索IAM服务并选中，操作列中搜索关键词iam:users:listUsers并选中，所有资源选择默认值。
3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

子用户启动其他用户的 SSH 实例

子用户可以看到所有用户的Notebook实例后，如果要通过SSH方式远程连接其他用户的Notebook实例，需要将SSH密钥对更新成自己的，否则会报错ModelArts.6786。更新密钥对具体操作请参见**修改Notebook SSH远程连接配置**。具体的错误信息提示：ModelArts.6789: 在ECS密钥对管理中找不到指定的ssh密钥对xxx，请更新密钥对并重试。

7.3.8 使用 Cloud Shell 登录训练容器

使用场景

允许用户使用ModelArts控制台提供的Cloud Shell登录运行中的训练容器。

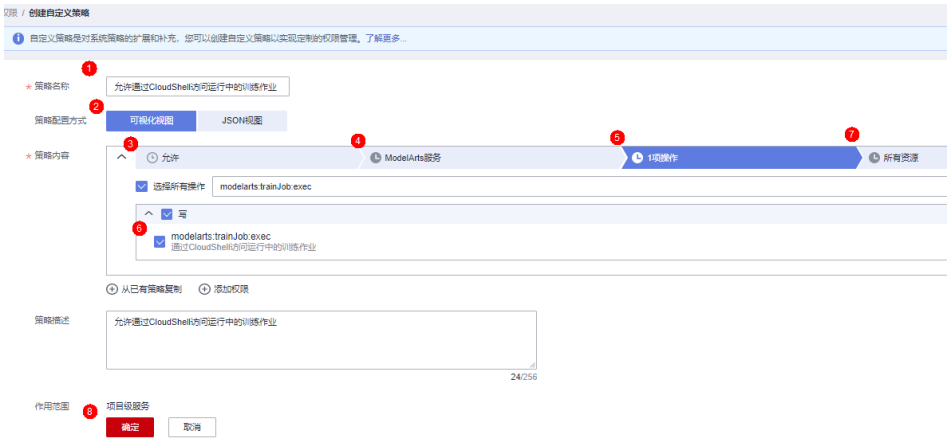
约束限制

仅专属资源池支持使用Cloud Shell，且训练作业必须处于“运行中”状态。

前提条件：给子账号配置允许使用 Cloud Shell 的权限

1. 使用主用户账号登录华为云的管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”按如下要求设置完成后单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：允许通过Cloud Shell访问运行中的训练作业。
   - “策略配置方式”：选择可视化视图。
   - “策略内容”：允许，云服务中搜索ModelArts服务并选中，操作列中搜索关键词modelarts:trainJob:exec并选中，所有资源选择默认值。

   图 7-17 创建自定义策略

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤2创建的自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限通过Cloud Shell登录运行中的训练作业容器。

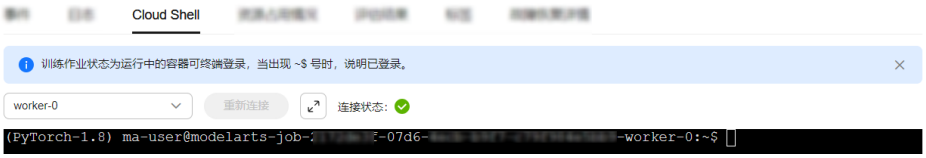
如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

如何使用 Cloud Shell

1. 参考**前提条件：给子账号配置允许使用Cloud Shell的权限**，完成配置。
2. 在ModelArts管理控制台的左侧导航栏中选择“模型训练 > 训练作业”。
3. 在训练作业列表中，单击作业名称进入训练作业详情页面。
4. 在训练作业详情页面，单击“Cloud Shell”页签，登录训练容器。

   连接成功后，Cloud Shell界面提示如下。

图 7-18 Cloud Shell 界面



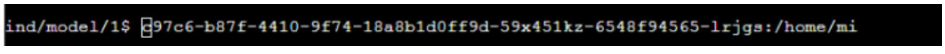
当作业处于非运行状态或权限不足时会导致无法使用Cloud Shell，请根据提示定位原因即可。

图 7-19 报错提示

说明

部分用户登录Cloud Shell界面时，可能会出现路径显示异常情况，此时在Cloud Shell中单击回车键即可恢复正常。

图 7-20 路径异常



7.3.9 限制用户使用公共资源池

本章节介绍如何控制ModelArts用户权限，限制用户使用ModelArts公共资源池的资源创建训练作业、创建开发环境实例，部署推理服务等。

场景介绍

对于ModelArts专属资源池的用户，不允许使用公共资源池创建训练作业、创建Notebook实例或者部署推理服务时，可以通过权限控制限制用户使用公共资源池。

涉及配置的自定义权限策略项如下;

- modelarts:notebook:create：此策略项表示创建Notebook实例。
- modelarts:trainJob:create：此策略项表示创建训练作业。
- modelarts:service:create：此策略项表示创建推理服务。

给子账号配置权限：限制使用公共资源池

1. 使用主用户账号登录管理控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。
2. 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略，单击“确定”。
   - “策略名称”：设置自定义策略名称，例如：不允许用户使用公共资源池创建。
   - “策略配置方式”：选择可视化视图或者JSON视图均可。
   - “策略内容”：拒绝，云服务中搜索“ModelArts”服务并选中，“操作”中查找**写**操作“modelarts:trainJob:create”、“modelarts:notebook:create”和“modelarts:service:create”并选中。“所有资源”选择“默认值”。“请求条件”中单击“添加条件”，设置“条件键”为“modelarts:poolType”，“运算符”为“StringEquals”，“值”为“public”。

     JSON视图的策略内容如下：

     ```
     {
       "Version": "1.1",
       "Statement": [
         {
           "Effect": "Deny",
           "Action": [
             "modelarts:trainJob:create",
             "modelarts:notebook:create",
             "modelarts:service:create"
           ],
           "Condition": {
             "StringEquals": {
               "modelarts:poolType": [
                 "public"
               ]
             }
           }
         }
       ]
     }
     ```

3. 在统一身份认证服务页面的左侧导航选择“用户组”，在用户组页面查找待授权的用户组名称，在右侧的操作列单击“授权”，勾选步骤**2**创建的两条自定义策略，单击“下一步”，选择授权范围方案，单击“确定”。

   此时，该用户组下的所有用户均有权限查看该用户组内成员创建的所有Notebook实例。

   如果没有用户组，也可以创建一个新的用户组，并通过“用户组管理”功能添加用户，并配置授权。如果指定的子用户没有在用户组中，也可以通过“用户组管理”功能增加用户。

4. 在用户的委托授权中同步增加此策略，避免在租户面通过委托token突破限制。

   在统一身份认证服务页面的左侧导航中选择委托，找到该用户组在ModelArts上使用的委托名称，单击右侧的“修改”操作，选择“授权记录”页签，单击“授权”，选中上一步创建的自定义策略“不允许用户使用公共资源池”，单击“下一步”，选择允许使用的资源区域，单击“确定”。

验证

使用子账号用户登录ModelArts控制台，选择“训练管理 > 训练作业”，单击“创建训练作业”，在创建训练页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“开发环境 > Notebook”，单击“创建”，在创建Notebook页面，资源池规格只能选择专属资源池。

使用子账号用户登录ModelArts控制台，选择“部署上线 > 在线服务”，单击“部署”，在部署服务页面，资源池规格只能选择专属资源池。

7.3.10 给子用户配置文件夹级的 SFS Turbo 访问权限

场景描述

本文介绍如何配置文件夹级的SFS Turbo访问权限，实现在ModelArts中访问挂载的SFS Turbo时，只允许子用户访问特定的SFS Turbo文件夹内容。

说明

给子用户配置文件夹级的SFS Turbo访问权限为白名单功能，如果有试用需求，请提工单申请权限。

前提条件

- 需要在ModelArts控制台打开严格授权模式，单击“权限管理 > 启用严格模式”。
- 如果打开严格模式前没有为子用户配置过ModelArts权限，开启严格授权模式后可能会导致子用户无法使用ModelArts功能，请根据您的业务需求配置需要的ModelArts服务的权限（参见**依赖和委托**中ModelArts服务对应的依赖策略项）。

操作步骤

步骤1 使用主用户账号登录管理控制台，鼠标放在右上角用户名，在下拉框中选择“统一身份认证”，进入统一身份认证（IAM）服务。

步骤2 在统一身份认证服务页面的左侧导航选择“权限管理 > 权限”，单击右上角的“创建自定义策略”，设置策略。

- “策略名称”：设置自定义策略名称，例如：ma_sfs_turbo。
- “策略配置方式”：JSON视图。
- “策略内容”：填入如下内容。

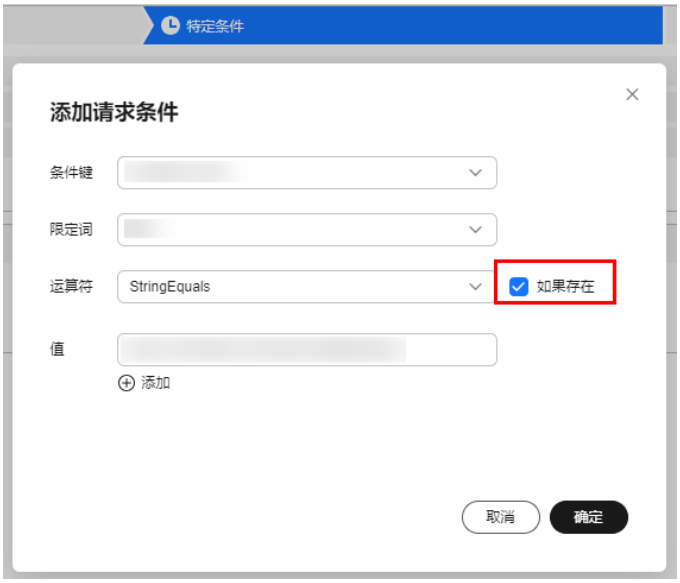
```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "<modelarts_action>"
            ],
```

```
            "Condition": {
                "StringEqualsIfExists": {
                    "modelarts:sfsId": [
                        "<your_ssf_id>"
                    ],
                    "modelarts:sfsPath": [
                        "<sfs_path>"
                    ],
                    "modelarts:sfsOption": [
                        "<sfs_option>"
                    ]
                }
            }
        }
    ]
}
```

说明

- 未创建以上权限策略前，所有子用户默认可以挂载SFS Turbo。当您创建了以上SFS权限管控策略后，没有被授予以上权限的子用户，默认在ModelArts Console上创建训练作业时无法挂载SFS Turbo（具有Tenant Administrator权限的子用户除外）。
- 当前仅支持配置允许策略的权限（即以上“策略内容”中的“Effect”只能配置为“Allow”），请勿配置拒绝策略的权限。
- Condition参数必须使用“StringEqualsIfExists”字段，对应可视化视图为勾选“如果存在”的开关。

图 7-21 “如果存在”的开关



以上代码中的"*<modelarts_action>*"、"*<your_ssf_id>*"、"*<sfs_path>*"、"*<sfs_option>*"，需要根据您的业务需求替换为实际的参数，各参数含义如下。

表 7-30 参数解释

参数	参数解释
Action	表示在何种场景下授予SFS Turbo文件夹访问权限。 ● 创建开发环境实例：modelarts:notebook:create ● 创建训练作业：modelarts:trainJob:create 支持填写多种Action，例如： "Action": ["modelarts:trainJob:create", "modelarts:notebook:create"],
modelarts:sfsId	SFS Turbo的ID，在SFS Turbo详情页查看。支持填写多个ID，例如： "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"],
modelarts:sfsPath	需要进行权限配置的SFS Turbo文件夹路径。支持填写多个路径，例如： "modelarts:sfsPath": ["/path1", "/path2/path2-1"], 如果sfsId中填写了多个ID，则sfsPath会应用于所有sfsId。例如以下代码含义为：为"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"的"/path1"和"/path2/path2-1"配置访问权限，同时也为"2a70da1e-ea87-4ee4-ae1e-55df846e7f41"的"/path1"和"/path2/path2-1"配置访问权限。 "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"], "modelarts:sfsPath": ["/path1", "/path2/path2-1"],

参数	参数解释
modelarts:sfsOption	设置用户对于SFS Turbo文件夹的权限类型，支持填写以下参数： • 仅读权限：readonly • 读写权限：readwrite（创建开发环境实例modelarts:notebook:create仅支持配置readwrite）

如果需要在一个自定义策略中添加多个不同的sfsOption，需要“Statement”中新增JSON结构体，例如：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:trainJob:create"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "modelarts:sfsId": [
                        "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
                    ],
                    "modelarts:sfsPath": [
                        "/path1"
                    ],
                    "modelarts:sfsOption": [
                        "readonly"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:trainJob:create"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "modelarts:sfsId": [
                        "0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
                    ],
                    "modelarts:sfsPath": [
                        "/path2"
                    ],
                    "modelarts:sfsOption": [
                        "readwrite"
                    ]
                }
            }
        }
    ]
}
```

步骤3 创建用户组并加入用户，步骤请参考**Step1 创建用户组并加入用户**。

步骤4 给用户组授权策略。在IAM服务的用户组列表页面，单击“授权”，进入到授权页面，为子用户配置权限。勾选**步骤2**中创建的“ma_sfs_turbo”策略。单击“下一步”和“确定”。

步骤5 在已有的ModelArts委托权限中，追加IAM ReadOnlyAccess权限。

1. 在ModelArts管理控制台，单击“权限管理”，在对应委托的操作列，单击“查看权限 > 去IAM修改委托权限”。

2. 在新页面中，单击“授权记录 > 授权”，搜索“IAM ReadOnlyAccess”，勾选后单击“下一步”并单击“确认”。

步骤6 验证权限是否配置成功。

登录子用户账号，在创建训练作业/创建Notebook时，仅能看到配置的SFS Turbo文件夹，则表示权限配置成功。

----结束

7.4 FAQ

7.4.1 使用 ModelArts 时提示“权限不足”，如何解决?

当您使用ModelArts时如果提示权限不足，请您按照如下指导对相关服务和用户进行授权，并对用户权限进行检查操作。

由于ModelArts的使用权限依赖OBS服务的授权，您需要为用户授予OBS的系统权限。

- 如果您需要授予用户关于OBS的所有权限和ModelArts的基础操作权限，请参见**配置基础操作权限**。
- 如果您需要对用户使用OBS和ModelArts的权限进行精细化管理，进行自定义策略配置，请参见**创建ModelArts自定义策略**。

配置基础操作权限

使用ModelArts的基本功能，您需要为用户配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，由于ModelArts依赖OBS权限，您还需要**登录IAM管理控制台**为用户授予“作用范围”为“全局级服务”的“OBS Administrator”策略。

具体操作步骤如下：

步骤1 创建用户组。

登录IAM管理控制台，单击“用户组>创建用户组”。在“创建用户组”界面，输入“用户组名称”单击“确定”。

步骤2 配置用户组权限。

在用户组列表中，单击步骤1新建的用户组右侧的“授权”，在用户组“授权”页面，您需要配置的权限如下：

1. 配置“作用范围”为“项目级服务”的“ModelArts CommonOperations”权限，如下图所示，然后单击“确定”完成授权。

   说明

   区域级项目授权后只在授权区域生效，如果需要所有区域都生效，则所有区域都需要进行授权操作。

2. 配置“作用范围”为“全局级服务”的“OBS Administrator”权限，然后单击“确定”完成授权。

步骤3 **创建用户并加入用户组**。

在IAM控制台创建用户，并将其加入步骤1中创建的用户组。

步骤4 用户登录并验证权限。

新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择ModelArts，进入ModelArts主界面，选择不同类型的专属资源池，在页面单击“创建”，如果无法进行创建（当前权限仅包含ModelArts CommonOperations），表“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择除ModelArts外（假设当前策略仅包含ModelArts CommonOperations）的任一服务，如果提示权限不足，表示“ModelArts CommonOperations”已生效。
- 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理>数据集>创建数据 > 集”，如果可以成功访问对应的OBS路径，表示全局级服务的“OBS Administrator”已生效。

----结束

创建 ModelArts 自定义策略

如果系统预置的ModelArts权限不满足您的授权要求，或者您需要管理用户操作OBS的操作权限，可以创建自定义策略。更多关于创建自定义策略操作和参数说明请参见**创建自定义策略**。

目前华为云支持可视化视图创建自定义策略和JSON视图创建自定义策略，本章节将使用JSON视图方式的策略，以为ModelArts用户授予开发环境的使用权限并且配置ModelArts用户OBS相关的最小化权限项为例，指导您进行自定义策略配置。

说明

如果一个自定义策略中包含多个服务的授权语句，这些服务必须是同一属性，即都是全局级服务或者项目级服务。

由于OBS为全局服务，ModelArts为项目级服务，所以需要创建两条“作用范围”别为“全局级服务”以及“项目级服务”的自定义策略，然后将两条策略同时授予用户。

1. 创建ModelArts相关OBS的最小化权限的自定义策略。

   登录IAM控制台，在“权限管理>权限”页面，单击“创建自定义策略”。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts依赖的OBS权限自定义策略样例**，如果您需要了解更多关于OBS的系统权限，请参见**OBS权限管理**。

2. 创建ModelArts开发环境的使用权限的自定义策略。参数配置说明如下：

   - “策略名称”支持自定义。
   - “策略配置方式”为“JSON视图”。
   - “策略内容”请参见**ModelArts开发环境使用权限的自定义策略样例**，ModelArts自定义策略中可以添加的授权项（Action）请参见**《ModelArts API参考》>权限策略和授权项**。
   - 如果您需要对除ModelArts和OBS之外的其它服务授权，IAM支持服务的所有策略请参见**权限策略**。

3. 在IAM控制台**创建用户组并授权**。

   在IAM控制台创建用户组之后，将步骤**1**中创建的自定义策略授权给该用户组。

4. **创建用户并加入用户组**。

在IAM控制台创建用户，并将其加入3中创建的用户组。

5. 用户登录并验证权限。

   新创建的用户登录控制台，切换至授权区域，验证权限：

   - 在“服务列表”中选择ModelArts，进入ModelArts主界面，单击“数据管理 > 数据集”，如果无法进行创建（当前仅包含开发环境的使用权限），表示仅为ModelArts用户授予开发环境的使用权限已生效。
   - 在“服务列表”中选择除ModelArts，进入ModelArts主界面，单击“开发环境>Notebook>创建”，如果可以成功访问“存储配置”项对应的OBS路径，表示为用户配置的OBS相关权限已生效。

ModelArts 依赖的 OBS 权限自定义策略样例

如下示例为ModelArts依赖OBS服务的最小化权限项，包含OBS桶和OBS对象的权限。授予示例中的权限您可以通过ModelArts正常访问OBS不受限制。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
                "obs:object:ListMultipartUploadParts",
                "obs:object:AbortMultipartUpload",
                "obs:object:GetObjectAcl",
                "obs:object:GetObjectVersionAcl",
                "obs:bucket:PutBucketAcl",
                "obs:object:PutObjectAcl"
            ],
            "Effect": "Allow"
        }
    ]
}
```

ModelArts 开发环境使用权限的自定义策略样例

```
{
    "Version": "1.1",
    "Statement": [

        {
            "Effect": "Allow",
            "Action": [
                "modelarts:notebook:list",
                "modelarts:notebook:create" ,
                "modelarts:notebook:get" ,
                "modelarts:notebook:update" ,
                "modelarts:notebook:delete" ,
                "modelarts:notebook:action" ,
                "modelarts:notebook:access"
            ]
        }
    ]
}
```

8 Standard 自动学习

8.1 使用 ModelArts Standard 自动学习实现口罩检测

该案例是使用华为云一站式AI开发平台ModelArts的新版“自动学习”功能，基于华为云AI开发者社区AI Gallery中的数据集资产，让零AI基础的开发者完成“物体检测”的AI模型的训练和部署。依据开发者提供的标注数据及选择的场景，无需任何代码开发，自动生成满足用户精度要求的模型。可支持图片分类、物体检测、预测分析、声音分类等场景。可根据最终部署环境和开发者需求的推理速度，自动调优并生成满足要求的模型。

说明

费用说明：本案例使用过程中，从AI Gallery下载数据集免费，但是数据集存储在OBS桶中会收取少量费用，具体计费请参见**OBS价格详情页**。

在ModelArts上运行训练作业、将模型部署为在线服务会收取计算资源费用。案例使用完成后请参考**步骤6：清除相应资源**及时清除资源和数据。

步骤 1：准备工作

- 注册华为账号并开通华为云、实名认证
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录ModelArts管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在弹出的“访问授权”窗口中，授权对象类型选“所有用户”，委托选择选“新增委托”，权限配置选择“普通用户”，并勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

  c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 2：创建训练数据集

1. 单击**口罩检测小数据集**进入数据集详情页，单击右侧“下载”。

2. 在弹出的窗口中选择云服务区域，例如该案例选择云服务区域为“华北-北京四”，单击“确定”进入下载详情页。
3. 在“下载详情”页面，填写参数。
   - 下载方式：ModelArts数据集。
   - 目标区域：华北-北京四，目标区域须与上一步中选择的云服务区域保持一致。
   - 数据类型：图片。
   - 数据集输入位置：用来存放源数据集信息，例如本案例中从Gallery下载的数据集。单击图标选择您的OBS桶下的任意一处目录，但不能与输出位置为同一目录。
   - 数据集输出位置：用来存放输出的数据标注的相关信息，或版本发布生成的Manifest文件等。单击图标选择OBS桶下的空目录，且此目录不能与输入位置一致，也不能为输入位置的子目录。
   - 名称：创建数据集名称，为方便后续创建物体检测项目选择对应的数据集，建议您的数据集名称具有可识别性。
   - 描述：描述数据集详细信息。

   图 8-1 下载详情

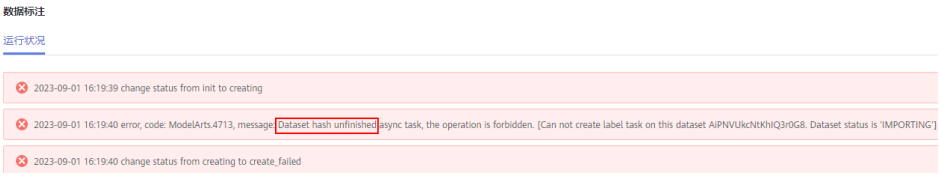
4. 确认无误后单击右下角“确定”。
5. 系统会跳转到我的下载页面，单击 按钮，查看下载进度，等待数据集下载完成（下载完成大约需要5分钟，请耐心等待）。单击 展开下载详情，可以查看该数据集的“目标位置”。
6. 查看数据集是否已导入ModelArts。

   返回ModelArts管理控制台，在左侧导航栏选择“数据管理 > 数据集”单击“前往新版”。在新版数据集列表页，单击数据集名称左侧的 ，展开数据集，查看“导入状态”，导入状态为“导入完成”代表示数据集导入成功，且数据集正常。

说明

数据集下载完成后，请务必先检查数据集是否已经导入成功，如果数据集还未成功导入，创建自动学习物体检测项目后数据标注节点会报错。

图 8-2 数据标注节点报错



步骤 3：创建自动学习物体检测项目

1. 确保数据集创建完成且可正常使用后，在ModelArts控制台，左侧导航栏选择“自动学习”默认进入新版自动学习页面，选择物体检测项目，单击“创建项目”。
2. 进入“创建物体检测”页面后，填写相关参数。
   - 计费模式：默认按需计费。
   - 名称：自行创建项目名称。
   - 描述：自行描述项目详情，例如口罩检测。
   - 数据集：下拉选择已下载的数据集（**步骤2**中已成功导入的数据集，默认为下拉数据集列表中的第一个数据集）。
   - 输出路径：选择**步骤2的3**中的数据集输出位置。
   - 训练规格：根据您的实际需要选择对应的训练规格。
3. 确认无误后单击右下角“创建项目”可自动跳转至自动学习的运行总览页面。

步骤 4：运行工作流

在自动学习的运行总览页面，会产生一条工作流。工作流会自动从数据标注节点开始，依次运行数据集版本发布、数据校验、物体检测、模型注册、服务部署等节点，直至工作流全部运行完成。您需要做的是：

1. 在数据标注节点，待数据标注节点变为橘色即为“等待操作”状态，双击数据标注节点，打开数据标注节点的运行详情页面。前往实例详情页确认所有图片是否都标注完成，确认无误后，回到工作流页面单击“继续运行”。
2. 在“确认是否继续允许”的弹窗中，单击“确定”，工作流会继续从数据标注节点依次运行到服务部署节点。该段时间不需要用户做任何操作。
3. 当工作流运行到“服务部署”节点，“服务部署”节点会变成橙色，双击“服务部署”节点。在服务部署页签中，可以看到状态变为了“等待输入”。
4. 需要选择填写以下两个参数，其他参数均为默认值，保持不变。
   - 计算节点规格：根据您的实际需求选择相应的规格。
   - 是否自动停止：为避免资源浪费，建议打开自动停止开关，根据您的实际需要，选择自动停止时间，也可以自定义自动停止的时间。

图 8-3 选择计算节点规格

图 8-4 设置自动停止

5. 参数填写完毕之后，单击运行状况右边的“继续运行”，单击确认弹窗中的“确定”即可继续完成工作流的运行。

步骤 5：预测分析

运行完成的工作流会自动部署为相应的在线服务，您只需要在相应的服务详情页面进行预测即可。

1. 在服务部署节点单击“实例详情”直接跳转进入在线服务详情页，或者在ModelArts管理控制台，选择“部署上线>在线服务”，单击生成的在线服务名称，即可进入在线服务详情页。
2. 在服务详情页，选择“预测”页签。

图 8-5 上传预测图片

3. 单击“上传”选择上传一张需要预测的图片，单击“预测”，即可在右边的预测结果显示区查看您的预测结果。

图 8-6 查看预测结果（1）--没戴口罩

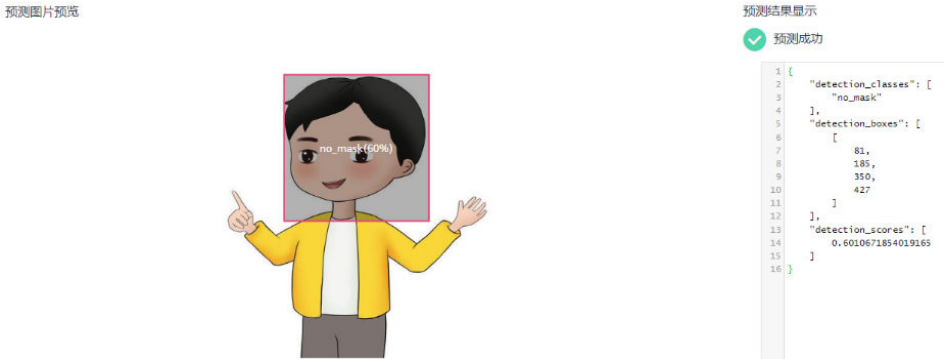
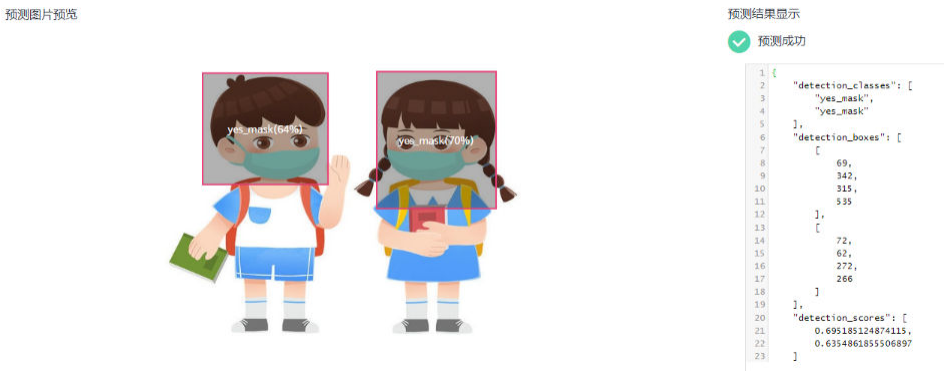


图 8-7 查看预测结果（2）--戴口罩



步骤 6：清除相应资源

在完成预测之后，建议关闭服务，以免产生不必要的计费。

1. 停止运行服务
   - 预测完成后，单击页面右上角的“停止”，即可停止该服务。
   - 单击左上角 返回在线服务，在对应的服务名称所在行，单击选择操作列的“更多>停止”，停止该服务。

     图 8-8 停止服务

2. 清除OBS中的数据。
   a. 在控制台左侧导航栏的服务列表 ，选择“对象存储服务OBS”，进入OBS服务详情页面。
   b. 在左侧导航栏选择“桶列表”，在列表详情，找到自己创建的OBS桶，单击桶名称，进入OBS桶详情。
   c. 在桶的详情页，左侧导航栏选择“对象”，在右侧“名称”列选中不需要的存储对象，单击操作列的“更多>删除”，即可删除相应的存储对象。

常见问题

- 创建数据集时找不到创建的OBS桶，请**查看OBS桶与ModelArts是否在同一个区域**。
- 数据校验节点失败。
  请查看您的数据集是否符合规范，数据集规范请参考**数据集要求与上传规范**。

8.2 使用 ModelArts Standard 自动学习实现垃圾分类

随着科技发展与人们生活质量的快速提升，生活垃圾分类成为当下越来越热门的话题，常见的生活垃圾分为厨余垃圾蛋壳、厨余垃圾水果果皮、可回收物塑料玩具、可回收物纸板箱、其他垃圾烟蒂、其他垃圾一次性餐盒、有害垃圾干电池、有害垃圾过期药物等。人工识别效率低下、费时费力，AI技术显然可以为此贡献一份力量。

该案例介绍了华为云一站式开发平台ModelArts的自动学习功能实现的常见生活垃圾分类，让您不用编写代码也可以实现生活垃圾分类。

说明

本案例只适用于新版自动学习功能。

步骤 1：准备工作

1. 注册华为账号并开通华为云、实名认证
   - **注册华为账号并开通华为云**
   - **进行实名认证**
2. 配置委托访问授权

   ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

   a. 使用华为云账号登录ModelArts管理控制台，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

   b. 在弹出的“访问授权”窗口中，授权对象类型选“所有用户”，委托选择选“新增委托”，权限配置选择“普通用户”，并勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

   c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 2：创建 OBS 桶

1. 登录**OBS管理控制台**，在桶列表页面右上角单击“创建桶”，创建OBS桶。例如，创建名称为“dataset-exeml”的OBS桶。

图 8-9 创建桶

说明

- 创建桶的区域需要与ModelArts所在的区域一致。例如：当前ModelArts在华北-北京四区域，在对象存储服务创建桶时，请选择华北-北京四。请参考**查看OBS桶与ModelArts是否在同一区域**检查您的OBS桶区域与ModelArts区域是否一致。
- 请勿开启桶加密，ModelArts不支持加密的OBS桶，会导致ModelArts读取OBS中的数据失败。

2. 在桶列表页面，单击桶名称，进入该桶的概览页面。

图 8-10 桶列表

3. 单击左侧导航的“对象”，在对象页面单击“新建文件夹”，创建OBS文件夹。具体请参见**新建文件夹**章节。

图 8-11 新建文件夹

步骤 3：准备训练数据集

1. 单击**8类常见生活垃圾图片数据集**，进入AI Gallery数据集详情页，单击右侧“下载”。
2. 选择对应的云服务区域例如：华北-北京四，需要确保您选择的区域与您的管理控制台所在的区域一致。
3. 进入“下载详情”页面，填写以下参数。
   - 下载方式：ModelArts数据集。
   - 目标区域：华北-北京四。
   - 数据类型：系统会根据您的数据集，匹配到相应的数据类型。例如本案例使用的数据集，系统匹配为“图片”类型。
   - 数据集输入位置：用来存放源数据集信息，例如本案例中从Gallery下载的数据集。单击图标选择您的OBS桶下的任意一处目录，但不能与输出位置为同一目录。
   - 数据集输出位置：用来存放输出的数据标注的相关信息，或版本发布生成的Manifest文件等。单击图标选择OBS桶下的空目录，且此目录不能与输入位置一致，也不能为输入位置的子目录。

图 8-12 下载详情

4. 完成参数填写，单击“确定”，自动跳转至AI Gallery个人中心“我的下载”页签，单击 按钮，查看下载进度，等待5分钟左右下载完成，单击 展开下载详情，可以查看该数据集的“目标位置”。

步骤 5：创建新版自动学习图像分类项目

1. 确保数据集创建完成且可正常使用后，在ModelArts控制台，左侧导航栏选择“自动学习”，进入自动学习总览页面。
2. 单击选择“图像分类”创建项目。完成参数填写。
   - 计费模式：按需计费。
   - 名称：自定义您的项目名称。
   - 描述：自定义描述您的项目详情，例如垃圾分类。
   - 数据集：下拉选择已下载的数据集（**步骤2**中已成功导入的数据集，默认为下拉数据集列表中的第一个数据集）。
   - 输出路径：选择您**步骤1**创建好的OBS文件夹下的路径，用来存储训练模型等相关文件。
   - 训练规格：根据您的实际需要选择对应的训练规格。
3. 参数填写完成，单击“创建项目”。

步骤 6：运行工作流

项目完成创建之后，会自动跳转到新版自动学习的运行总览页面。同时您的工作流会自动从数据标注节点开始运行。您需要做的是：

1. 观察数据标注节点，待数据标注节点变为橙色即为“等待操作”状态。双击数据标注节点，打开数据标注节点的运行详情页面，单击“继续运行”。
2. 在弹出的窗口中，单击“确定”，工作流会开始继续运行。当工作流运行到“服务部署”节点，状态会变为“等待输入”，您需要填写以下两个输入参数，其他参数保持默认。
   - 计算节点规格：根据您的实际需求选择相应的规格，不同规格的配置费用不同，选择好规格后，配置费用处会显示相应的费用。

- 是否自动停止：为了避免资源浪费，建议您打开该开关，根据您的需求，选择自动停止时间，也可以自定义自动停止的时间。

图 8-13 选择计算节点规格

图 8-14 设置自动停止

3. 参数填写完毕之后，单击运行状况右边的“继续运行”，单击确认弹窗中的“确定”即可继续完成工作流的运行。

步骤 7：预测分析

运行完成的工作流会自动部署相应的在线服务，您只需要在相应的服务详情页面进行预测即可。

1. 在服务部署节点单击“实例详情”或者在ModelArts管理控制台，选择“部署上线>在线服务”，单击生成的在线服务名称，即可进入在线服务详情页。
2. 在服务详情页，单击选择“预测”页签。

图 8-15 上传预测图片

3. 单击“上传”选择一张需要预测的图片，单击“预测”，即可在右边的预测结果显示区查看您的预测结果。

图 8-16 预测样例图

图 8-17 查看预测结果

说明

本案例中数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练数据集中的图片相似才可能预测准确。

ModelArts的AI Gallery中提供了常见的精度较高的算法和相应的训练数据集，用户可以在**AI Gallery的资产集市**中获取。

步骤 8：清除相应资源

在完成预测之后，建议关闭服务，以免产生不必要的计费。

1. 停止运行服务
   - 预测完成后，单击页面右上角的“停止”，即可停止该服务。
   - 单击左上角 < 返回在线服务，在对应的服务名称所在行，单击选择操作列的“更多>停止”，停止该服务。

图 8-18 停止服务

2. 清除OBS中的数据。

   a. 在控制台左侧导航栏的服务列表 ☰，选择“对象存储服务OBS”，进入OBS服务详情页面。

   b. 在左侧导航栏选择“桶列表”，在列表详情，找到自己创建的OBS桶，单击桶名称，进入OBS桶详情。

   c. 在桶的详情页，左侧导航栏选择“对象”，在右侧“名称”列选中不需要的存储对象，单击“操作”列的“更多>删除”，即可删除相应的存储对象。

常见问题

- 创建数据集时找不到创建的OBS桶，请**查看OBS桶与ModelArts是否在同一个区域**。
- 数据校验节点失败。

  请查看您的数据集是否符合规范，数据集规范请参考**数据集要求与上传规范**。

9 Standard 开发环境

9.1 将 Notebook 的 Conda 环境迁移到 SFS 磁盘

本文介绍了如何将Notebook的Conda环境迁移到SFS磁盘上。这样重启Notebook实例后，Conda环境不会丢失。

步骤如下:

1. **创建新的虚拟环境并保存到SFS目录**
2. **克隆原有的虚拟环境到SFS盘**
3. **重新启动镜像激活SFS盘中的虚拟环境**
4. **保存并共享虚拟环境**

前提条件

创建一个Notebook，“资源类型”选择“专属资源池”，“存储配置”选择“SFS弹性文件服务器”，打开terminal。

创建新的虚拟环境并保存到 SFS 目录

创建新的conda虚拟环境。

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

查看现有的conda虚拟环境，此时可能出现新创建的虚拟环境的名称为空的情况。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
                      /home/ma-user/work/envs/user_conda/sfs-new-env
```

添加新创建的虚拟环境到conda env。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

查看现有的conda虚拟环境，此时新的虚拟环境已经能够正常显示，可以直接通过名称进行虚拟环境的切换。


```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8             /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10        *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env             /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）。

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-new-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-new-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-new-env”为举例，请以用户实际的安装路径为准。

图 9-1 安装路径回显

```
(PyTorch-1.8) [ma-user work]$ipython kernel install --user --name=sfs-clone-env
Installed kernelspec sfs-clone-env in /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env
(PyTorch-1.8) [ma-user work]$cd /home/ma-user/.local/share/jupyter/kernels/sfs-
sfs-clone-env/ sfs-new-env/
(PyTorch-1.8) [ma-user work]$cd /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/
(PyTorch-1.8) [ma-user sfs-clone-env]$ll
total 20
```

刷新JupyterLab页面，可以看到新的kernel。

说明

重启Notebook后kernel需要重新注册。

克隆原有的虚拟环境到 SFS 盘

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-clone-env --clone PyTorch-1.8 -
y
Source:      /home/ma-user/anaconda3/envs/PyTorch-1.8
Destination: /home/ma-user/work/envs/user_conda/sfs-clone-env
Packages: 20
Files: 39687
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate /home/ma-user/work/envs/user_conda/sfs-clone-env
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

查看新创建的clone虚拟环境，如果出现新创建的虚拟环境的名称为空的情况，可以参考**添加新创建到虚拟环境到conda env**。

```
# shell
conda env list
# conda environments:
#
```

```
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10            /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env            /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env           *  /home/ma-user/work/envs/user_conda/sfs-new-env
```

（可选）将新建的虚拟环境注册到JupyterLab kernel（可以在JupyterLab中直接使用虚拟环境）

```
# shell
pip install ipykernel
ipython kernel install --user --name=sfs-clone-env
rm -rf /home/ma-user/.local/share/jupyter/kernels/sfs-clone-env/logo-*
```

说明：此处“.local/share/jupyter/kernels/sfs-clone-env”为举例，请以用户实际的安装路径为准。

刷新JupyterLab页面，可以看到新的kernel。

重新启动镜像激活 SFS 盘中的虚拟环境

方法一，直接使用完整conda env路径。

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

方法二，先添加虚拟环境到conda env，然后使用名称激活。

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

方法三，直接使用完成虚拟环境中的python或者pip。

```
# shell
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/pip list
/home/ma-user/work/envs/user_conda/sfs-new-env/bin/python -V
```

保存并共享虚拟环境

将要迁移的虚拟环境打包。

```
# shell
pip install conda-pack
conda pack -n sfs-clone-env -o sfs-clone-env.tar.gz --ignore-editable-packages
Collecting packages...
Packing environment at '/home/ma-user/work/envs/user_conda/sfs-clone-env' to 'sfs-clone-env.tar.gz'
[########################################] | 100% Completed |  3min 33.9s
```

解压到SFS目录。

```
# shell

mkdir /home/ma-user/work/envs/user_conda/sfs-tar-env
tar -zxvf sfs-clone-env.tar.gz -C /home/ma-user/work/envs/user_conda/sfs-tar-env
```

查看现有的conda虚拟环境。

```
# shell
conda env list
# conda environments:
#
base                  /home/ma-user/anaconda3
```

```
PyTorch-1.8          *  /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10           /home/ma-user/anaconda3/envs/python-3.7.10
sfs-clone-env           /home/ma-user/work/envs/user_conda/sfs-clone-env
sfs-new-env             /home/ma-user/work/envs/user_conda/sfs-new-env
sfs-tar-env             /home/ma-user/work/envs/user_conda/sfs-tar-env
test-env                /home/ma-user/work/envs/user_conda/test-env
```

9.2 使用 ModelArts PyCharm 插件调试训练 ResNet50 图像分类模型

本案例介绍如何将本地开发好的MindSpore模型代码，通过PyCharm ToolKit连接到ModelArts进行云上调试和训练。

开始使用样例前，请仔细阅读**准备工作**罗列的要求，提前完成准备工作。本案例的步骤如下所示：

步骤1：安装和登录PyCharm ToolKit

步骤2：使用PyCharm进行本地开发调试

步骤3：使用ModelArts Notebook进行开发调试

步骤4：使用PyCharm提交训练作业至ModelArts

步骤5：清除相应资源

准备工作

- 本地已安装PyCharm 2019.2或以上版本，推荐Windows版本，社区版或专业版均可，请单击**PyCharm工具下载地址**获取工具并在本地完成安装。
  - 使用PyCharm ToolKit远程连接Notebook开发环境，仅限PyCharm专业版。
  - 使用PyCharm ToolKit提交训练作业，社区版和专业版都支持。
- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 已创建当前使用账号的访问密钥，并获得对应的AK和SK。如果未创建，请参见**创建访问密钥（AK和SK）**。
- 当前账号已完成访问授权的配置。如未完成，请参考**使用委托授权**。

环境说明

- Python 3.7.6
- PyCharm 2023.1.3 (Professional Edition)

  说明

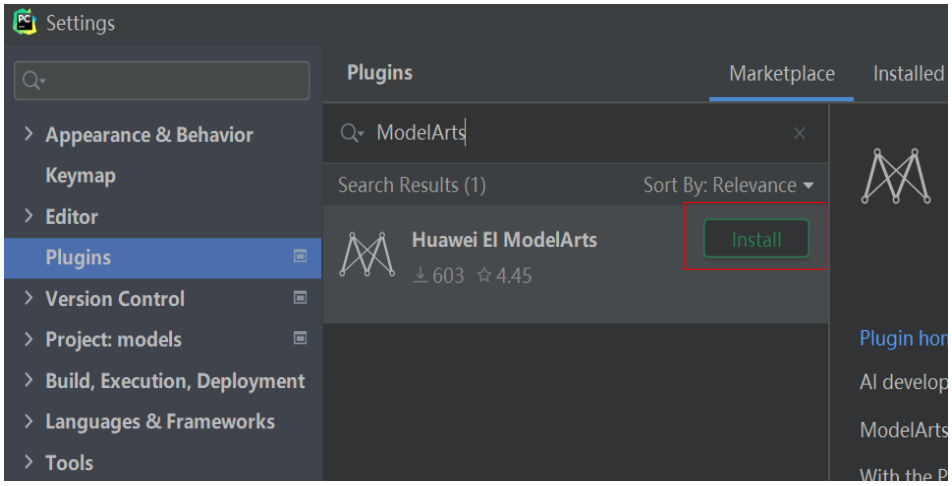
  本案例使用PyCharm版本为PyCharm 2023.1.3 (Professional Edition)，不同版本PyCharm之间部分界面可能不同，仅供参考。

步骤 1：安装和登录 PyCharm ToolKit

1. 安装PyCharm ToolKit。

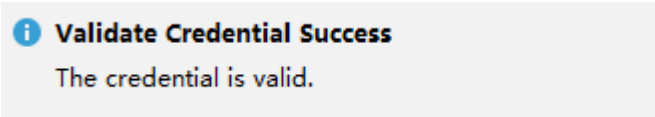
   在PyCharm中选择“File>Settings>Plugins”，在Marketplace里搜索“ModelArts”，单击“Install”即可完成安装。

图 9-2 通过 Marketplace 安装



2. 登录PyCharm ToolKit。
   a. 打开“Edit Credential”界面。
      安装完插件后，会在IDE菜单栏出现“ModelArts”，单击后选择“Edit Credential”。
   b. 验证登录信息。
      将创建访问密钥（AK和SK）输入到ToolKit对应位置，单击OK按钮进行登录，出现下图提示即为登录成功。
      如果未创建，请参见**创建访问密钥（AK和SK）**

      图 9-3 成功登录提示

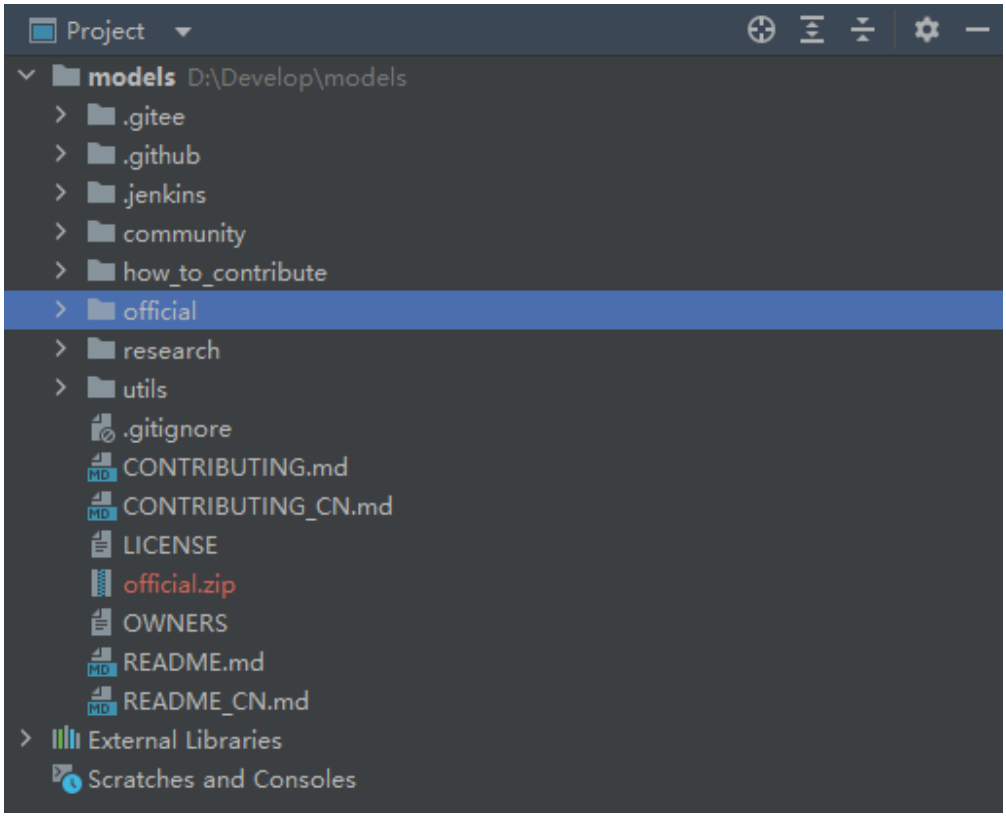
      

步骤 2：使用 PyCharm 进行本地开发调试

1. 下载代码至本地。
   本案例中，以图像分类模型resnet50模型为例，路径为“./models/official/cv/resnet/”

```
# 在本地电脑Terminal下载代码至本地
git clone https://gitee.com/mindspore/models.git -b v1.5.0
```

图 9-4 下载代码至本地



2. 配置本地PC开发环境。

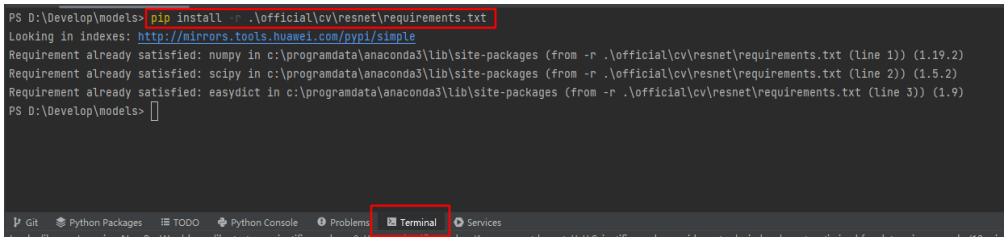
   修改“models/official/cv/resnet/requirements.txt”文件，改为：

   ```
   numpy==1.17.5
   scipy==1.5.4
   easydict==1.9
   ```

   执行pip命令安装：

   ```
   # 在PyCharm的Terminal安装mindspore
   pip install mindspore==1.7.0 --trusted-host https://repo.huaweicloud.com -i https://
   repo.huaweicloud.com/repository/pypi/simple
   # 在PyCharm的Terminal安装resnet依赖
   pip install -r .\official\cv\resnet\requirements.txt --trusted-host https://repo.huaweicloud.com -i https://
   repo.huaweicloud.com/repository/pypi/simple
   ```

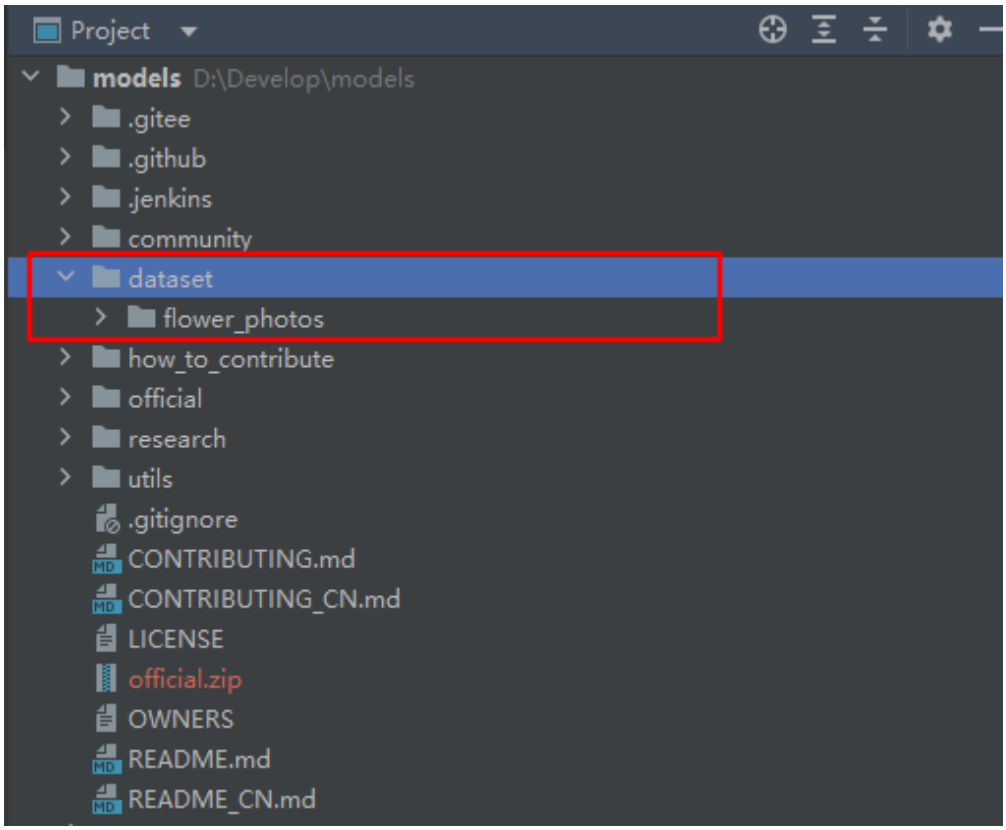
图 9-5 安装 resnet 依赖



3. 准备数据集。

   本样例使用的数据集为类别数为五类的花卉识别数据集，**下载数据集**并解压数据到工程目录。新建dataset文件夹，将解压后数据集保存在dataset文件夹下。

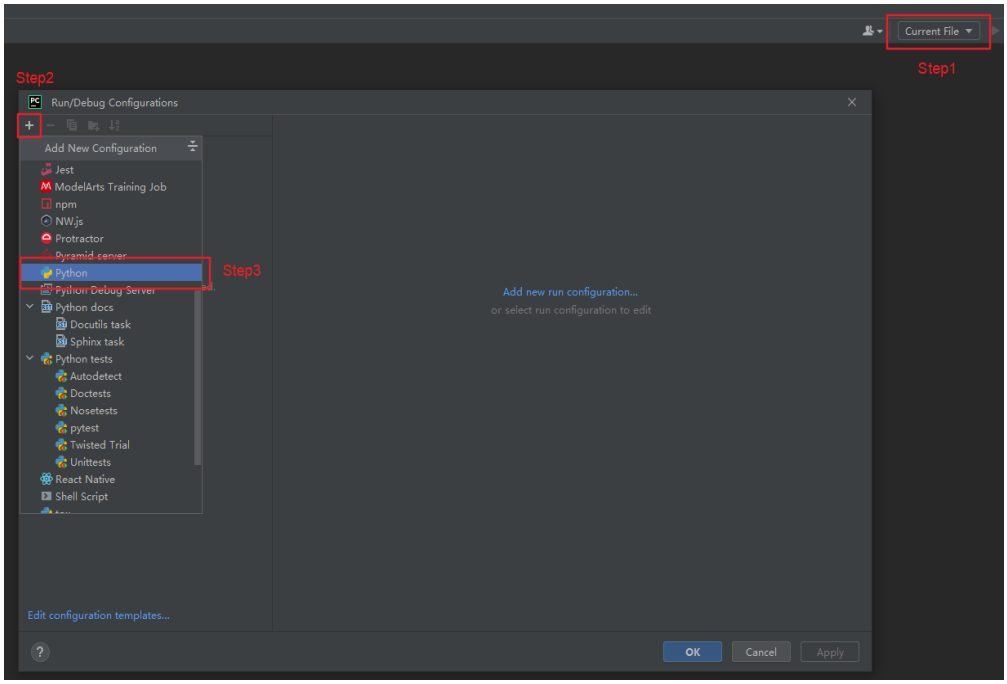
图 9-6 准备数据集



4. 配置PyCharm解释器和入参。

   单击右上角“Current File”，选择“Edit Configuration”，打开“Run/Debug Configuration”对话框。在对话框中单击“+”，选择“Python”。

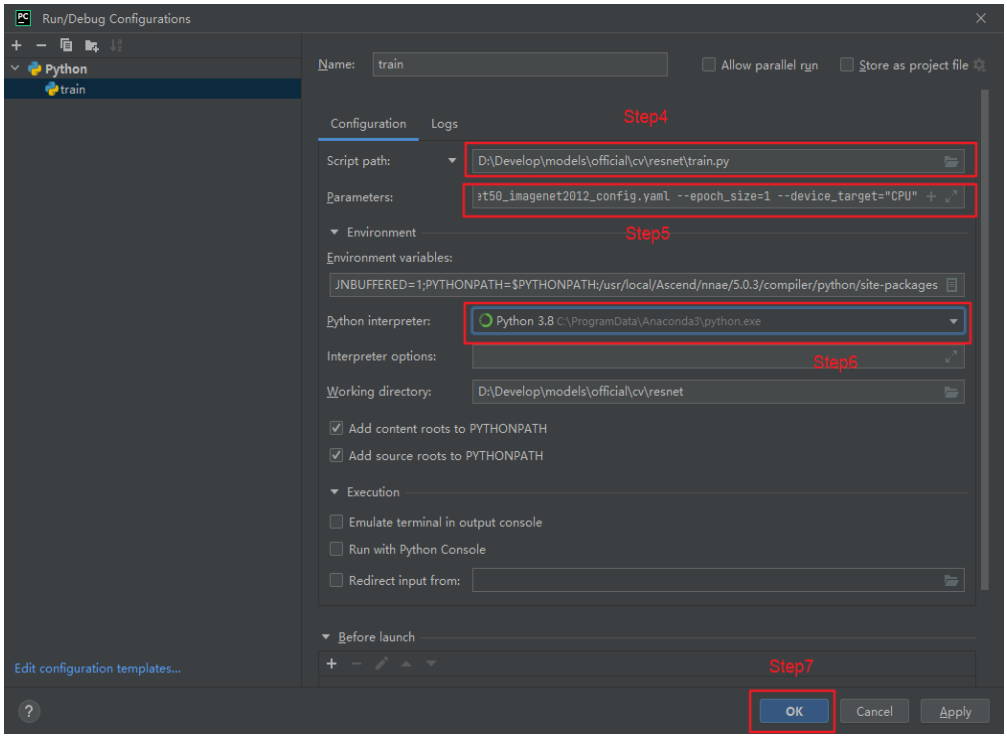
图 9-7 前往 PyCharm 解释器



“Script path”选择train.py文件，“Parameters”命令如下所示，并选择Python解释器，然后单击“OK”：

```
--net_name=resnet50 --dataset=imagenet2012 --data_path=../../../dataset/flower_photos/ --class_num=5 --config_path=./config/resnet50_imagenet2012_config.yaml --epoch_size=1 --device_target="CPU"
```

图 9-8 配置 PyCharm 解释器



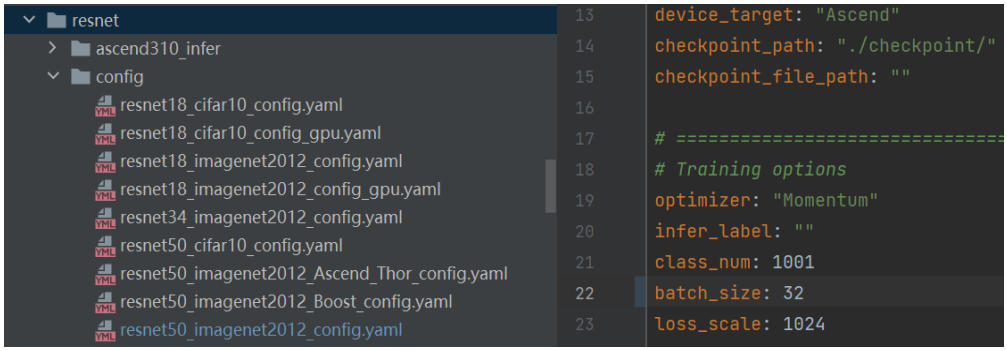
说明

根据README说明文档，配置Parameter参数device_target="CPU"表示CPU环境运行，device_target="Ascend"表示在Ascend环境运行。

5. 本地代码开发调测。

一般本地CPU算力较低并且内存较小，可能出现内存溢出的报错，因此可以把“models/official/cv/resnet/config/resnet50_imagenet2012_config.yaml”的“batch_size”由“256”改为“32”，使得训练作业可以快速运行。

图 9-9 修改 batch_size



AI开发过程中的数据集开发及模型开发是和硬件规格无关的，而且这一部分的开发耗时是最长的，因此可以先在本地PC的CPU环境进行数据集和模型开发调试。

说明

本例中，因为样例代码已经支持在CPU上进行训练，因此用户能够在CPU上完成整个训练流程。如果代码只支持在GPU或者Ascend上训练，那么可能会报错，需要使用Notebook进行云端调试。

设置断点后单击“调试”，可实现代码逐步调试，查看中间变量值。

图 9-10 “调试”按钮

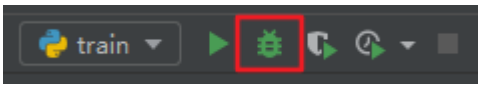
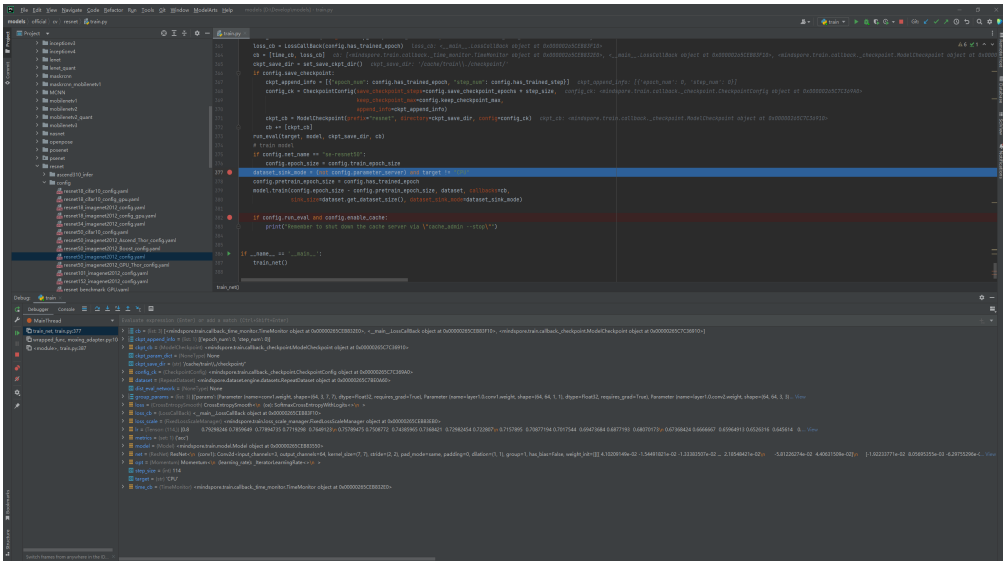


图 9-11 通过设置断点实现代码调试



可单击“运行”按钮，通过日志观察是否能正常训练。

图 9-12 “运行”按钮

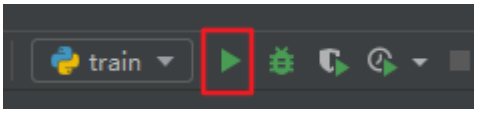
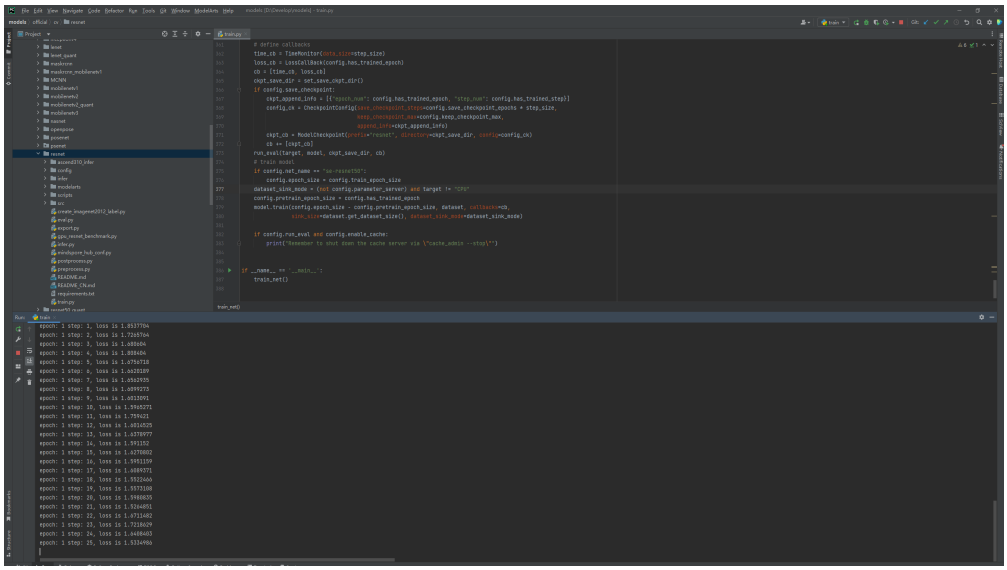


图 9-13 训练日志



步骤 3：使用 ModelArts Notebook 进行开发调试

使用ModelArts Notebook进行开发调试具有如下优势：

- 环境保持一致
- 配置一键完成
- 代码远程Debug
- 资源按需使用

说明

只有PyCharm专业版支持本章节，社区版可以直接跳转至**步骤4：使用PyCharm提交训练作业至ModelArts**完成创建训练作业。

1. 连接Notebook开发环境。
   a. 创建或打开云端Ascend规格的Notebook。创建Notebook详细操作请参见**创建Notebook实例**，Notebook规格相关信息如下所示：

   “镜像”：tensorflow1.15-mindspore1.7.0-cann5.1.0-euler2.8-aarch64。

   “资源选择”：公共资源池。

   “类型”：ASCEND。

   “规格”：选Ascend类型的，以界面实际可选值为准。

   “存储配置”：EVS存储。

   “SSH远程开发”：开启。

   “密钥对”：选择已有密钥对，或单击密钥对右侧的“立即创建”创建密钥对。
2. 通过ToolKit连接云端Notebook。
   a. 在IDE菜单栏中选择“ModelArts>Notebook>Remote Config”，在打开的界面中选择要连接的Notebook实例。

说明

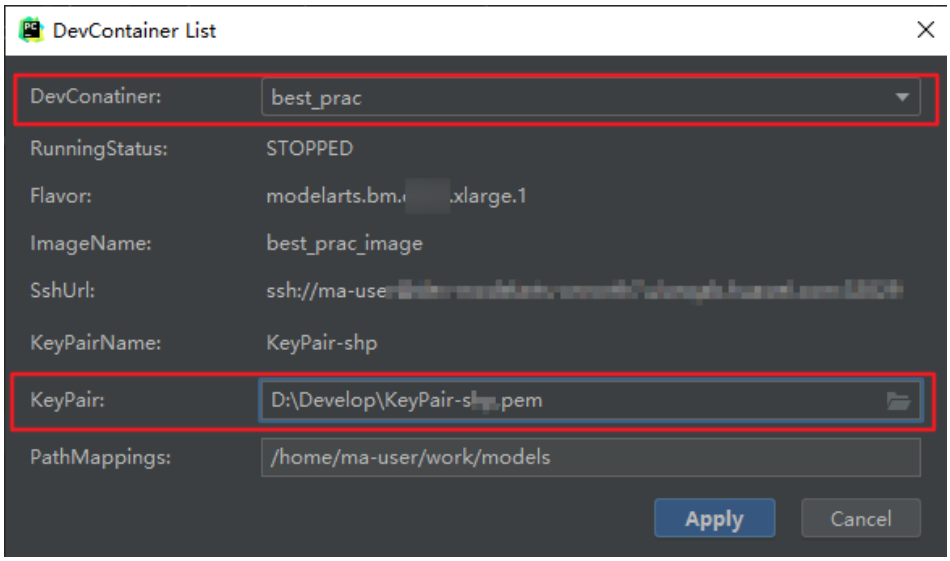
此处如果看不到Connect to Remote选项，请先参考**创建Notebook实例**章节，创建Notebook实例，并开启该实例的SSH远程开发功能。

也可能是PyCharm ToolKit的版本不正确，请按照文档要求下载新版本的PyCharm ToolKit。

下载前请先清除浏览器缓存，如果之前下载过老版本的PyCharm ToolKit，浏览器会有缓存，可能会导致新版本下载失败。

b. 在KeyPair中选择该Notebook实例对应的密钥，选择完成后，单击Apply进行远程Notebook一键配置，等待一段时间后，会出现重启IDE的确认框，单击确认重启，重启后即可生效。

图 9-14 ToolKit 连接 Notebook 配置界面



说明

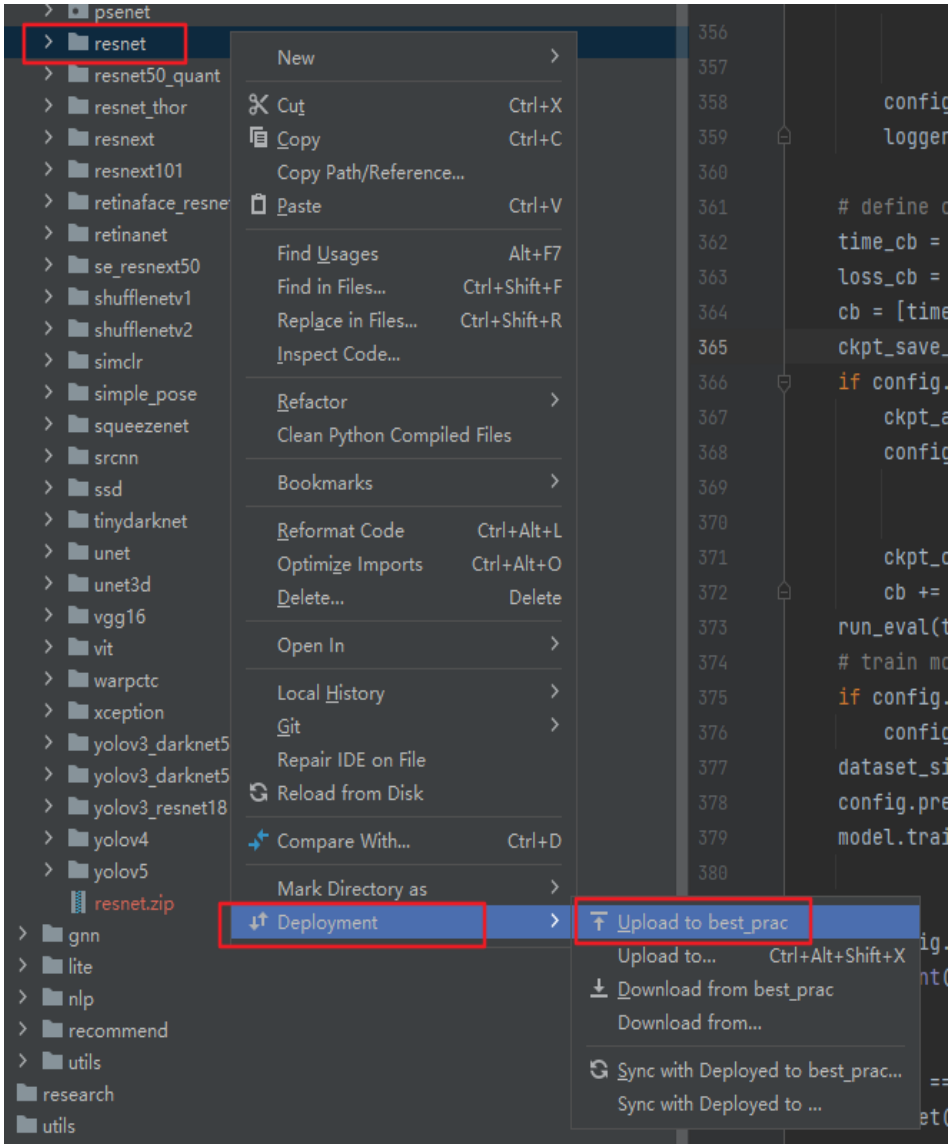
- KeyPair: 需要选择保存在本地的Notebook对应的keypair认证。即创建Notebook时创建的密钥对文件，创建时会直接保存到浏览器默认的下载文件夹中。
- PathMappings: 该参数为本地IDE项目和Notebook对应的同步目录，默认为“/home/ma-user/work/project”，可根据自己实际情况更改。

3. 同步代码和数据至云端Notebook。

a. 将代码同步至Notebook。

选择resnet文件夹，右键选择“Deployment>Upload to”上传代码至Notebook。

图 9-15 同步代码至 Notebook



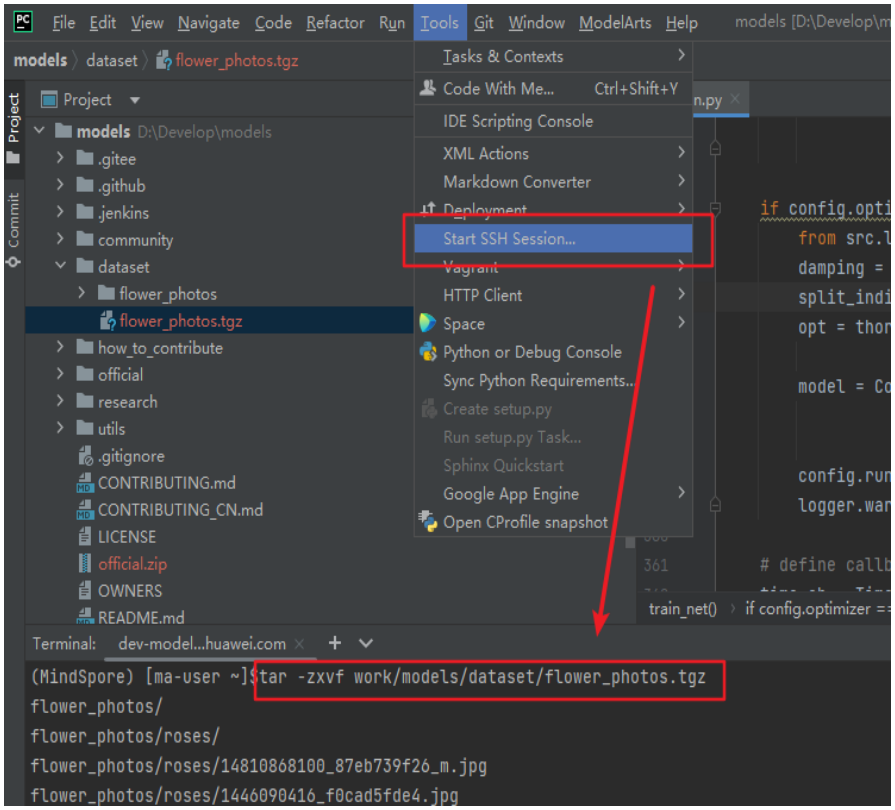
b. 将数据同步至Notebook。

- （推荐）方法一：数据集压缩包上传至Notebook后解压

  把数据集压缩包右键选择“Deployment>Upload to”的方式上传至Notebook后，在Notebook中对数据集进行解压操作，解压命令如下：

  ```
  tar -zxvf work/models/dataset/flower_photos.tgz
  ```

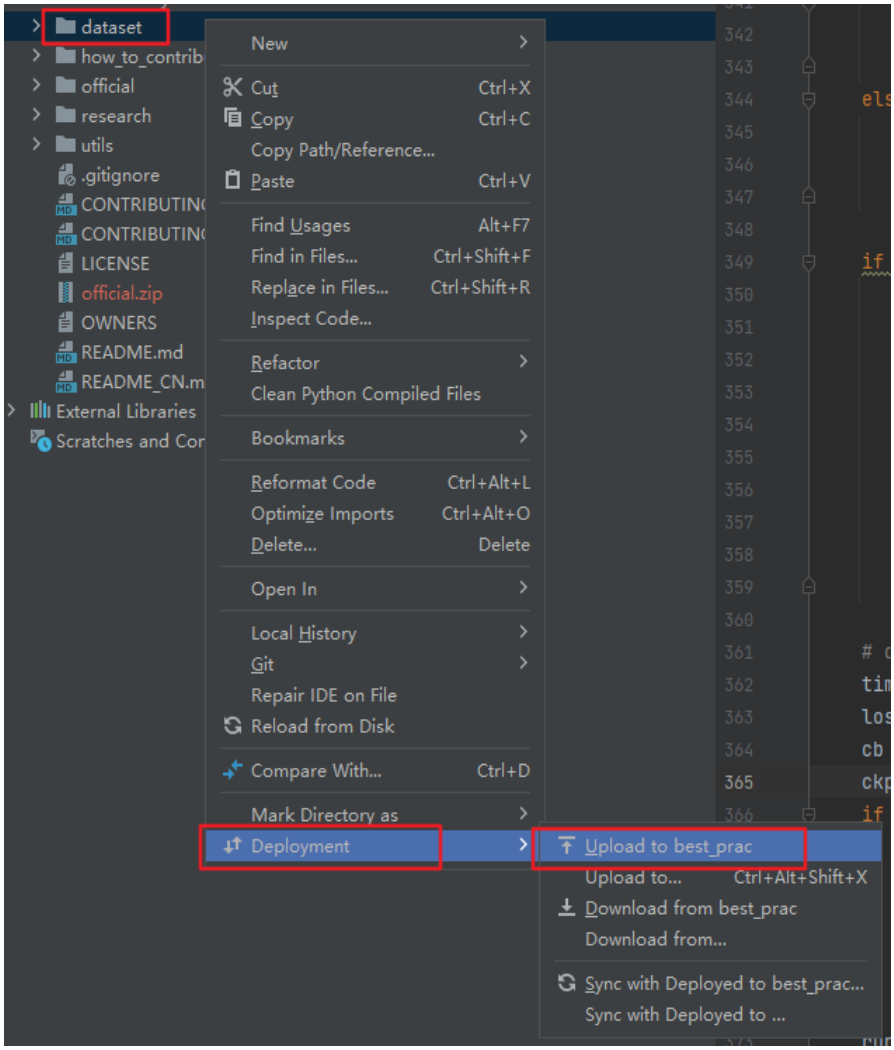
图 9-16 数据集压缩包上传至 Notebook 后解压



- 方法二：文件夹直接上传至Notebook。

  类似上传代码至Notebook，直接上传数据文件夹。（由于本案例数据集中图片数量较多，通过IDE进行上传比较耗时，推荐使用方法一进行上传）

图 9-17 文件夹直接上传至 Notebook



注意

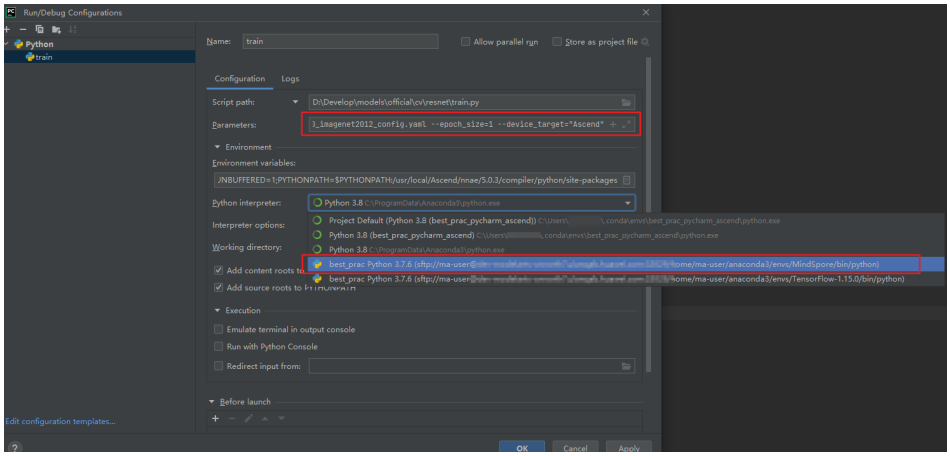
- 当数据集比较大达到数GB时，建议先将数据集先上传至OBS再通过OBS上传至Notebook，PyCharm只适合做小文件的同步上传。
- 调试时建议使用较小的数据集子集，方便数据同步与数据加载。

4. 配置云端Python解释器。

   修改Parameters参数，并选择云端Python解释器。

```
--net_name=resnet50 --dataset=imagenet2012 --data_path=../../../dataset/flower_photos/ --class_num=5 --config_path=./config/resnet50_imagenet2012_config.yaml --epoch_size=1 --device_target="Ascend"
```

图 9-18 配置云端 python 解释器

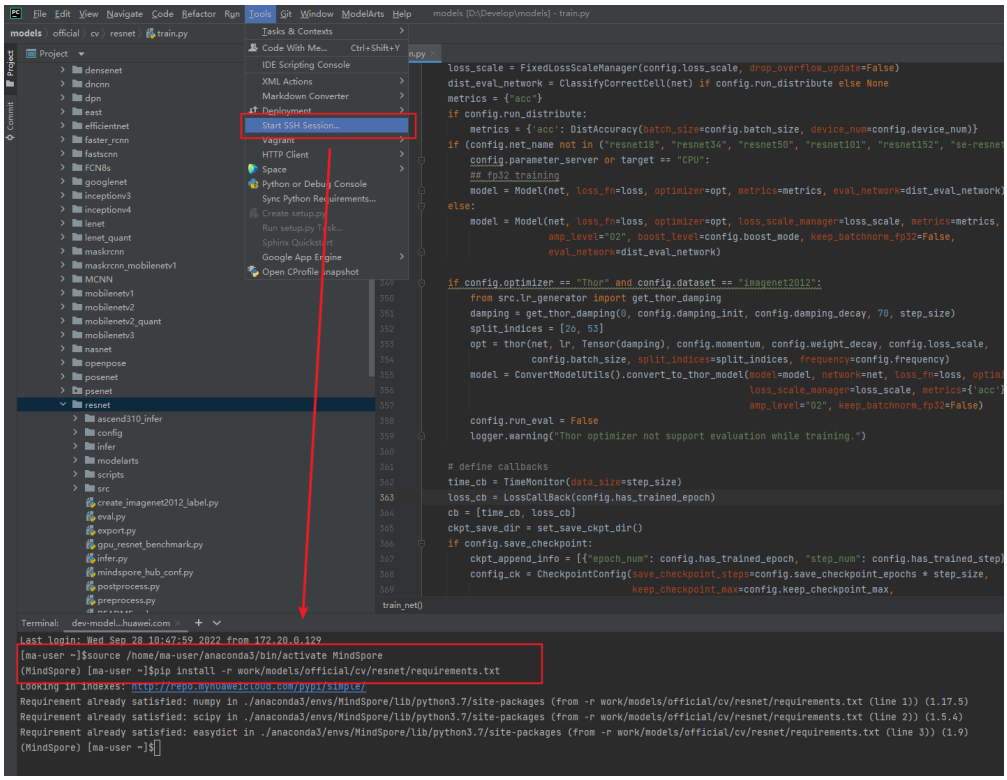


5. 云端Notebook安装依赖。

   打开“Tool>Start SSH Section”，安装依赖软件。

```
# 进入MindSpore环境
source /home/ma-user/anaconda3/bin/activate MindSpore
# 安装resnet依赖
pip install -r work/models/official/cv/resnet/requirements.txt
```

图 9-19 云端 Notebook 安装依赖



6. 云端调试与运行。

   配置完云端的解释器后，PyCharm可以直接使用远端Notebook中的python解释器和硬件规格，满足用户在本地体验到真实的硬件环境并进行全流程的调试和验证。

⚠ 注意

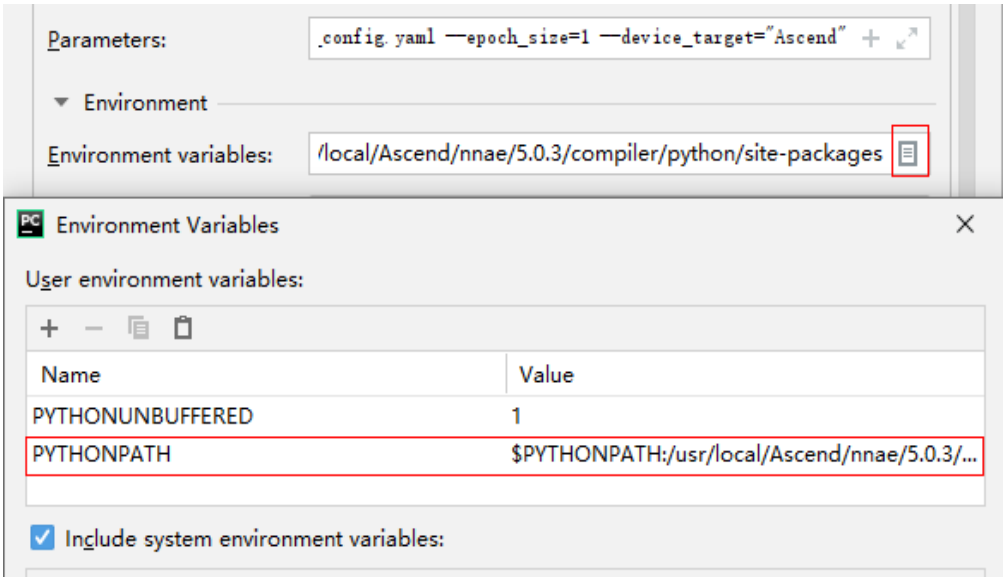
基于Ascend的样例中，可能会抛出异常。

```
ModuleNotFoundError: No module named 'te'
```

原因是：PyCharm的PYTHONPATH会将Notebook中的环境变量中指定的“PYTHONPATH”进行覆盖，因此，还需要将te包所在的路径添加到PyCharm的“PYTHONPATH”中。

te包的路径通过“pip show te”查看，例如te包返回对应的路径为：“/usr/local/Ascend/nnae/5.0.3/compiler/python/site-package”，则“PYTHONPATH”对应的“Value”为“$PYTHONPATH:/usr/local/Ascend/nnae/5.0.3/compiler/python/site-package”

图 9-20 将 te 包所在的路径添加到 PyCharm 的 PYTHONPATH 中



7. 保存开发环境镜像。

   成功完成Notebook调测后，此时的Notebook已经包含了模型训练所有的依赖环境，因此可以将已经调测完成的开发环境保存成一个镜像，选择“Notebook>更多>保存镜像”。此时Notebook会冻结，需要等待几分钟（只需要保存一次）。

   保存后的镜像可以在“ModelArts>镜像管理”中进行查看，对应完整的镜像名称为“详情->SWR地址”。

   图 9-21 查看保存后的镜像

   说明

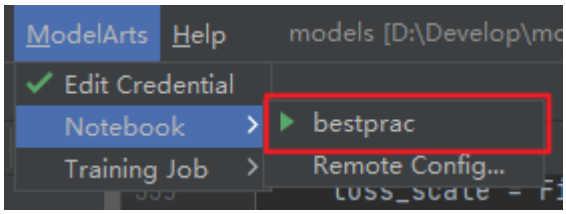
   Notebook在代码调试完成及保存镜像后就可以关闭了，减少资源浪费。

8. 连接、停止、启动和断开Notebook实例。

- 连接Notebook实例。

  当Notebook实例为绿色三角形状态时，表示该实例运行中（但未与PyCharm连接）。此时单击该实例名称，实例会变为绿色勾状态，表示PyCharm已与实例连接成功。

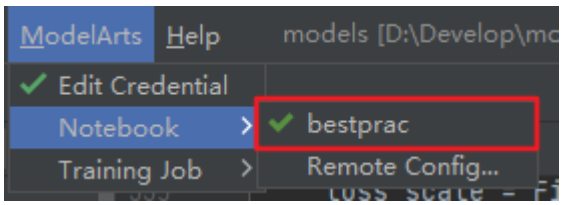
  图 9-22 实例运行中状态

- 停止Notebook实例。

  当Notebook实例为绿色勾状态时，表示该实例运行中且与PyCharm连接成功。此时单击该实例名称，实例会变为黄色感叹号状态，表示停止Notebook实例。

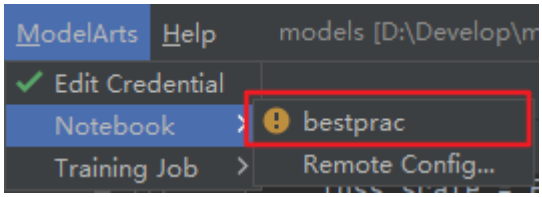
  图 9-23 实例运行中且与 PyCharm 连接成功状态

- 启动Notebook实例。

  当Notebook实例为黄色感叹号状态时，表示该实例已停止。此时单击该实例名称，实例会变为绿色勾状态，表示启动Notebook实例且与PyCharm连接成功（默认启动时间为4小时）。

  图 9-24 实例已停止状态

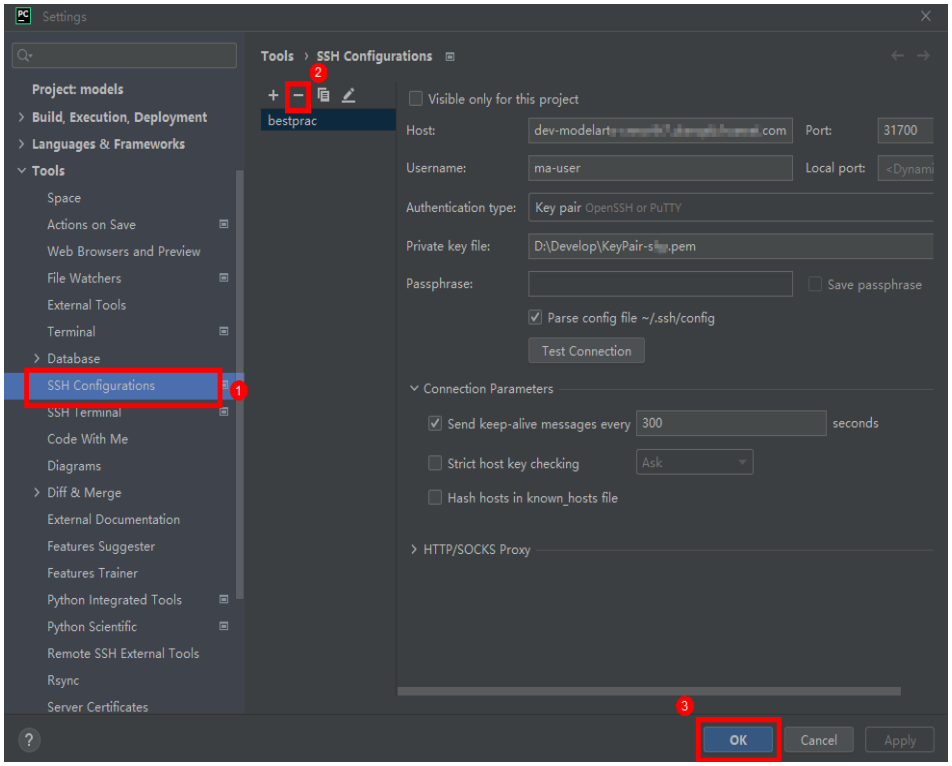
- 断开PyCharm ToolKit中的Notebook实例SSH连接。

  选择“File>Settings>Tool>SSH Configurations”，单击需要断开的实例，选择“-”，单击“OK”，则IDE菜单栏“ModelArts>Notebook”中的Notebook实例连接断开。

⚠ 注意

该步骤会使Notebook实例不在PyCharm ToolKit中呈现，但Notebook实例仍然存在于控制台。如果想删除Notebook实例以释放资源，请登录ModelArts管理控制台，在Notebook管理页面进行删除。

图 9-25 断开 PyCharm ToolKit 中的 Notebook 实例 SSH 连接



步骤 4：使用 PyCharm 提交训练作业至 ModelArts

ModelArts训练平台提供了海量的算力规格和训练优化，支持将本地调试好的代码以及之前保存的开发环境镜像直接在PyCharm中提交训练作业。

1. 创建OBS桶并上传数据。

   由于训练作业是在ModelArts端运行，因此需要把训练数据和训练代码上传至云端Notebook。可借助OBS Browser+把下载好的训练数据上传至OBS，具体安装步骤请见**安装OBS Browser+**。

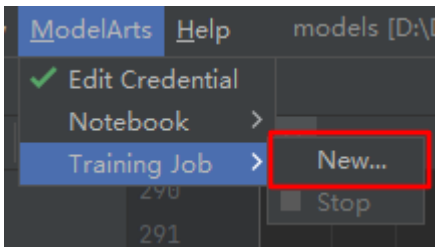
   新建data-flower桶，把训练数据flower_photos文件夹通过OBS Browser+上传至对应的OBS，并新建train文件夹用来存放训练作业相关数据。

图 9-26 上传数据至 OBS

2. 创建训练作业。

   在IDE菜单栏选择“ModelArts>Training Job>New...”创建训练作业。

图 9-27 创建训练作业



创建训练作业界面各参数名称及含义如下表所示。

表 9-1 参数名称及含义

参数名称	含义
JobName	训练作业的名称，默认为当前的时间。
AI Engine	训练引擎，这里选择“mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64”
Boot File Path	本地训练启动代码。
Code Directory	本地代码目录
Image Path(optional)	可选项，输入自定义镜像swr路径地址（使用的自定义镜像和预置的训练镜像引擎一致）
Data OBS Path	OBS上的数据集路径（需要提前把数据上传到OBS中）
Training OBS Path	OBS路径（该路径必须是存在的），用于保存代码和训练模型及日志的输出

参数名称	含义
Running Parameters	训练脚本接收的参数。
Specifications	计算规格，这里选择Ascend类型的，以界面实际可选值为准。
Compute Node	节点数（单机训练默认为1）

PyCharm中支持两种方式创建训练作业：使用预置镜像训练作业、自定义镜像创建训练作业。

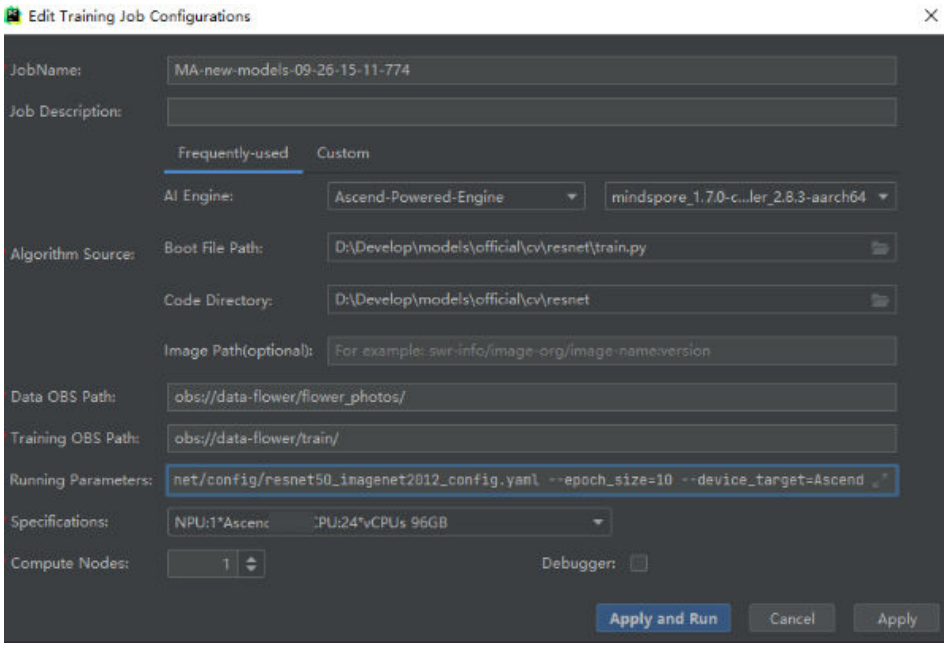
- 使用预置镜像创建训练作业。

  在RunningParameters中填入如下训练参数，其余参数按实际路径填写。

  ```
  --net_name=resnet50 --dataset=imagenet2012 --enable_modelarts=True --class_num=5 --config_path=/home/ma-user/modelarts/user-job-dir/resnet/config/resnet50_imagenet2012_config.yaml --epoch_size=10 --device_target=Ascend
  ```

  填写完训练作业参数后，单击“Apply and Run”即完成训练作业创建。

  图 9-28 使用预置镜像创建训练作业

- 使用自定义镜像创建训练作业。

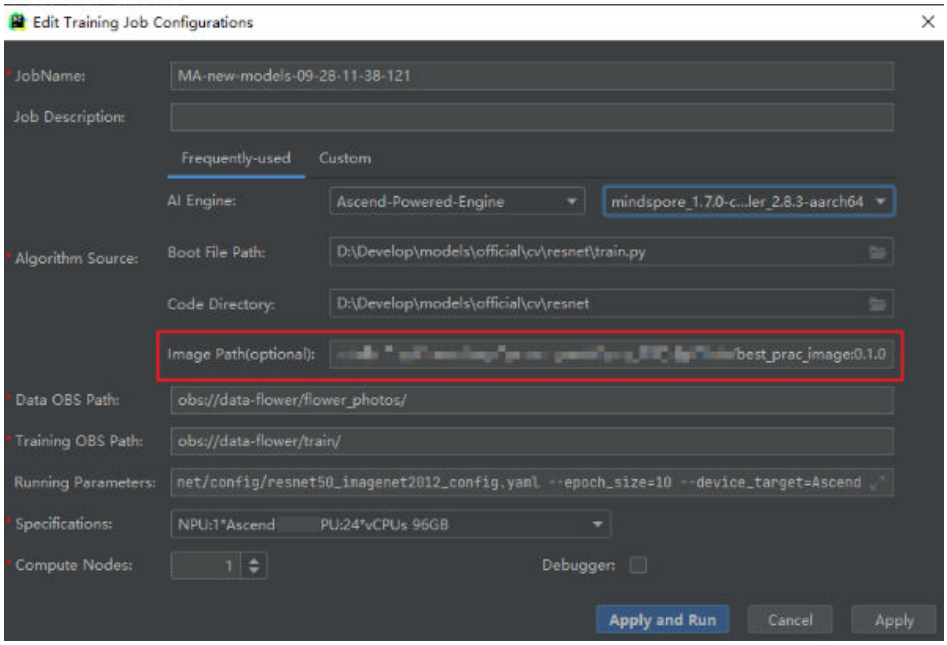
  使用自定义镜像创建训练作业和使用预置镜像创建训练作业的差别，在于Image Path处填入了自定义镜像的地址。填写完训练作业参数后，单击“Apply and Run”即完成训练作业创建。

  说明

  在选择AI Engine预置镜像时，需要和自定义镜像保持一致，该设置的作用为通过预置镜像的启动命令启动自定义镜像。

  例如自定义镜像中用到Mindspore，则预置镜像中可选择包含Mindspore的镜像。

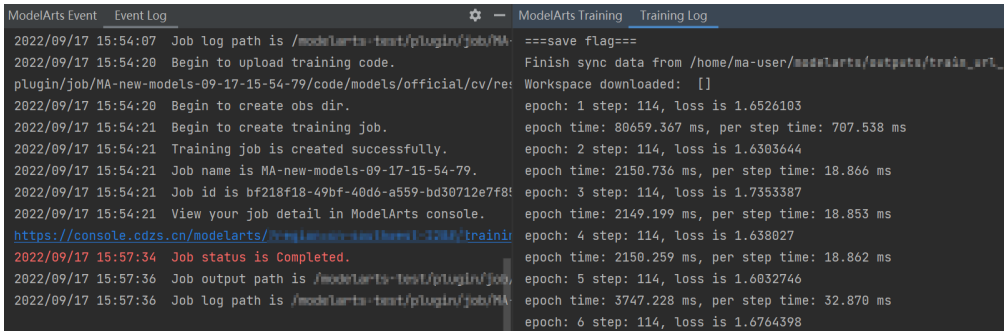
图 9-29 使用自定义镜像创建训练作业



3. 查看训练日志。

   在单击“Apply and Run”按钮后，训练的日志可以在PyCharm窗口中实时展示。也可以单击Event Log中的控制台链接，转调到网页端中查看训练日志。

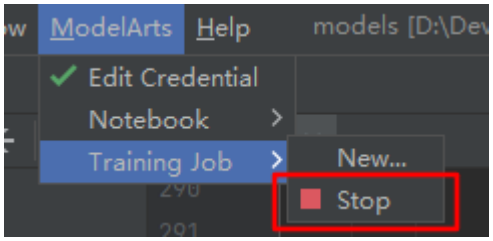
图 9-30 在 PyCharm 中查看训练日志



4. 终止训练作业。

   如果想要在中途终止训练，可以在PyCharm中单击“ModelArts>Training Job>Stop”，或者直接在网页端单击终止。

图 9-31 终止训练作业



步骤 5：清除相应资源

为避免产生不必要的费用，在完成试用后，建议您删除相关资源，如在线服务、训练作业及其OBS目录。

- 停止Notebook：在“Notebook”页面，单击对应实例操作列的“停止”。
- 在PyCharm菜单栏中，选择“ModelArts > Stop Training Job”停止此训练作业。
- 进入OBS管理控制台，删除创建的OBS桶。先逐个删除桶内文件夹和文件，再执行删除桶的操作。

9.3 使用 ModelArts VSCode 插件调试训练 ResNet50 图像分类模型

应用场景

Notebook等线上开发工具工程化开发体验不如IDE，但是本地开发服务器等资源有限，运行和调试环境大多使用团队公共搭建的CPU或GPU服务器，并且是多人共用，这带来一定的环境搭建和维护成本。因此使用本地IDE+远程Notebook结合的方式，可以同时享受IDE工程化开发和云上资源的即开即用，优势互补，满足开发者需求。

VS Code在Python项目开发中提供了优秀的代码编辑、调试、远程连接和同步能力，在开发者中广受欢迎。本文以Ascend Model Zoo为例，介绍如何通过VS Code插件及ModelArts Notebook进行云端数据调试及模型开发。

方案优势

云端开发调试优势：

- 环境保持一致
- 配置一键完成
- 代码远程调试
- 资源按需使用

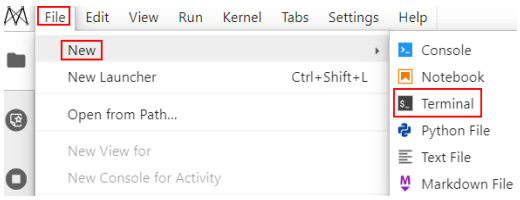
准备工作

1. 下载VS Code IDE，下载路径：**开源Visual Studio Code**。根据不同的操作系统选择不同的安装包。
2. 创建Notebook实例。
   a. **登录ModelArts控制台**，单击左侧导航“开发环境 > Notebook”，然后单击“创建”。

      镜像选择“mindspore1.7.0-cann5.1.0-py3.7-euler2.8.3”，类型选择“ASCEND”，并打开“SSH远程开发”开关，密钥对选择已有的或单击“立即创建”。

   b. Notebook创建后，“状态”为“运行中”。单击“操作”列的“打开”，进入JupyterLab，然后参考下图打开Terminal。

图 9-32 打开 Terminal

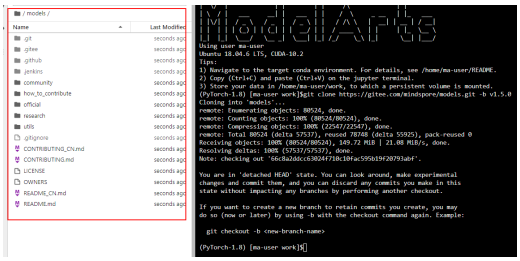


3. 下载项目代码。

   在Terminal执行如下命令下载项目代码。本例中，以图像分类模型resnet50模型为例。下载后的文件如图9-33所示，代码所在路径为“./models/official/cv/resnet/”。

   ```
   # 下载代码
   git clone https://gitee.com/mindspore/models.git -b v1.5.0
   ```

   图 9-33 下载后的模型包文件

4. 下载花卉识别数据集。

   本样例使用的数据集为类别数为五类的花卉识别数据集。

   在Terminal里执行如下命令下载并解压数据集，将数据集保存在“./models/dataset/flower_photos”文件夹。

   ```
   cd models
   mkdir dataset
   cd dataset
   wget http://download.tensorflow.org/example_images/flower_photos.tgz
   tar zxvf flower_photos.tgz
   ```

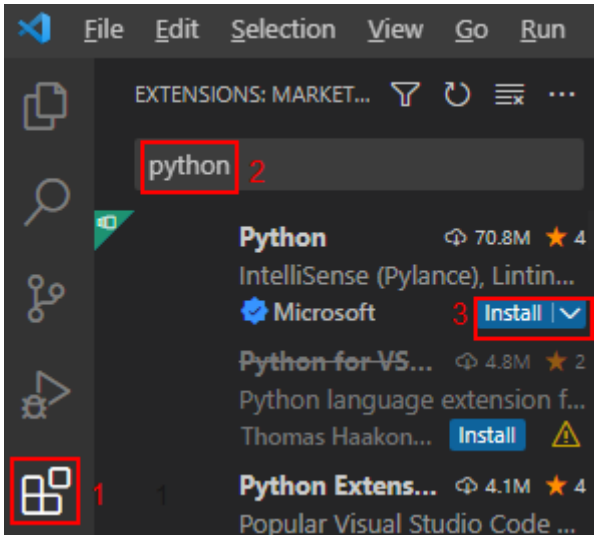
步骤 1：通过 VS Code 插件连接云端 Notebook

通过VS Code插件连接**准备工作**里创建的云端Notebook，详细操作请参考**VS Code一键连接Notebook**。

步骤 2：安装 Python 插件以及配置入参

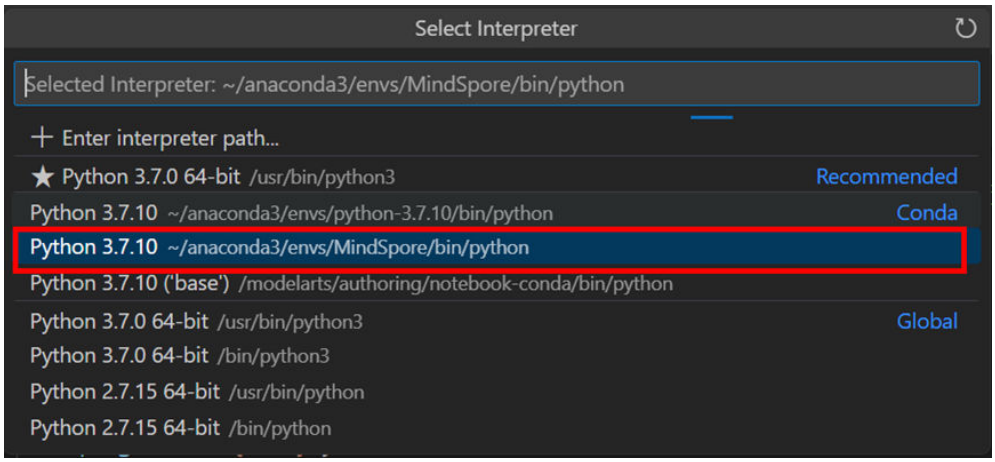
1. 打开VS Code工具，单击“Extensions”，搜索python，然后单击“Install”。

图 9-34 安装 Python



2. 输入Ctrl+Shirt+P，搜索“python:select interpreter”，选择Python解释器。

图 9-35 选择 python 解释器



3. 单击“RUN > Add Configuration...” 选择Python > Python File，填入如下代码。

   如果文件已创建，单击“RUN > Open Configurations”，填入如下代码。

   # 根据README说明文档，配置的Parameter入参如下，其中device_target="CPU"表示CPU环境运行，device_target="Ascend"表示在Ascend环境运行

```
"configurations": [
    {
        "name": "Python: Django Debug Single Test",
        "type": "python",
        "request": "launch",
        "program": "${file}",
        "args":[
            "--net_name", "resnet50",
            "--dataset", "imagenet2012",
            "--data_path", "/home/ma-user/work/models/dataset/flower_photos/",
            "--class_num", "5",
            "--config_path", "/home/ma-user/work/models/official/cv/resnet/config/
```

```
resnet50_imagenet2012_config.yaml",
                "--epoch_size", "1",
                "--device_target","Ascend"
            ]
        }
    ]
```

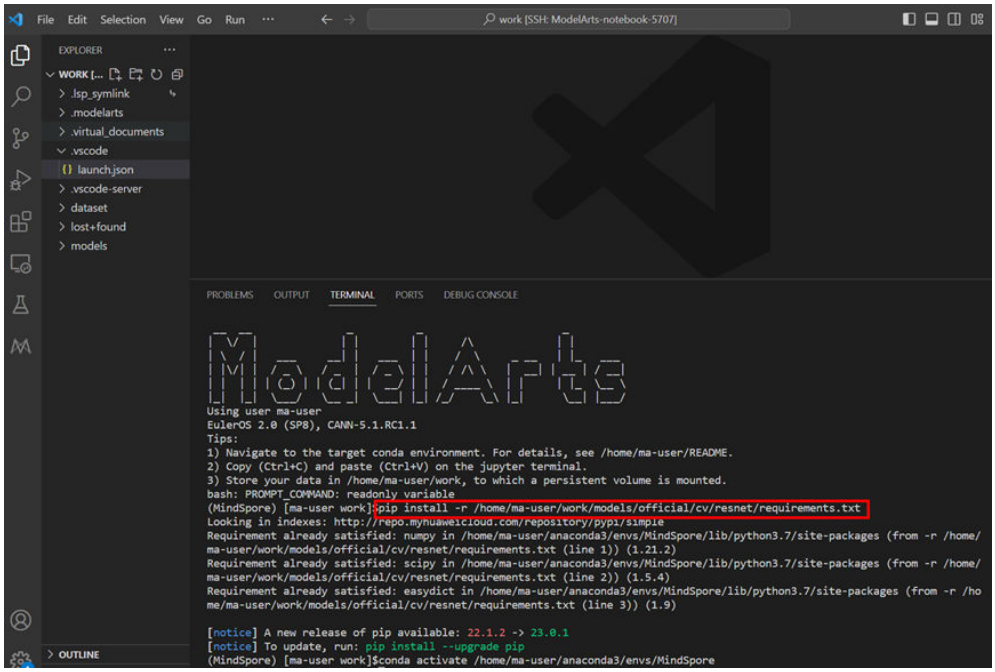
步骤 3：在 VS Code 中远程调试代码

1. 参考**准备工作**上传本地代码和数据至云端Notebook。
2. 云端Notebook安装依赖。

   在本地IDE中打开“Terminal > New Terminal”，执行如下命令。

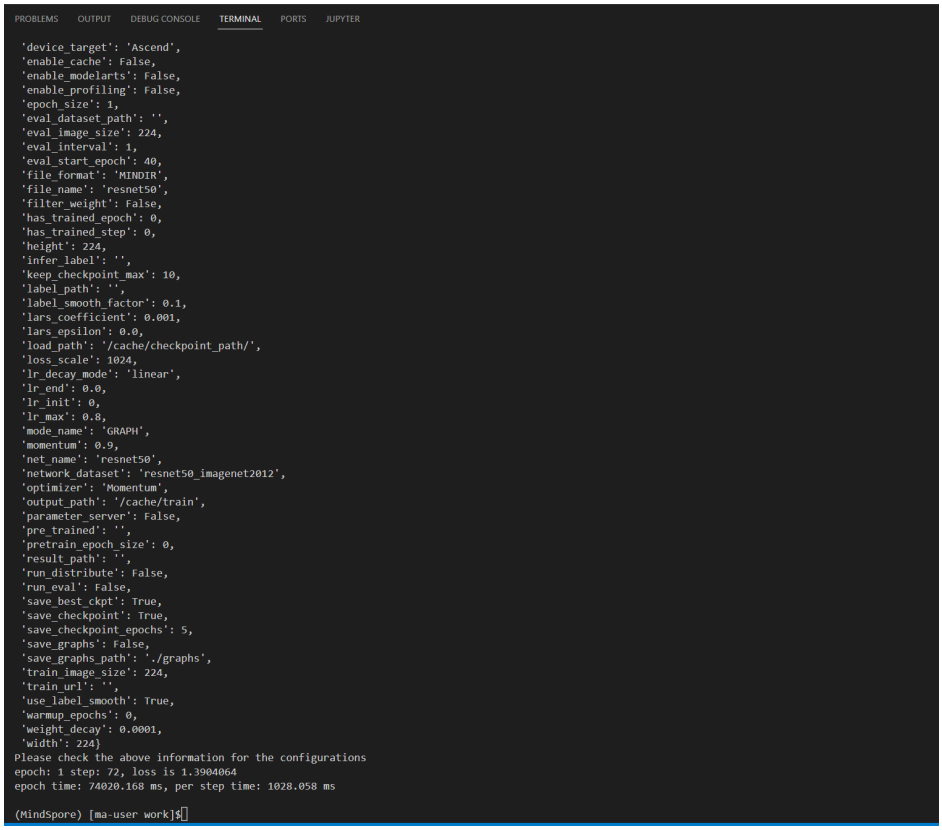
   ```
   pip install -r /home/ma-user/work/models/official/cv/resnet/requirements.txt
   ```

   图 9-36 执行命令

3. 云端调试与运行。
   a. 打开训练文件。文件所在路径为“/home/ma-user/work/models/official/cv/resnet/train.py”
   b. 代码调测：在需要调测点打断点，然后单击“RUN > Start Debugging”。
   c. 代码运行：单击“RUN > Run Without Debugging”，运行结果如下：

图 9-37 代码运行结果



步骤 4：保存开发环境镜像

完成Notebook调测后，此时的Notebook已经包含了模型训练所有的依赖环境，因此可以将已经调测完成的开发环境保存成一个镜像。

- 方式一：保存镜像需要指定镜像名称、镜像标签、SWR服务的组织等信息，保存镜像需要等待几分钟时间，期间不能对Notebook有额外操作。

  SWR服务的组织可以在SWR服务中进行创建，也可以使用SDK创建默认的SWR组织，默认最多只能创建5个组织。

  a. 在“/home/ma-user/work/models/official/cv/resnet/”下创建save_image.py，

  b. 复制代码至save_image.py，

  c. 运行save_image.py，进行保存镜像。

  save_image.py代码如下：

```
# save_image.py
# 导入ModelArts SDK的依赖，并初始化Session，此处的ak、sk、project_id、region_name请替换成用户自己的信息
from modelarts.session import Session
# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')
```

```
# 保存notebook镜像
from modelarts.image_mgmt import ImageSave
from modelarts.service import SWRManagement
# 创建一个镜像组织。如果组织数量已超过阈值，则会报错“namespace is invalid”，需要删除一个组织或手动指定一个已有的组织信息（使用image_organization = “your-swr-namespace-name”指定）
image_organization = SWRManagement(session).get_default_namespace()
# image_organization = “your-swr-namespace-name”
print("Default image_organization:", image_organization)
image_name = "mindspore-image-models-image" #@param {type:"string"}
image_tag = "1.0.0" #@param {type:"string"}
image_save = ImageSave(session=session, name=image_name, tag=image_tag, organization=image_organization)
image_save.save()
```

- 方式二：在ModelArts控制台单击“保存镜像”。

  在Notebook列表中，对于要保存的Notebook实例，单击右侧“操作”列的“更多 > 保存镜像”，进入“保存镜像”页面，设置组织、镜像名称、镜像版本和描述信息后单击“确认”保存镜像。此时Notebook会冻结，需要等待几分钟。详细操作请参考**保存Notebook镜像环境**。

  图 9-38 保存镜像

查看所保存的镜像

保存后的镜像可以在ModelArts控制台“镜像管理”页面查看到该镜像详情。单击镜像的名称，进入镜像详情页，可以查看镜像版本/ID，状态，资源类型，镜像大小，SWR地址等。

步骤 5：使用 SDK 提交训练作业

本地调测完成后可以提交训练作业。因为数据在Notebook中，设置InputData中“is_local_source”的参数为“True”，会自动将本地数据同步上传到OBS中。

步骤如下：

1. 在“/home/ma-user/work/models/official/cv/resnet/”下创建train_notebook.py，
2. 复制代码至train_notebook.py，
3. 运行train_notebook.py，进行训练作业提交。

```
# train_notebook.py
# 导入ModelArts SDK的依赖，并初始化Session，此处的ak、sk、project_id、region_name请替换
```

```
成用户自己的信息
from modelarts.train_params import TrainingFiles
from modelarts.train_params import OutputData
from modelarts.train_params import InputData
from modelarts.estimatorV2 import Estimator
from modelarts.session import Session

# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')

# 样例中为了方便默认创建一个OBS桶，推荐将调测所需要传输的文件统一放到`${default_bucket}/intermidiate`目录下，也可以按照注释代码自行指定

obs_bucket = session.obs.get_default_bucket()
print("Default bucket name: ", obs_bucket)
default_obs_dir = f"{obs_bucket}/intermidiate"
#default_obs_dir = "obs://your-bucket-name/folder-name"

# 本地的工程代码文件夹路径
code_dir_local = "/home/ma-user/work/models/official/cv/resnet/" #@param {type:"string"}

# 代码的启动文件名称
boot_file = "train.py" #@param {type:"string"}
train_file = TrainingFiles(code_dir=code_dir_local, boot_file=boot_file)

# 本地数据集路径
local_data_path = "/home/ma-user/work/models/dataset/flower_photos" #@param {type:"string"}

# 模型输出保存路径
output_local = "/home/ma-user/work/models/official/cv/resnet/output" #@param {type:"string"}
# 模拟训练过程中模型输出回传至指定OBS的路径，需要以"/"结尾
obs_output_path = f"{default_obs_dir}/mindspore_model/output/"

# 指定一个obs路径用于存储输出结果
output = [OutputData(local_path=output_local, obs_path=obs_output_path, name="output")]

# 模拟训练过程中模训练日志回传至指定OBS的路径，需要以"/"结尾
log_obs_path = f"{default_obs_dir}/mindspore_model/logs/"

# 训练所需的代码路径，代码会自动从本地上传至OBS
code_obs_path = f"{default_obs_dir}/mindspore_model/"
data_obs_path = f"{default_obs_dir}/dataset/flower_photos/"

# sdk会将代码自动上传至OBS，并同步到训练环境
train_file = TrainingFiles(code_dir=code_dir_local, boot_file=boot_file, obs_path=code_obs_path)

# 指定OBS中的数据集路径，会自动将local_path数据上传至obs_path，用户可以在代码中通过 --data_url接收这个数据集路径
input_data = InputData(local_path=local_data_path, obs_path=data_obs_path, is_local_source=True, name="data_url")


from modelarts.service import SWRManagement
image_organization = SWRManagement(session).get_default_namespace()
```

```
# image_organization = "your-swr-namespace-name"
print("Default image_organization:", image_organization)

image_name = "mindspore-image-models-image" #@param {type:"string"}
image_tag = "1.0.0" #@param {type:"string"}

import os
ENV_NAME=os.getenv('ENV_NAME')

# 启动训练任务：使用user_command（shell命令）方式启动训练任务
# 注意：训练启动默认的工作路径为"/home/ma-user/modelarts/user-job-dir"，而代码上传路径为"./resnet/${code_dir}"下
# --enable_modelarts=True 该代码仓已适配ModelArts
estimator = Estimator(session=session,
                      training_files=train_file,
                      outputs=output,
                      user_image_url=f"{image_organization}/{image_name}:{image_tag}", # 自定义镜像swr地址，由镜像仓库组织/镜像名称:镜像tag组成
                      user_command=f'cd /home/ma-user/modelarts/user-job-dir/ && /home/ma-user/anaconda3/envs/MindSpore/bin/python ./resnet/train.py --net_name=resnet50 --dataset=imagenet2012 --enable_modelarts=True --class_num=5 --config_path=./resnet/config/resnet50_imagenet2012_config.yaml --epoch_size=10 --device_target="Ascend" --enable_modelarts=True', # 执行训练命令
                      train_instance_type="modelarts.p3.large.public", # 虚拟资源规格，不同region的资源规格可能不同，请参考“Estimator参数说明”表下的说明查询修改
                      train_instance_count=1, # 节点数，适用于多机分布式训练，默认是1
                      #pool_id='若指定专属池，替换为页面上查到的poolId'，同时修改资源规格为专属池专用的虚拟子规格
                      log_url=log_obs_path
                     )
# job_name是可选参数，可不填随机生成工作名
job_instance = estimator.fit(inputs=[input_data],
                             job_name="modelarts_training_job_with_sdk_by_command_v01")
```

表 9-2 Estimator 参数说明

参数名称	参数说明
session	modelarts session
training_files	训练代码的路径和启动文件
user_image_url	自定义镜像swr地址，由镜像仓库组织/镜像名称:镜像tag组成
user_command	执行训练命令
train_instance_type	本地调测'local'或云端资源规格。每个region的资源规格可能是不同的，可以通过下述说明查询对应的资源规格信息。
train_instance_count	节点数
log_url	日志输出路径
job_name	作业名称，不可以重复

说明

train_instance_type表示训练的资源规格，每个region的资源规格可能是不同的。通过如下方法查询资源规格：

- 公共资源池执行如下命令查询

```
from modelarts.session import Session
from modelarts.estimatorV2 import Estimator
from pprint import pprint

# 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
# 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
__AK = os.environ["HUAWEICLOUD_SDK_AK"]
__SK = os.environ["HUAWEICLOUD_SDK_SK"]
# 如果进行了加密还需要进行解密操作
session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')

info = Estimator.get_train_instance_types(session=session)
pprint(info)
```

- 专属池规格

  ModelArts专属资源池统一使用虚拟子规格，不区分GPU和Ascend。资源规格参考**表9-3**查询。

表 9-3 专属资源池虚拟规格的说明

train_instance_type	说明
modelarts.pool.visual.xlarge	1卡
modelarts.pool.visual.2xlarge	2卡
modelarts.pool.visual.4xlarge	4卡
modelarts.pool.visual.8xlarge	8卡

步骤 6：清除资源

Notebook在代码调试完成及提交训练作业后就可以关闭了，减少资源扣费。

当调测完成且实例处于运行状态时，单击停止；

当下次调测且实例处于停止状态时，单击启动实例，随开随用。

训练输出保存结构说明

ModelArts训练作业的模型输出和日志信息会定时同步到指定的OBS中，本示例中模型输出路径和日志输出路径分别为f"{default_obs_dir}/mindspore_model/output/"和f"{default_obs_dir}/mindspore_model/logs/"，用户可以在OBS中查看训练输出信息。

本示例中训练输出保存在OBS的目录结构如下所示：

```
${your_bucket}
└── intermidiate
├── dataset
```

```
│   └── flower_photos
│   └── flower_photos.zip
└── mindspore_model
├── logs
│   └── xxx-xxx-xxx--0.log
├── output
│   └── 20220627-105226-resnet50-224
└── mindspore-image-models.zip
```

提交训练作业常见问题

- **报错信息：Exception: You have attempted to create more buckets than allowed**

  原因分析：由于桶的数量多于限额，无法自动创建。

  解决方法：用户可以删除一个桶，或者直接指定一个已存在的桶（修改变量obs_bucket的值）。

- **报错信息："errorMessage":"The number of namespaces exceeds the upper limit"或"namespace is invalid"**

  原因分析：SWR组织数限额，SWR组织默认最多只能创建5个组织。

  解决方法：用户可以删除一个SWR组织，或者直接指定一个已存在的SWR组织（修改变量image_organization的值）。

- **报错信息：standard_init_linux.go:224: exec user process caused "exet format error"**

  原因分析：可能由于训练规格错误导致训练作业卡死。

  解决方法：请参考**说明**查询资源规格。

- **报错信息：报错镜像失败，报错：401，'Unauthorized'，b'{errors": [{"errorCode":"SVCSTG.SWR.4010000",errorMessage":"Authenticate Error",……}]**

  原因分析：远程连接Notebook时需要输入鉴权信息。

  解决方法：传入AK，SK信息。

  ```
  # 认证用的ak和sk硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
  # 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
  __AK = os.environ["HUAWEICLOUD_SDK_AK"]
  __SK = os.environ["HUAWEICLOUD_SDK_SK"]
  # 如果进行了加密还需要进行解密操作
  session = Session(access_key=__AK,secret_key=__SK, project_id='***', region_name='***')
  ```

10 Standard 模型训练

10.1 使用 ModelArts Standard 自定义算法实现手写数字识别

本文为用户提供如何将本地的自定义算法通过简单的代码适配，实现在ModelArts上进行模型训练与部署的全流程指导。

场景描述

本案例用于指导用户使用PyTorch1.8实现手写数字图像识别，示例采用的数据集为MNIST官方数据集。

通过学习本案例，您可以了解如何在ModelArts平台上训练作业、部署推理模型并预测的完整流程。

操作流程

开始使用如下样例前，请务必按**准备工作**指导完成必要操作。

1. **Step1 准备训练数据**：下载MNIST数据集。
2. **Step2 准备训练文件和推理文件**：编写训练与推理代码。
3. **Step3 创建OBS桶并上传文件**：创建OBS桶和文件夹，并将数据集和训练脚本，推理脚本，推理配置文件上传到OBS中。
4. **Step4 创建训练作业**：进行模型训练。
5. **Step5 推理部署**：训练结束后，将生成的模型导入ModelArts用于创建AI应用，并将AI应用部署为在线服务。
6. **Step6 预测结果**：上传一张手写数字图片，发起预测请求获取预测结果。
7. **Step7 清除资源**：运行完成后，停止服务并删除OBS中的数据，避免不必要的扣费。

准备工作

- 已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
- 配置委托访问授权

ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

b. 在弹出的“访问授权”窗口中，

授权对象类型：所有用户

委托选择：新增委托

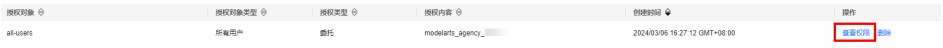
权限配置：普通用户

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

图 10-1 配置委托访问授权

c. 完成配置后，在ModelArts控制台的全局配置列表，可查看到此账号的委托配置信息。

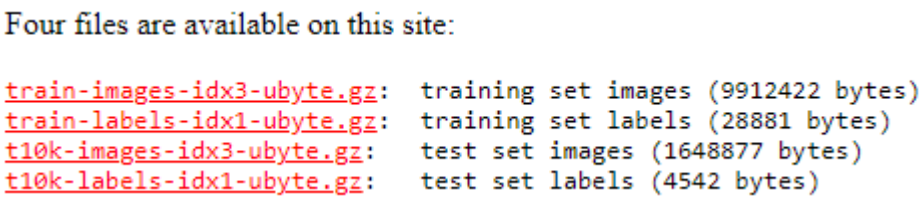
图 10-2 查看委托配置信息



Step1 准备训练数据

本案例使用的数据是MNIST数据集，您可以在浏览器中搜索“MNIST数据集”下载如**图10-3**所示的4个文件。

图 10-3 MNIST 数据集



- “train-images-idx3-ubyte.gz”：训练集的压缩包文件，共包含60000个样本。
- “train-labels-idx1-ubyte.gz”：训练集标签的压缩包文件，共包含60000个样本的类别标签。
- “t10k-images-idx3-ubyte.gz”：验证集的压缩包文件，共包含10000个样本。

- “t10k-labels-idx1-ubyte.gz”：验证集标签的压缩包文件，共包含10000个样本的类别标签。

Step2 准备训练文件和推理文件

针对此案例，ModelArts提供了需使用的训练脚本、推理脚本和推理配置文件。请参考如下文件内容。

说明

粘贴“.py”文件代码时，请直接新建“.py”文件，否则会可能出现“SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence”报错。

粘贴完代码后，建议检查代码文件是否出现中文注释变为乱码的情况，如果出现该情况请将编辑器改为utf-8格式后再粘贴代码。

在本地电脑中创建训练脚本“train.py”，内容如下：

```
# base on https://github.com/pytorch/examples/blob/main/mnist/main.py

from __future__ import print_function

import os
import gzip
import codecs
import argparse
from typing import IO, Union

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR

import shutil


# 定义网络模型
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
# 模型训练，设置模型为训练模式，加载训练数据，计算损失函数，执行梯度下降
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            if args.dry_run:
                break


# 模型验证，设置模型为验证模式，加载验证数据，计算损失函数和准确率
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))


# 以下为pytorch mnist
# https://github.com/pytorch/vision/blob/v0.9.0/torchvision/datasets/mnist.py
def get_int(b: bytes) -> int:
    return int(codecs.encode(b, 'hex'), 16)


def open_maybe_compressed_file(path: Union[str, IO]) -> Union[IO, gzip.GzipFile]:
    """Return a file object that possibly decompresses 'path' on the fly.
       Decompression occurs when argument `path` is a string and ends with '.gz' or '.xz'.
    """
    if not isinstance(path, torch._six.string_classes):
        return path
    if path.endswith('.gz'):
        return gzip.open(path, 'rb')
    if path.endswith('.xz'):
        return lzma.open(path, 'rb')
    return open(path, 'rb')


SN3_PASCALVINCENT_TYPEMAP = {
    8: (torch.uint8, np.uint8, np.uint8),
    9: (torch.int8, np.int8, np.int8),
    11: (torch.int16, np.dtype('>i2'), 'i2'),
    12: (torch.int32, np.dtype('>i4'), 'i4'),
    13: (torch.float32, np.dtype('>f4'), 'f4'),
    14: (torch.float64, np.dtype('>f8'), 'f8')
}


def read_sn3_pascalvincent_tensor(path: Union[str, IO], strict: bool = True) -> torch.Tensor:
    """Read a SN3 file in "Pascal Vincent" format (Lush file 'libidx/idx-io.lsh').
```

```
    Argument may be a filename, compressed filename, or file object.
    """
    # read
    with open_maybe_compressed_file(path) as f:
        data = f.read()
    # parse
    magic = get_int(data[0:4])
    nd = magic % 256
    ty = magic // 256
    assert 1 <= nd <= 3
    assert 8 <= ty <= 14
    m = SN3_PASCALVINCENT_TYPEMAP[ty]
    s = [get_int(data[4 * (i + 1): 4 * (i + 2)]) for i in range(nd)]
    parsed = np.frombuffer(data, dtype=m[1], offset=(4 * (nd + 1)))
    assert parsed.shape[0] == np.prod(s) or not strict
    return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)


def read_label_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 1)
    return x.long()


def read_image_file(path: str) -> torch.Tensor:
    with open(path, 'rb') as f:
        x = read_sn3_pascalvincent_tensor(f, strict=False)
    assert(x.dtype == torch.uint8)
    assert(x.ndimension() == 3)
    return x


def extract_archive(from_path, to_path):
    to_path = os.path.join(to_path, os.path.splitext(os.path.basename(from_path))[0])
    with open(to_path, "wb") as out_f, gzip.GzipFile(from_path) as zip_f:
        out_f.write(zip_f.read())
# --- 以上为pytorch mnist
# --- end


# raw mnist 数据处理
def convert_raw_mnist_dataset_to_pytorch_mnist_dataset(data_url):
    """
    raw

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz

    processed

    {data_url}/
        train-images-idx3-ubyte.gz
        train-labels-idx1-ubyte.gz
        t10k-images-idx3-ubyte.gz
        t10k-labels-idx1-ubyte.gz
        MNIST/raw
            train-images-idx3-ubyte
            train-labels-idx1-ubyte
            t10k-images-idx3-ubyte
            t10k-labels-idx1-ubyte
        MNIST/processed
            training.pt
            test.pt
    """
```

```
    resources = [
        "train-images-idx3-ubyte.gz",
        "train-labels-idx1-ubyte.gz",
        "t10k-images-idx3-ubyte.gz",
        "t10k-labels-idx1-ubyte.gz"
    ]

    pytorch_mnist_dataset = os.path.join(data_url, 'MNIST')

    raw_folder = os.path.join(pytorch_mnist_dataset, 'raw')
    processed_folder = os.path.join(pytorch_mnist_dataset, 'processed')

    os.makedirs(raw_folder, exist_ok=True)
    os.makedirs(processed_folder, exist_ok=True)

    print('Processing...')

    for f in resources:
        extract_archive(os.path.join(data_url, f), raw_folder)

    training_set = (
        read_image_file(os.path.join(raw_folder, 'train-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 'train-labels-idx1-ubyte'))
    )
    test_set = (
        read_image_file(os.path.join(raw_folder, 't10k-images-idx3-ubyte')),
        read_label_file(os.path.join(raw_folder, 't10k-labels-idx1-ubyte'))
    )
    with open(os.path.join(processed_folder, 'training.pt'), 'wb') as f:
        torch.save(training_set, f)
    with open(os.path.join(processed_folder, 'test.pt'), 'wb') as f:
        torch.save(test_set, f)

    print('Done!')


def main():
    # 定义可以接收的训练作业运行参数
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')

    parser.add_argument('--data_url', type=str, default=False,
                        help='mnist dataset path')
    parser.add_argument('--train_url', type=str, default=False,
                        help='mnist model path')

    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=14, metavar='N',
                        help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1.0, metavar='LR',
                        help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, metavar='M',
                        help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--dry-run', action='store_true', default=False,
                        help='quickly check a single pass')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True,
                        help='For Saving the current Model')
    args = parser.parse_args()

    use_cuda = not args.no_cuda and torch.cuda.is_available()
```

```
    torch.manual_seed(args.seed)

    # 设置使用 GPU 还是 CPU 来运行算法
    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': args.batch_size}
    test_kwargs = {'batch_size': args.test_batch_size}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': True}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    # 定义数据预处理方法
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # 将 raw mnist 数据集转换为 pytorch mnist 数据集
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # 分别创建训练和验证数据集
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)

    # 分别构建训练和验证数据迭代器
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # 初始化神经网络模型并复制模型到计算设备上
    model = Net().to(device)
    # 定义训练优化器和学习率策略，用于梯度下降计算
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # 训练神经网络，每一轮进行一次验证
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # 保存模型与适配 ModelArts 推理模型包规范
    if args.save_model:

        # 在 train_url 训练参数对应的路径内创建 model 目录
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # 按 ModelArts 推理模型包规范，保存模型到 model 目录内
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # 复制推理代码与配置文件到 model 目录内
        the_path_of_current_file = os.path.dirname(__file__)
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/customize_service.py'),
os.path.join(model_path, 'customize_service.py'))
        shutil.copyfile(os.path.join(the_path_of_current_file, 'infer/config.json'), os.path.join(model_path,
'config.json'))

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

在本地电脑中创建推理脚本“customize_service.py”，内容如下：

```
import os
import log
import json
```

```
import torch.nn.functional as F
import torch.nn as nn
import torch
import torchvision.transforms as transforms

import numpy as np
from PIL import Image

from model_service.pytorch_model_service import PTServingBaseService

logger = log.getLogger(__name__)

# 定义模型预处理
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 模型推理服务
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # 调用父类构造方法
        super(PTVisionService, self).__init__(model_name, model_path)

        # 调用自定义函数加载模型
        self.model = Mnist(model_path)

        # 加载标签
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # 接收request数据，并转换为模型可以接受的输入格式
    def _preprocess(self, data):
        preprocessed_data = {}
        for k, v in data.items():
            input_batch = []
            for file_name, file_content in v.items():
                with Image.open(file_content) as image1:
                    # 灰度处理
                    image1 = image1.convert("L")
                    if torch.cuda.is_available():
                        input_batch.append(infer_transformation(image1).cuda())
                    else:
                        input_batch.append(infer_transformation(image1))
            input_batch_var = torch.autograd.Variable(torch.stack(input_batch, dim=0), volatile=True)
            print(input_batch_var.shape)
            preprocessed_data[k] = input_batch_var

        return preprocessed_data

    # 将推理的结果进行后处理，得到预期的输出格式，该结果就是最终的返回值
    def _postprocess(self, data):
        results = []
        for k, v in data.items():
            result = torch.argmax(v[0])
            result = {k: self.label[result]}
            results.append(result)
        return results

    # 对于输入数据进行前向推理，得到推理结果
    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)
```

```
        return result

# 定义网络
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


def Mnist(model_path, **kwargs):
    # 生成网络
    model = Net()

    # 加载模型
    if torch.cuda.is_available():
        device = torch.device('cuda')
        model.load_state_dict(torch.load(model_path, map_location="cuda:0"))
    else:
        device = torch.device('cpu')
        model.load_state_dict(torch.load(model_path, map_location=device))

    # CPU 或者 GPU 映射
    model.to(device)

    # 声明为推理模式
    model.eval()

    return model
```

在本地电脑中推理配置文件“config.json”，内容如下：

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

Step3 创建 OBS 桶并上传文件

将上一步中的数据和代码文件、推理代码文件与推理配置文件，从本地上传到OBS桶中。在ModelArts上运行训练作业时，需要从OBS桶中读取数据和代码文件。

1. 登录OBS管理控制台，按照如下示例创建OBS桶和文件夹。创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

   ```
   {OBS桶}                 # OBS对象桶，用户可以自定义名称，例如：test-modelarts-xx
       -{OBS文件夹}          # OBS文件夹，自定义名称，此处举例为pytorch
           - mnist-data     # OBS文件夹，用于存放训练数据集，可以自定义名称，此处举例为mnist-data
           - mnist-code     # OBS文件夹，用于存放训练脚本train.py，可以自定义名称，此处举例为mnist-
   ```

```
code
        - infer       # OBS文件夹，用于存放推理脚本customize_service.py和配置文件config.json
      - mnist-output    # OBS文件夹，用于存放训练输出模型，可以自定义名称，此处举例为mnist-
output
```

⚠ 注意

- 创建的OBS桶所在区域和后续使用ModelArts必须在同一个区域Region，否则会导致训练时找不到OBS桶。具体操作可参见**查看OBS桶与ModelArts是否在同一区域**。
- 创建OBS桶时，桶的存储类别请勿选择“归档存储”，归档存储的OBS桶会导致模型训练失败。

2. 上传**Step1 准备训练数据**下载的MNIST数据集压缩包文件到OBS的“mnist-data”文件夹中。上传文件至OBS的操作指导请参见**上传对象**。

⚠ 注意

- 上传数据到OBS中时，请不要加密，否则会导致训练失败。
- 文件无需解压，直接上传压缩包至OBS中即可。

3. 上传训练脚本“train.py”到“mnist-code”文件夹中。
4. 上传推理脚本“customize_service.py”和推理配置文件“config.json”到“mnist-code”的“infer”文件中。

Step4 创建训练作业

1. 登录ModelArts管理控制台，选择和OBS桶相同的区域。
2. 在“权限管理”中检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
3. 在左侧导航栏选择“模型训练 > 训练作业”进入训练作业页面，单击“创建训练作业”。
4. 填写创建训练作业相关信息。
   - “创建方式”：选择“自定义算法”。
   - “启动方式”：选择“预置框架”，下拉框中选择PyTorch，pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64。
   - “代码目录”：选择已创建的OBS代码目录路径，例如“/test-modelarts-xx/pytorch/mnist-code/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “启动文件”：选择代码目录下上传的训练脚本“train.py”。
   - “输入”：单击“增加训练输入”，设置训练输入的“参数名称”为“data_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-data/”（test-modelarts-xx需替换为您的OBS桶名称）。
   - “输出”：单击“增加训练输出”，设置训练输出的“参数名称”为“train_url”。设置数据存储位置为您的OBS目录，例如“/test-modelarts-xx/pytorch/mnist-output/”（test-modelarts-xx需替换为您的OBS桶名称）。预下载至本地目录选择“不下载”。

- “资源类型”：选择GPU单卡的规格。如果有免费GPU规格，可以选择免费规格进行训练。
- 其他参数保持默认即可。

说明

本样例代码为单机单卡场景，选择GPU多卡规格会导致训练失败。

5. 单击“提交”，确认训练作业的参数信息，确认无误后单击“确定”。
   页面自动返回“训练作业”列表页，当训练作业状态变为“已完成”时，即完成了模型训练过程。

说明

本案例的训练作业预计运行十分钟。

6. 单击训练作业名称，进入作业详情界面查看训练作业日志信息，观察日志是否有明显的Error信息，如果有则表示训练失败，请根据日志提示定位原因并解决。
7. 在训练详情页左下方单击训练输出路径，如图10-4所示，跳转到OBS目录，查看是否存在model文件夹，且model文件夹中是否有生成训练模型。如果未生成model文件夹或者训练模型，可能是训练输入数据不完整导致，请检查训练数据上传是否完整，并重新训练。

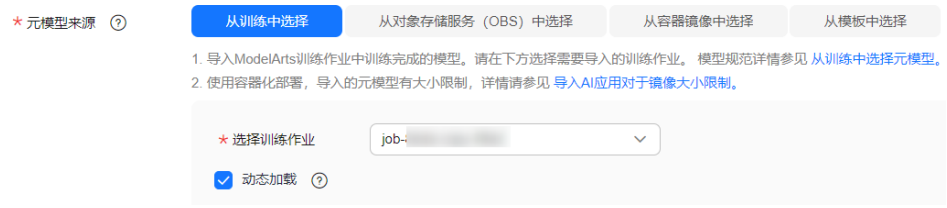
图 10-4 训练输出路径

Step5 推理部署

模型训练完成后，可以创建AI应用，将AI应用部署为在线服务。

1. 在ModelArts管理控制台，单击左侧导航栏中的“AI应用”，进入“自定义应用”页面，单击“创建应用”。
2. 在“创建应用”页面，填写相关参数，然后单击“立即创建”。
   在“元模型来源”中，选择“从训练中选择”页签，选择**Step4 创建训练作业**中完成的训练作业，勾选“动态加载”。AI引擎的值是系统自动写入的，无需设置。

图 10-5 设置元模型来源



3. 在AI应用列表页面，当AI应用状态变为“正常”时，表示AI应用创建成功。单击AI应用操作列的“部署”，弹出“版本列表”，单击操作列“部署>在线服务”，将AI应用部署为在线服务。

图 10-6 部署在线服务

4. 在“部署”页面，参考下图填写参数，然后根据界面提示完成在线服务创建。本案例适用于CPU规格，节点规格需选择CPU。如果有免费CPU规格，可选择免费规格进行部署（每名用户限部署一个免费的在线服务，如果您已经部署了一个免费在线服务，需要先将其删除才能部署新的免费在线服务）。

图 10-7 部署模型

完成服务部署后，返回在线服务页面列表页，等待服务部署完成，当服务状态显示为“运行中”，表示服务已部署成功。

Step6 预测结果

1. 在“在线服务”页面，单击在线服务名称，进入服务详情页面。
2. 单击“预测”页签，请求类型选择“multipart/form-data”，请求参数填写“image”，单击“上传”按钮上传示例图片，然后单击“预测”。

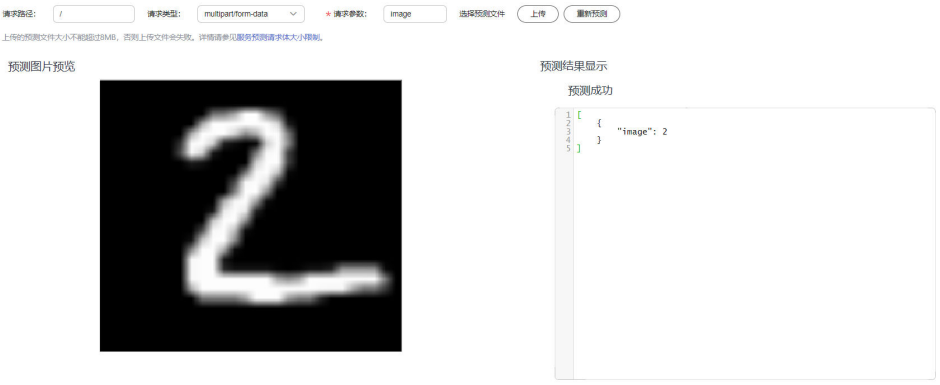
   预测完成后，预测结果显示区域将展示预测结果，根据预测结果内容，可识别出此图片的数字是“2”。

说明

本案例中使用的MNIST是比较简单的用做demo的数据集，配套算法也是比较简单的用于教学的神经网络算法。这样的数据和算法生成的模型仅适用于教学模式，并不能应对复杂的预测场景。即生成的模型对预测图片有一定范围和要求，预测图片必须和训练集中的图片相似（黑底白字）才可能预测准确。

图 10-8 示例图片

图 10-9 预测结果展示



Step7 清除资源

如果不再需要使用此模型及在线服务，建议清除相关资源，避免产生不必要的费用。

- 在“在线服务”页面，“停止”或“删除”刚创建的在线服务。
- 在“AI应用”页面，“删除”刚创建的AI应用。
- 在“训练作业”页面，“删除”运行结束的训练作业。
- 进入OBS，删除本示例使用的OBS桶及文件夹，以及文件夹的文件。

常见问题

- 训练作业一直在等待中（排队）？

  训练作业状态一直在等待中状态表示当前所选的资源池规格资源紧张，作业需要进行排队，请耐心等待。请参考**训练作业一直在等待中（排队）？** 。
- 在ModelArts中选择OBS路径时，找不到已创建的OBS桶?

  请确保创建的桶和ModelArts服务在同一区域，详细操作请参考**查看OBS桶与ModelArts是否在同一个区域**。

10.2 Standard 专属资源池训练

10.2.1 资源选择推荐

不同AI模型训练所需要的数据量和算力不同，在训练时选择合适存储及训练方案可提升模型训练效率与资源性价比。ModelArts支持单机单卡、单机多卡和多机多卡的训练场景，满足不同AI模型训练的要求。针对第一次使用ModelArts的用户，本文提供端到端案例指导，帮助您快速了解如何在ModelArts上选择合适的训练方案并进行模型训练。

针对不同的数据量和算法情况，推荐以下训练方案：

- 单机单卡：小数据量（1G训练数据）、低算力场景（1卡Vnt1），存储方案使用“OBS的并行文件系统（存放数据和代码）”。

- 单机多卡：中等数据量（50G左右训练数据）、中等算力场景（8卡Vnt1），存储方案使用“SFS（存放数据和代码）”。
- 多机多卡：大数据量（1T训练数据）、高算力场景（4台8卡Vnt1），存储方案使用“SFS（存放数据）+普通OBS桶（存放代码）”，采用分布式训练。

表 10-1 不同场景所需服务及购买推荐

场景	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
单机单卡	按需购买。（并行文件系统）	×	免费。	免费。	包月购买。	免费。	×	按需购买。
单机多卡	×	包月购买。（HPC型 500G）	免费。	免费。	包月购买。	免费。	包月购买。（Ubuntu 18.04，建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×
多机多卡	按需购买。（普通OBS桶）	包月购买。（HPC型 500G）	免费。	免费。	包月购买。	免费。	包月购买。（建议不小于2U8G，本地存储空间100G，带EIP全动态BGP，按流量10M带宽）	×

表 10-2 开源数据集训练效率参考

算法及数据	资源规格	Epoch数	运行时长（hh:mm:ss）
算法：PyTorch官方针对ImageNet的样例 数据：ImageNet分类数据子集	1机1卡Vnt1	10	0:05:03
算法：YOLOX 数据：COCO2017	1机1卡Vnt1	10	03:33:13
	1机8卡Vnt1	10	01:11:48
	4机8卡Vnt1	10	0:36:17
算法：Swin-Transformer 数据：ImageNet21K	1机1卡Vnt1	10	197:25:03
	1机8卡Vnt1	10	26:10:25
	4机8卡Vnt1	10	07:08:44

表 10-3 训练各步骤性能参考

步骤	说明	时长
镜像下载	首次下载镜像的时间（25G）。	8分钟
资源调度	点创建训练任务开始到变成运行中的时间（资源充足、镜像已缓存）。	20秒
训练列表页打开	已有50条训练作业，单击训练模块后的时间。	6秒
日志加载	作业运行中，已经输出1兆的日志文本，单击训练详情页面需要多久加载出日志。	2.5秒
训练详情页	作业运行中，没有用户日志情况下，在ModelArts控制台主页面单击训练详情页面后加载页面内容。	2.5秒
JupyterLab页面	进入JupyterLab页面后加载页面内容。	0.5秒
Notebook列表页	已有50个Notebook实例，在ModelArts控制台主页面单击开发环境后的时间。	4.5秒

说明

镜像下载时间受节点规格、节点硬盘类型（高IO/普通IO）、是否SSD等因素影响，以上数据仅供参考。

10.2.2 步骤总览

单机单卡

1. 资源购买：
   a. **购买对象存储服务OBS**
   b. **购买容器镜像服务SWR**
   c. **创建网络**
   d. **购买ModelArts专属资源池**
2. 基本配置：
   a. **权限配置**
   b. **obsutils安装和配置**
   c. （可选）**工作空间配置**
3. 训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据和算法至OBS（首次使用时需要）**
   d. **使用Notebook进行代码调试**
   e. **创建训练任务**

单机多卡

1. 资源购买：
   a. **购买虚拟私有云VPC**
   b. **购买弹性文件服务SFS**
   c. **购买容器镜像服务SWR**
   d. **创建网络**
   e. **购买ModelArts专属资源池**
   f. **购买弹性云服务器ECS**
2. 基本配置：
   a. **权限配置**
   b. **专属资源池VPC打通**
   c. **ECS服务器挂载SFS Turbo存储**
   d. （可选）**工作空间配置**
3. 训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据和算法至SFS（首次使用时需要）**
   d. **使用Notebook进行代码调试**
   e. **创建训练任务**

多机多卡

1. 资源购买：
   a. **购买虚拟私有云VPC**
   b. **购买弹性文件服务SFS**
   c. **购买对象存储服务OBS**
   d. **购买容器镜像服务SWR**
   e. **创建网络**
   f. **购买ModelArts专属资源池**
   g. **购买弹性云服务器ECS**
2. 基本配置：
   a. **权限配置**
   b. **专属资源池VPC打通**
   c. **ECS服务器挂载SFS Turbo存储**
   d. **在ECS中创建ma-user和ma-group**
   e. **obsutils安装和配置**
   f. （可选）**工作空间配置**
3. 训练：
   a. **线下容器镜像构建及调试**
   b. **上传镜像**
   c. **上传数据至OBS（首次使用时需要）**
   d. **上传算法至SFS**
   e. **使用Notebook进行代码调试**
   f. **创建训练任务**

10.2.3 资源购买

购买弹性文件服务 SFS

弹性文件服务默认为按需计费，即按购买的存储容量和时长收费。您也可以购买包年包月套餐，提前规划资源的使用额度和时长。在欠费时，您需要及时（15天之内）续费以避免您的文件系统资源被清空。SFS购买指导请参考**如何购买弹性文件服务？**。

购买容器镜像服务 SWR

容器镜像服务分为企业版和共享版。

共享版计费项包括存储空间和流量费用，目前均免费提供给您。

企业版当前仅支持按需计费模式，公测期间，可免费使用。

说明

上传镜像前需要创建组织，创建步骤请参考**创建组织**。

购买对象存储服务 OBS

对象存储服务提供按需计费和包年包月两种计费模式，用户可以根据实际需求购买OBS服务。OBS服务支持以下两种存储方式，单机单卡场景使用文件系统，多机多卡场景使用普通OBS桶。

- **创建普通OBS桶**
- **创建并行文件系统**

购买数据加密服务 DEW

在使用Notebook进行代码调试时，如果要开启“SSH远程开发”功能，需要选择已有密钥对。密钥对可免费创建，您可通过管理控制台创建密钥对，操作指导请参考**如何创建密钥对?**

购买虚拟私有云 VPC

虚拟私有云可以为您构建隔离的、用户自主配置和管理的虚拟网络环境，操作指导请参考**创建虚拟私有云和子网**。

购买弹性云服务器 ECS

如果您需要在服务器上部署相关业务，较之物理服务器，弹性云服务器的创建成本较低，并且可以在几分钟之内快速获得基于云服务平台的弹性云服务器设施，并且这些基础设施是弹性的，可以根据需求伸缩。下面介绍如何在管理控制台购买弹性云服务器。

购买流程：

- **步骤一：基础配置**
- **步骤二：网络配置**
- **步骤三：高级配置**
- **步骤四：确认订单**

说明

购买时需注意，ECS需要和SFS买到同一个VPC才能挂载SFS存储。

购买 ModelArts 专属资源池

提供独享的计算资源，可用于Notebook、训练作业、部署模型。专属资源池不与其他用户共享，更加高效。在使用专属资源池之前，您需要先创建一个专属资源池，操作指导请参考**创建专属资源池**。

说明

创建一个专属资源池前需要先创建网络，创建网络指导可参考**创建网络**。

购买 Notebook 存储

使用Notebook代码调试时，需要创建Notebook实例，如果创建时选择“云硬盘EVS”作为存储位置，会创建云硬盘EVS。

磁盘规格默认5GB，从Notebook实例创建成功开始，直至实例删除成功，磁盘每GB按照规定费用收费。

说明

云硬盘EVS会在创建Notebook实例时自动购买，无需用户单独创建。

10.2.4 基本配置

10.2.4.1 权限配置

权限列表

为了便于理解权限相关内容，建议先阅读**ModelArts权限管理基本概念**。

表 10-4 服务授权列表

待授权的服务	适用场景
ModelArts	授予子用户使用ModelArts服务的权限。 ModelArts CommonOperations没有任何专属资源池的创建、更新、删除权限，只有使用权限。推荐给子用户配置此权限。
	如果需要给子用户开通专属资源池的创建、更新、删除权限，此处要勾选ModelArts FullAccess，请谨慎配置。 ModelArts FullAccess权限和ModelArts CommonOperations权限只能二选一，不能同时选。
SFS弹性文件服务	弹性文件服务SFS Turbo的所有权限。使用SFS服务时需要配置。
ECS弹性云服务器	弹性云服务器所有权限。使用ECS服务时需要配置。
SWR容器镜像仓库	容器镜像仓库所有权限。使用SWR服务时需要配置。同时，还需开通SWR组织权限。
VPC虚拟私有云	子用户在创建ModelArts的专属资源池过程中，如果需要开启自定义网络配置，需要配置VPC权限。
DEW密钥管理服务	当子用户使用ModelArts Notebook的SSH远程功能时，需要配置子用户密钥管理服务的使用权限。
OBS对象存储服务	具有对象存储服务（OBS）查看桶列表、获取桶元数据、列举桶内对象、查询桶位置、上传对象、获取对象、删除对象、获取对象ACL等对象基本操作权限。

10.2.4.1.1 配置 IAM 权限

1. 使用华为云主帐号创建一个开发者用户组user_group，将开发者帐号加入用户组user_group中。具体操作请参见**Step1 创建用户组并加入用户**。
2. 创建自定义策略。
   a. 使用华为云主帐号登录控制台，单击右上角用户名，在下拉框中选择“统一身份认证”，进入IAM服务。

b. 在统一身份认证服务控制台的左侧菜单栏中，选择“权限管理> 权限”。单击右上角“创建自定义策略”，“策略名称”为“Policy1”，策略配置方式选择JSON视图，输入策略内容，单击“确定”。

c. 自定义策略“Policy1”的具体内容如下，可以直接复制粘贴。

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Action": [
            "modelarts:*:*"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
            "modelarts:pool:create",
            "modelarts:pool:update",
            "modelarts:pool:delete"
         ],
         "Effect": "Deny"
      },
      {
         "Action": [
            "sfsturbo:*:*",
            "vpc:*:*",
            "dss:*:get",
            "dss:*:list"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
            "ecs:*:*",
            "evs:*:get",
            "evs:*:list",
            "evs:volumes:create",
            "evs:volumes:delete",
            "evs:volumes:attach",
            "evs:volumes:detach",
            "evs:volumes:manage",
            "evs:volumes:update",
            "evs:volumes:use",
            "evs:volumes:uploadImage",
            "evs:snapshots:create",
            "vpc:*:get",
            "vpc:*:list",
            "vpc:networks:create",
            "vpc:networks:update",
            "vpc:subnets:update",
            "vpc:subnets:create",
            "vpc:ports:*",
            "vpc:routers:get",
            "vpc:routers:update",
            "vpc:securityGroups:*",
            "vpc:securityGroupRules:*",
            "vpc:floatingIps:*",
            "vpc:publicIps:*",
            "ims:images:create",
            "ims:images:delete",
            "ims:images:get",
            "ims:images:list",
            "ims:images:update",
            "ims:images:upload"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
```

```
                "vpc:*:*",
                "ecs:*:get*",
                "ecs:*:list*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:cmk:*",
                "kms:dek:*",
                "kms:grant:*",
                "kms:cmkTag:*",
                "kms:partition:*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

3. 自定义策略“Policy2”的具体内容如下，可以直接复制粘贴。

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "obs:bucket:ListAllMybuckets",
                "obs:bucket:HeadBucket",
                "obs:bucket:ListBucket",
                "obs:bucket:GetBucketLocation",
                "obs:object:GetObject",
                "obs:object:GetObjectVersion",
                "obs:object:PutObject",
                "obs:object:DeleteObject",
                "obs:object:DeleteObjectVersion",
                "obs:object:ListMultipartUploadParts",
                "obs:object:AbortMultipartUpload",
                "obs:object:GetObjectAcl",
                "obs:object:GetObjectVersionAcl"
            ],
            "Effect": "Allow"
        }
    ]
}
```

说明

创建自定义策略时，建议将项目级云服务和全局级云服务拆分为两条策略，便于授权时设置最小授权范围。此处的“Policy1”为项目级云服务、“Policy2”为全局级云服务。**了解更多**。

4. 将自定义策略授权给开发者用户组user_group。
   a. 在统一身份认证服务控制台的左侧菜单栏中，选择“用户组”。在用户组页面单击对应用户组名称user_group操作列的“授权”，勾选策略“Policy1”、“Policy2”、“SWR Admin”。单击“下一步”。

      说明

      SWR的权限有SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess。但SWR FullAccess、SWR OperateAccess、SWR ReadOnlyAccess仅限容器镜像服务企业版使用，目前企业版已暂停公测。非企业版用户暂不支持使用此权限。因此需要在此勾选“SWR Admin” 策略。

   b. 选择授权范围方案为“所有资源”，单击“确定”。

精细化授权管理

如果您需要进行精细的权限管理，可参考《 ModelArts API参考 》中的权限策略和授权项。

- **数据管理权限**
- **开发环境权限**
- **训练作业权限**
- **模型管理权限**
- **服务管理权限**
- **工作空间管理权限**

精细化授权案例可参考**管理员和开发者权限分离**。

10.2.4.1.2 配置 ModelArts 委托权限

给用户配置ModelArts委托授权，允许ModelArts服务在运行时访问OBS等依赖服务。

1. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。
2. 在弹出的“访问授权”窗口中，

   授权对象类型：所有用户

   委托选择：新增委托

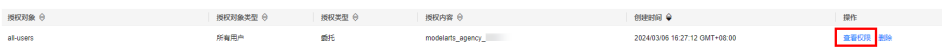
   权限配置：普通用户

   选择完成后勾选“我已经详细阅读并同意《 ModelArts服务声明 》”，然后单击“创建”。

   图 10-10 配置委托访问授权

3. 完成配置后，在ModelArts控制台的全局配置列表，可查看到此账号的委托配置信息。

   图 10-11 查看委托配置信息



10.2.4.1.3 配置 SWR 组织权限

IAM用户创建后，需要管理员在组织中为用户添加授权，使IAM用户对组织内所有镜像享有读取/编辑/管理的权限。

只有具备“管理”权限的帐号和IAM用户才能添加授权。

1. 登录容器镜像服务控制台。
2. 在左侧菜单栏选择“组织管理”，单击组织名称。
3. 在“用户”页签下单击“添加授权”，在弹出的窗口中为IAM用户选择权限，然后单击“确定”。

SWR授权管理详情可参考**授权管理**。

说明

如果给子用户的SWR授权不是SWR Admin权限，则需要继续配置SWR组织权限。

10.2.4.1.4 测试用户权限

由于权限配置需要等待15-30分钟生效，建议在配置完成后，等待30分钟，再执行如下验证操作。

1. 使用用户组02中任意一个子用户登录ModelArts管理控制台。在登录页面，请使用“IAM用户登录”方式进行登录。

   首次登录会提示修改密码，请根据界面提示进行修改。
2. 验证ModelArts权限。
   a. 在左上角的服务列表中，选择ModelArts服务，进入ModelArts管理控制台。
   b. 在ModelArts管理控制台，可正常创建Notebook、训练作业、注册镜像。
3. 验证SFS权限。
   a. 在左上角的服务列表中，选择SFS服务，进入SFS管理控制台。
   b. 在SFS管理控制台，在SFS Turo中单击右上角的“创建文件系统”，如果能正常打开页面，表示当前用户具备SFS的操作权限。
4. 验证ECS权限。
   a. 在左上角的服务列表中，选择ECS服务，进入ECS管理控制台。
   b. 在ECS管理控制台，单击右上角的“购买弹性云服务器”，如果能正常打开页面，表示当前用户具备ECS的操作权限。
5. 验证VPC权限。
   a. 在左上角的服务列表中，选择VPC服务，进入VPC管理控制台。
   b. 在VPC管理控制台，单击右上角的“创建虚拟私有云”，如果能正常打开页面，表示当前用户具备VPC的操作权限。
6. 验证DEW权限。
   a. 在左上角的服务列表中，选择DEW服务，进入DEW管理控制台。
   b. 在DEW管理控制台，在“密钥对管理”-“私有密钥对”中单击“创建密钥对”，如果能正常打开页面，表示当前用户具备DEW的操作权限。
7. 验证OBS权限。
   a. 在左上角的服务列表中，选择OBS服务，进入OBS管理控制台。
   b. 在OBS管理控制台，单击右上角的“创建桶”，如果能正常打开页面，表示当前用户具备OBS的操作权限。
8. 验证SWR权限。
   a. 在左上角的服务列表中，选择SWR服务，进入SWR管理控制台。

b. 在SWR管理控制台，如果能正常打开页面，表示当前用户具备SWR的操作权限。

c. 单击右上角的“上传镜像”，如果能看到授权的组织，表示当前用户具备SWR组织权限。

10.2.4.2 创建网络

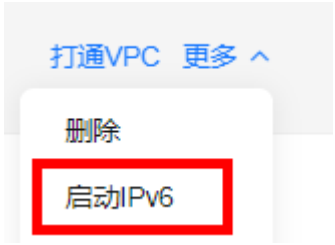
1. 登录ModelArts管理控制台，在左侧导航栏中选择“AI专属资源池 > 弹性集群 Cluster”，进入“弹性集群 Cluster”页面。
2. 切换到“网络”页签，单击“创建”，弹出“创建网络”页面。

图 10-12 网络列表

3. 在“创建网络”弹窗中填写网络信息。
   - 网络名称：创建网络时默认生成网络名称，也可自行修改。
   - 网段类型：可选“预置”和“自定义”。自定义网络目前支持网段范围：10.0.0.0/8~26、172.16.0.0/12~26、192.168.0.0/16~26。
   - IPV6：开启IPv6功能后，将自动为子网分配IPv6网段，暂不支持自定义设置IPv6网段，该功能一旦开启，将不能关闭。
     - 若创建网络时未勾选开启IPv6，也可在创建网络后在操作列单击“启动IPv6”，如图10-14
     - https://support.huaweicloud.com/usermanual-standard-modelarts/resmgmt-modelarts_0012.html#section2前，需要保证ModelArts网络和您的VPC网络都已开启IPv6，IPv6才会生效。若是打通VPC后，才开启ModelArts网络的IPv6或VPC网络的IPv6，此时需要重新打通VPC及子网，IPv6才会生效。

图 10-13 创建网络

图 10-14 启动 IPv6



说明

- 单用户最多可创建15个网络。
- 网段设置以后不能修改，避免与将要打通的VPC网段冲突。可能冲突的网段包括：
  - 用户的vpc网段
  - 容器网段（固定是172.16.0.0/16）
  - 服务网段（固定是10.247.0.0/16）

4. 确认无误后，单击“确定”。

10.2.4.3 专属资源池 VPC 打通

通过打通VPC，可以方便用户跨VPC使用资源，提升资源利用率。

1. 在“网络”页签，单击网络列表中某个网络操作列的“打通VPC”。

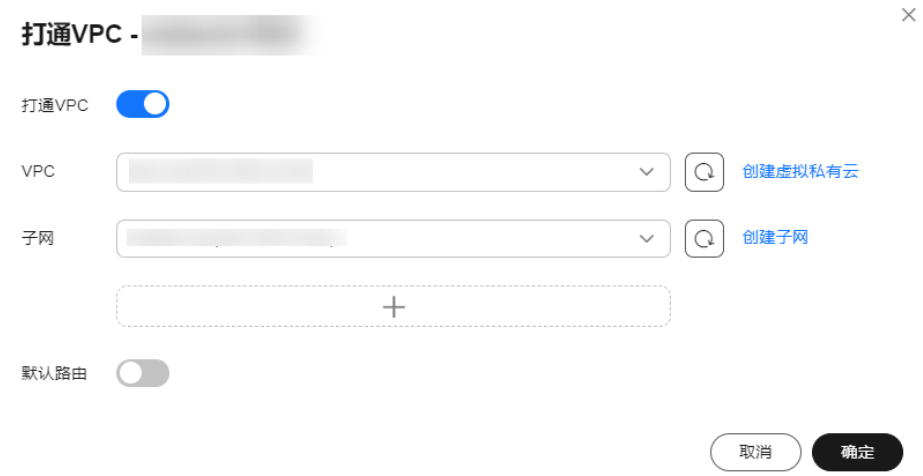
图 10-15 打通 VPC

2. 在打通VPC弹框中，打开“打通VPC”开关，在下拉框中选择可用的VPC和子网。

说明

需要打通的对端网络不能和当前网段重叠。

图 10-16 打通 VPC 参数选择



- 如果没有VPC可选，可以单击右侧的“创建虚拟私有云”，跳转到网络控制台，申请创建虚拟私有云。
- 如果没有子网可选，可以单击右侧的“创建子网”，跳转到网络控制台，创建可用的子网。
- 支持1个VPC下多个子网的打通，若VPC下有多个子网，会显示“+”，您可单击“+”即可添加子网（上限10个）。
- 若需要使用打通VPC的方式实现专属资源池访问公网，由于要访问的公网地址不确定，一般是建议用户在VPC中创建SNAT。此场景下，在打通VPC后，专属资源池中作业访问公网地址，默认不能转发到用户VPC的SNAT，需要提交工单联系技术支持在专属资源池VPC的路由中添加指向对等连接的缺省路由。当您开启默认路由后，在打通VPC时，会将ModelArts网络0.0.0.0/0路由作为默认路由，此时无需提交工单添加缺省路由即可完成网络配置。

10.2.4.4 ECS 服务器挂载 SFS Turbo 存储

本小节介绍如何在ECS服务器挂载SFS Turbo存储，挂载完成后可在后续步骤中，将训练所需的数据通过ECS上传至SFS Turbo。

前提条件

- 已创建SFS Turbo，如果未创建，请参考**创建文件系统**。
- 数据及算法已经上传至OBS，如果未上传，请参考**上传数据和算法至OBS（首次使用时需要）**。
- ECS服务器和SFS的共享硬盘在相同的VPC或者对应VPC能够互联。
- ECS服务器基础镜像需要用Ubuntu 18.04的。
- ECS服务器和SFS Turbo需要在同一子网中。

操作步骤

1. 在ECS服务器中设置华为云镜像源。
```
sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
```
2. 安装NFS客户端，挂载对应盘。
```
sudo apt-get update
sudo apt-get install nfs-common
```
3. 获取SFS Turbo的挂载命令。
   a. 进入弹性文件服务SFS管理控制台。
   b. 选择“SFS Turbo”进入文件系统列表，单击文件系统名称，进入详情页面。
   c. 在“基本信息”页签获取并记录“Linux挂载命令”。
4. 在ECS服务器中挂载NFS存储。

   首先保证对应目录存在，然后输入对应指令即可。命令参考：
```
mkdir -p /mnt/sfs_turbo
mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
```

10.2.4.5 在 ECS 中创建 ma-user 和 ma-group

在ModelArts训练平台使用的自定义镜像时，默认用户为ma-user、默认用户组为ma-group。如果在训练时调用ECS中的文件，需要修改文件权限改为ma-user可读，否则会出现Permission denied错误，因此需要在ECS中提前创建好ma-user和ma-group。

在terminal中执行以下命令：

```
default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
    groupdel -f ${default_group}; \
    groupadd -g 100 ma-group; \
fi && \
if [ -z ${default_group} ]; then \
    groupadd -g 100 ma-group; \
fi && \
if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
    userdel -r ${default_user}; \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
if [ -z ${default_user} ]; then \
    useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
    chmod -R 750 /home/ma-user; \
fi && \
# set bash as default
rm /bin/sh && ln -s /bin/bash /bin/sh
```

查看创建的用户，执行以下命令：

```
id ma-user
```

如果出现以下信息则表示创建成功。

```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

10.2.4.6 obsutil 安装和配置

obsutil是用于访问、管理对象存储服务OBS的命令行工具，使用该工具可以对OBS进行常用的配置管理操作，如创建桶、上传文件/文件夹、下载文件/文件夹、删除文件/文件夹等。

obsutil安装和配置的具体操作指导请参见**obsutils快速入门**。

须知

操作命令中的AK/SK要换成用户实际获取的AK/SK，Endpoint可以参考**终端节点（Endpoint）和访问域名**获取。

10.2.4.7 （可选）工作空间配置

ModelArts支持设置子用户的细粒度权限、不同工作空间之间资源隔离。ModelArts工作空间帮您实现项目资源隔离、多项目分开结算等功能。

如果您开通了企业项目管理服务的权限，可以在创建工作空间的时候绑定企业项目ID，并在企业项目下添加用户组，为不同的用户组设置细粒度权限供组里的用户使用。

如果您未开通企业项目管理服务的权限，也可以在ModelArts创建自己独立的工作空间，但是无法使用跟企业项目相关的功能。

说明

工作空间为白名单功能，使用该功能需要提工单申请开通。

10.2.5 调试与训练

10.2.5.1 单机单卡

10.2.5.1.1 线下容器镜像构建及调试

镜像构建

1. 导出conda环境

   首先拉起线下的容器镜像：

   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```

   在容器中输入如下命令，得到pytorch.tar.gz：

   ```
   # run on container

   # 基于想要迁移的base环境创建一个名为pytorch的conda环境
   conda create --name pytorch --clone base

   pip install conda-pack

   #将pytorch env打包生成pytorch.tar.gz
   conda pack -n pytorch -o pytorch.tar.gz
   ```

   将打包好的压缩包传到本地：

   ```
   # run on terminal
   docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
   ```

   将pytorch.tar.gz上传到OBS并**设置公共读**，并在构建时wget获取、解压、清理。

2. 新镜像构建

   基础镜像一般选用ubuntu 18.04的官方镜像，或者nvidia官方提供的带cuda驱动的镜像。相关镜像直接到dockerhub官网查找即可。

构建流程：安装所需的apt包、驱动，配置ma-user用户、导入conda环境、配置Notebook依赖。

说明

- 推荐使用Dockerfile的方式构建镜像。这样既满足dockerfile可追溯及构建归档的需求，也保证镜像内容无冗余和残留。
- 每层构建的时候都尽量把tar包等中间态文件删除，保证最终镜像更小，清理缓存的方法可参考：**conda clean**。

3. 构建参考样例

Dockerfile样例：

```
FROM nvidia/cuda:11.3.1-cudnn8-devel-ubuntu18.04

USER root

# section1: add user ma-user whose uid is 1000 and user group ma-group whose gid is 100. If there
alreay exists 1000:100 but not ma-user:ma-group, below code will remove it
RUN default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" && \
    default_group=$(getent group 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" && \
    if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then \
        groupdel -f ${default_group}; \
        groupadd -g 100 ma-group; \
    fi && \
    if [ -z ${default_group} ]; then \
        groupadd -g 100 ma-group; \
    fi && \
    if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then \
        userdel -r ${default_user}; \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    if [ -z ${default_user} ]; then \
        useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user; \
        chmod -R 750 /home/ma-user; \
    fi && \
    # set bash as default
    rm /bin/sh && ln -s /bin/bash /bin/sh

# section2: config apt source and install tools needed.
RUN sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y ca-certificates curl ffmpeg git libgl1-mesa-glx libglib2.0-0 libibverbs-dev libjpeg-
dev libpng-dev libsm6 libxext6 libxrender-dev ninja-build screen sudo vim wget zip && \
    apt-get clean  && \
    rm -rf /var/lib/apt/lists/*

USER ma-user

# section3: install miniconda and rebuild conda env
RUN mkdir -p /home/ma-user/work/ && cd /home/ma-user/work/ && \
    wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    chmod 777 Miniconda3-py37_4.12.0-Linux-x86_64.sh && \
    bash Miniconda3-py37_4.12.0-Linux-x86_64.sh -bfp /home/ma-user/anaconda3 && \
    wget https://${bucketname}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz && \
    mkdir -p /home/ma-user/anaconda3/envs/pytorch && \
    tar -xzf pytorch.tar.gz -C /home/ma-user/anaconda3/envs/pytorch && \
    source /home/ma-user/anaconda3/envs/pytorch/bin/activate && conda-unpack && \
    /home/ma-user/anaconda3/bin/conda init bash && \
    rm -rf /home/ma-user/work/*

ENV PATH=/home/ma-user/anaconda3/envs/pytorch/bin:$PATH

# section4: settings of Jupyter Notebook for pytorch env
RUN source /home/ma-user/anaconda3/envs/pytorch/bin/activate && \
    pip install ipykernel==6.7.0 --trusted-host https://repo.huaweicloud.com -i https://
repo.huaweicloud.com/repository/pypi/simple && \
```

```
    ipython kernel install --user --env PATH /home/ma-user/anaconda3/envs/pytorch/bin:$PATH --
name=pytorch && \
    rm -rf /home/ma-user/.local/share/jupyter/kernels/pytorch/logo-* && \
    rm -rf ~/.cache/pip/* && \
    echo 'export PATH=$PATH:/home/ma-user/.local/bin' >> /home/ma-user/.bashrc && \
    echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/nvidia/lib64' >> /home/ma-
user/.bashrc && \
    echo 'conda activate pytorch' >> /home/ma-user/.bashrc

ENV DEFAULT_CONDA_ENV_NAME=pytorch
```

说明

Dockerfile中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz"，需要替换为1中pytorch.tar.gz在OBS上的路径（需将文件设置为公共读）。

进入Dockerfile目录，通过Dockerfile构建镜像命令：

```
# cd 到Dockerfile所在目录下，输入构建命令
# docker build -t ${image_name}:${image_version} .
# 例如
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

说明

- 容器镜像的大小建议小于15G，不能大于25G。否则镜像的迁移、拉起都会存在性能问题。
- 建议通过开源的官方镜像来构建，例如PyTorch的官方镜像。
- 建议容器分层构建，单层容量不要超过1G、文件数不大于10w个。分层时，先构建不常变化的层，例如：先OS，再cuda驱动，再Python，再pytorch，再其他依赖包。
- 不建议把数据、代码放到容器镜像里。因为对应内容应该是经常变动的，会导致频繁的容器镜像构建操作。
- 不建议在容器内再创建多个conda env。因为容器已经能满足隔离需求，没有必要再通过conda env做隔离。
- 本教程通过打包conda env来构建环境，也可以通过pip install、conda install等方式安装conda环境的依赖。
- 更多ModelArts自定义镜像介绍请见**自定义镜像简介**。

调试要点

1. 确认对应的脚本、代码、流程在linux服务器上运行正常。

   如果在linux服务器上运行就有问题，那么先调通以后再做容器镜像。

2. 确认打入镜像的文件是否在正确的位置、是否有正确的权限。

   训练场景主要查看自研的依赖包是否正常，查看pip list是否包含所需的包，查看容器直接调用的python是否是自己所需要的那个（如果容器镜像装了多个python，需要设置python路径的环境变量）。

3. 测试训练启动脚本。

   a. 优先使用手工进行数据复制的工作并验证

      一般在镜像里不包含训练所用的数据和代码，所以在启动镜像以后需要手工把需要的文件复制进去。建议数据、代码和中间数据都放到"/cache"目录，防止正式运行时磁盘占满（请见**ModelArts环境挂载目录说明**）。建议linux服务器申请的时候，有足够大的内存（8G以上）以及足够大的硬盘（100G以上）。

      docker和linux的文件交互命令如下：

      ```
      docker cp data/ 39c9ceedb1f6:/cache/
      ```

      数据准备完成后，启动训练的脚本，查看训练是否能够正常拉起。一般来说，启动脚本为：

```
cd /cache/code/
python start_train.py
```

如果训练流程不符合预期，可以在容器实例中查看日志、错误等，并进行代码、环境变量的修正。

b. 预制脚本测试整体流程

一般使用run.sh封装训练外的文件复制工作（数据、代码：OBS-->容器，输出结果：容器-->OBS），run.sh的构建方法参考**run.sh脚本测试ModelArts训练整体流程**。

如果预制脚本调用结果不符合预期，可以在容器实例中进行修改和迭代。

c. 针对专属池场景

由于专属池支持SFS挂载，因此代码、数据的导入会更简单，甚至可以不用再关注OBS的相关操作。

可以直接把SFS的目录直接挂载到调试节点的"/mnt/sfs_turbo"目录，或者保证对应目录的内容和SFS盘匹配。

调试时建议使用接近的方式，即：启动容器实例时使用"-v"参数来指定挂载某个宿主机目录到容器环境。

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

上述命令表示把宿主机的"/mnt/sfs_turbo"目录挂载到容器的"/sfs"目录，在宿主机和容器对应目录的所有改动都是实时同步的。

4. 分析错误时：训练镜像先看日志，推理镜像先看API的返回。

   可以通过命令查看容器输出到stdout的所有日志：

   ```
   docker logs -f 39c9ceedb1f6
   ```

   一般在做推理镜像时，部分日志是直接存储在容器内部的，所以需要进入容器看日志。注意：重点对应日志中是否有ERROR（包括，容器启动时、API执行时）。

5. 牵扯部分文件用户组不一致的情况，可以在宿主机用root权限执行命令进行修改

   ```
   docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
   ```

6. 针对调试中遇到的错误，可以直接在容器实例里修改，修改结果可以通过commit命令持久化。

说明

建议把调试过程中的修改点通过Dockerfile固化到容器构建正式流程，并重新测试。

10.2.5.1.2 上传镜像

操作场景

客户端上传镜像，是指在安装了容器引擎客户端的机器上使用docker命令将镜像上传到容器镜像服务的镜像仓库。

如果容器引擎客户端机器为云上的ECS或CCE节点，根据机器所在区域有两种网络链路可以选择：

- 如果机器与容器镜像仓库在同一区域，则上传镜像走内网链路。
- 如果机器与容器镜像仓库不在同一区域，则上传镜像走公网链路，机器需要绑定弹性公网IP。

约束与限制

- 使用客户端上传镜像，镜像的每个layer大小不能大于10G。

- 上传镜像的容器引擎客户端版本必须为1.11.2及以上。

操作步骤

1. 连接容器镜像服务。
   a. 登录容器镜像服务控制台。
   b. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
   c. 选择左侧导航栏的“总览”，单击页面右上角的“登录指令”，在弹出的页面中单击复制登录指令。

      说明

      - 此处生成的登录指令有效期为24小时，如果需要长期有效的登录指令，请参见**获取长期有效登录指令**。获取了长期有效的登录指令后，在有效期内的临时登录指令仍然可以使用。
      - 登录指令末尾的域名为镜像仓库地址，请记录该地址，后面会使用到。

   d. 在安装容器引擎的机器中执行上一步复制的登录指令。
      登录成功会显示“Login Succeeded”。
2. 在安装容器引擎的机器上执行如下命令，为镜像打标签。
   docker tag [镜像名称1:版本名称1] [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]
   - [镜像名称1:版本名称1]：${image_name}:${image_version}请替换为您所要上传的实际镜像的名称和版本名称。
   - [镜像仓库地址]：可在SWR控制台上查询，即**1.c**中登录指令末尾的域名。
   - [组织名称]：/${organization_name}请替换为您创建的组织。
   - [镜像名称2:版本名称2]：${image_name}:${image_version}请替换为您期待的镜像名称和镜像版本。

   示例：

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```
3. 上传镜像至镜像仓库。
   docker push [镜像仓库地址]/[组织名称]/[镜像名称2:版本名称2]

   示例：

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   上传镜像完成后，返回容器镜像服务控制台，在“我的镜像”页面，执行刷新操作后可查看到对应的镜像信息。

常见问题

为什么使用客户端上传镜像失败?

10.2.5.1.3 上传数据和算法至 OBS（首次使用时需要）

前提条件

- 已经在OBS上创建好并行文件系统，请参见**创建并行文件系统**。
- 已经在obsutil安装和配置，请参见**obsutils安装和配置**。

准备数据

1. 单击下载**动物数据集**至本地，并解压。
2. 通过obsutil将数据集上传至OBS桶中。

```
./obsutil cp ./dog_cat_1w obs://${your_obs_buck}/demo/ -f -r
```

说明

OBS支持多种文件上传方式，当文件少于100个时，可以在OBS Console中上传，当文件大于100个时，推荐使用工具，推荐**OBS Browser+**（win）、**obsutil**（linux）。上述例子为obsutil使用方法。

准备算法

main.py文件内容如下，并将其上传至OBS桶的demo文件夹中：

```
import argparse
import os
import random
import shutil
import time
import warnings
from enum import Enum
import torch
import torch.nn as nn
import torch.nn.parallel
import torch.backends.cudnn as cudnn
import torch.distributed as dist
import torch.optim
from torch.optim.lr_scheduler import StepLR
import torch.multiprocessing as mp
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
model_names = sorted(name for name in models.__dict__
                     if name.islower() and not name.startswith("__")
                     and callable(models.__dict__[name]))
parser = argparse.ArgumentParser(description='PyTorch ImageNet Training')
parser.add_argument('data', metavar='DIR', default='imagenet',
                    help='path to dataset (default: imagenet)')
parser.add_argument('-a', '--arch', metavar='ARCH', default='resnet18',
                    choices=model_names,
                    help='model architecture: ' +
                        ' | '.join(model_names) +
                        ' (default: resnet18)')
parser.add_argument('-j', '--workers', default=4, type=int, metavar='N',
                    help='number of data loading workers (default: 4)')
parser.add_argument('--epochs', default=90, type=int, metavar='N',
                    help='number of total epochs to run')
parser.add_argument('--start-epoch', default=0, type=int, metavar='N',
                    help='manual epoch number (useful on restarts)')
parser.add_argument('-b', '--batch-size', default=256, type=int,
                    metavar='N',
                    help='mini-batch size (default: 256), this is the total '
                        'batch size of all GPUs on the current node when '
```

```
                    'using Data Parallel or Distributed Data Parallel')
parser.add_argument('--lr', '--learning-rate', default=0.1, type=float,
                    metavar='LR', help='initial learning rate', dest='lr')
parser.add_argument('--momentum', default=0.9, type=float, metavar='M',
                    help='momentum')
parser.add_argument('--wd', '--weight-decay', default=1e-4, type=float,
                    metavar='W', help='weight decay (default: 1e-4)',
                    dest='weight_decay')
parser.add_argument('-p', '--print-freq', default=10, type=int,
                    metavar='N', help='print frequency (default: 10)')
parser.add_argument('--resume', default='', type=str, metavar='PATH',
                    help='path to latest checkpoint (default: none)')
parser.add_argument('-e', '--evaluate', dest='evaluate', action='store_true',
                    help='evaluate model on validation set')
parser.add_argument('--pretrained', dest='pretrained', action='store_true',
                    help='use pre-trained model')
parser.add_argument('--world-size', default=-1, type=int,
                    help='number of nodes for distributed training')
parser.add_argument('--rank', default=-1, type=int,
                    help='node rank for distributed training')
parser.add_argument('--dist-url', default='tcp://224.66.41.62:23456', type=str,
                    help='url used to set up distributed training')
parser.add_argument('--dist-backend', default='nccl', type=str,
                    help='distributed backend')
parser.add_argument('--seed', default=None, type=int,
                    help='seed for initializing training. ')
parser.add_argument('--gpu', default=None, type=int,
                    help='GPU id to use.')
parser.add_argument('--multiprocessing-distributed', action='store_true',
                    help='Use multi-processing distributed training to launch '
                         'N processes per node, which has N GPUs. This is the '
                         'fastest way to use PyTorch for either single node or '
                         'multi node data parallel training')
best_acc1 = 0


def main():
    args = parser.parse_args()
    if args.seed is not None:
        random.seed(args.seed)
        torch.manual_seed(args.seed)
        cudnn.deterministic = True
        warnings.warn('You have chosen to seed training. '
                      'This will turn on the CUDNN deterministic setting, '
                      'which can slow down your training considerably! '
                      'You may see unexpected behavior when restarting '
                      'from checkpoints.')
    if args.gpu is not None:
        warnings.warn('You have chosen a specific GPU. This will completely '
                      'disable data parallelism.')
    if args.dist_url == "env://" and args.world_size == -1:
        args.world_size = int(os.environ["WORLD_SIZE"])
    args.distributed = args.world_size > 1 or args.multiprocessing_distributed
    ngpus_per_node = torch.cuda.device_count()
    if args.multiprocessing_distributed:
        # Since we have ngpus_per_node processes per node, the total world_size
        # needs to be adjusted accordingly
        args.world_size = ngpus_per_node * args.world_size
        # Use torch.multiprocessing.spawn to launch distributed processes: the
        # main_worker process function
        mp.spawn(main_worker, nprocs=ngpus_per_node, args=(ngpus_per_node, args))
    else:
        # Simply call main_worker function
        main_worker(args.gpu, ngpus_per_node, args)
def main_worker(gpu, ngpus_per_node, args):
    global best_acc1
    args.gpu = gpu
    if args.gpu is not None:
        print("Use GPU: {} for training".format(args.gpu))
```

```
if args.distributed:
    if args.dist_url == "env://" and args.rank == -1:
        args.rank = int(os.environ["RANK"])
    if args.multiprocessing_distributed:
        # For multiprocessing distributed training, rank needs to be the
        # global rank among all the processes
        args.rank = args.rank * ngpus_per_node + gpu
    dist.init_process_group(backend=args.dist_backend, init_method=args.dist_url,
                            world_size=args.world_size, rank=args.rank)
# create model
if args.pretrained:
    print("=> using pre-trained model '{}'".format(args.arch))
    model = models.__dict__[args.arch](pretrained=True)
else:
    print("=> creating model '{}'".format(args.arch))
    model = models.__dict__[args.arch]()
if not torch.cuda.is_available():
    print('using CPU, this will be slow')
elif args.distributed:
    # For multiprocessing distributed, DistributedDataParallel constructor
    # should always set the single device scope, otherwise,
    # DistributedDataParallel will use all available devices.
    if args.gpu is not None:
        torch.cuda.set_device(args.gpu)
        model.cuda(args.gpu)
        # When using a single GPU per process and per
        # DistributedDataParallel, we need to divide the batch size
        # ourselves based on the total number of GPUs of the current node.
        args.batch_size = int(args.batch_size / ngpus_per_node)
        args.workers = int((args.workers + ngpus_per_node - 1) / ngpus_per_node)
        model = torch.nn.parallel.DistributedDataParallel(model, device_ids=[args.gpu])
    else:
        model.cuda()
        # DistributedDataParallel will divide and allocate batch_size to all
        # available GPUs if device_ids are not set
        model = torch.nn.parallel.DistributedDataParallel(model)
elif args.gpu is not None:
    torch.cuda.set_device(args.gpu)
    model = model.cuda(args.gpu)
else:
    # DataParallel will divide and allocate batch_size to all available GPUs
    if args.arch.startswith('alexnet') or args.arch.startswith('vgg'):
        model.features = torch.nn.DataParallel(model.features)
        model.cuda()
    else:
        model = torch.nn.DataParallel(model).cuda()
# define loss function (criterion), optimizer, and learning rate scheduler
criterion = nn.CrossEntropyLoss().cuda(args.gpu)
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
"""Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
# optionally resume from a checkpoint
if args.resume:
    if os.path.isfile(args.resume):
        print("=> loading checkpoint '{}'".format(args.resume))
        if args.gpu is None:
            checkpoint = torch.load(args.resume)
        else:
            # Map model to be loaded to specified single gpu.
            loc = 'cuda:{}'.format(args.gpu)
            checkpoint = torch.load(args.resume, map_location=loc)
        args.start_epoch = checkpoint['epoch']
        best_acc1 = checkpoint['best_acc1']
        if args.gpu is not None:
            # best_acc1 may be from a checkpoint from a different GPU
            best_acc1 = best_acc1.to(args.gpu)
        model.load_state_dict(checkpoint['state_dict'])
```

```
            optimizer.load_state_dict(checkpoint['optimizer'])
            scheduler.load_state_dict(checkpoint['scheduler'])
            print("=> loaded checkpoint '{}' (epoch {})"
                  .format(args.resume, checkpoint['epoch']))
        else:
            print("=> no checkpoint found at '{}'".format(args.resume))
    cudnn.benchmark = True

    # Data loading code
    traindir = os.path.join(args.data, 'train')
    valdir = os.path.join(args.data, 'val')
    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
    train_dataset = datasets.ImageFolder(
        traindir,
        transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.ToTensor(),
            normalize,
        ]))
    if args.distributed:
        train_sampler = torch.utils.data.distributed.DistributedSampler(train_dataset)
    else:
        train_sampler = None

    train_loader = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size, shuffle=(train_sampler is None),
        num_workers=args.workers, pin_memory=True, sampler=train_sampler)
    val_loader = torch.utils.data.DataLoader(
        datasets.ImageFolder(valdir, transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            normalize,
        ])),
        batch_size=args.batch_size, shuffle=False,
        num_workers=args.workers, pin_memory=True)
    if args.evaluate:
        validate(val_loader, model, criterion, args)
        return

    for epoch in range(args.start_epoch, args.epochs):
        if args.distributed:
            train_sampler.set_epoch(epoch)
        # train for one epoch
        train(train_loader, model, criterion, optimizer, epoch, args)
        # evaluate on validation set
        acc1 = validate(val_loader, model, criterion, args)
        scheduler.step()
        # remember best acc@1 and save checkpoint
        is_best = acc1 > best_acc1
        best_acc1 = max(acc1, best_acc1)
        if not args.multiprocessing_distributed or (args.multiprocessing_distributed
                                                    and args.rank % ngpus_per_node == 0):
            save_checkpoint({
                'epoch': epoch + 1,
                'arch': args.arch,
                'state_dict': model.state_dict(),
                'best_acc1': best_acc1,
                'optimizer': optimizer.state_dict(),
                'scheduler': scheduler.state_dict()
            }, is_best)
def train(train_loader, model, criterion, optimizer, epoch, args):
    batch_time = AverageMeter('Time', ':6.3f')
    data_time = AverageMeter('Data', ':6.3f')
    losses = AverageMeter('Loss', ':.4e')
    top1 = AverageMeter('Acc@1', ':6.2f')
    top5 = AverageMeter('Acc@5', ':6.2f')
```

```
    progress = ProgressMeter(
        len(train_loader),
        [batch_time, data_time, losses, top1, top5],
        prefix="Epoch: [{}]".format(epoch))
    # switch to train mode
    model.train()
    end = time.time()
    for i, (images, target) in enumerate(train_loader):
        # measure data loading time
        data_time.update(time.time() - end)
        if args.gpu is not None:
            images = images.cuda(args.gpu, non_blocking=True)
        if torch.cuda.is_available():
            target = target.cuda(args.gpu, non_blocking=True)
        # compute output
        output = model(images)
        loss = criterion(output, target)
        # measure accuracy and record loss
        acc1, acc5 = accuracy(output, target, topk=(1, 5))
        losses.update(loss.item(), images.size(0))
        top1.update(acc1[0], images.size(0))
        top5.update(acc5[0], images.size(0))
        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # measure elapsed time
        batch_time.update(time.time() - end)
        end = time.time()
        if i % args.print_freq == 0:
            progress.display(i)
def validate(val_loader, model, criterion, args):
    batch_time = AverageMeter('Time', ':6.3f', Summary.NONE)
    losses = AverageMeter('Loss', ':.4e', Summary.NONE)
    top1 = AverageMeter('Acc@1', ':6.2f', Summary.AVERAGE)
    top5 = AverageMeter('Acc@5', ':6.2f', Summary.AVERAGE)
    progress = ProgressMeter(
        len(val_loader),
        [batch_time, losses, top1, top5],
        prefix='Test: ')
    # switch to evaluate mode
    model.eval()
    with torch.no_grad():
        end = time.time()
        for i, (images, target) in enumerate(val_loader):
            if args.gpu is not None:
                images = images.cuda(args.gpu, non_blocking=True)
            if torch.cuda.is_available():
                target = target.cuda(args.gpu, non_blocking=True)
            # compute output
            output = model(images)
            loss = criterion(output, target)
            # measure accuracy and record loss
            acc1, acc5 = accuracy(output, target, topk=(1, 5))
            losses.update(loss.item(), images.size(0))
            top1.update(acc1[0], images.size(0))
            top5.update(acc5[0], images.size(0))
            # measure elapsed time
            batch_time.update(time.time() - end)
            end = time.time()
            if i % args.print_freq == 0:
                progress.display(i)
        progress.display_summary()
    return top1.avg
def save_checkpoint(state, is_best, filename='checkpoint.pth.tar'):
    torch.save(state, filename)
    if is_best:
        shutil.copyfile(filename, 'model_best.pth.tar')
class Summary(Enum):
```

```
    NONE = 0
    AVERAGE = 1
    SUM = 2
    COUNT = 3


class AverageMeter(object):
    """Computes and stores the average and current value"""

    def __init__(self, name, fmt=':f', summary_type=Summary.AVERAGE):
        self.name = name
        self.fmt = fmt
        self.summary_type = summary_type
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
    def __str__(self):
        fmtstr = '{name} {val' + self.fmt + '} ({avg' + self.fmt + '})'
        return fmtstr.format(**self.__dict__)
    def summary(self):
        fmtstr = ''
        if self.summary_type is Summary.NONE:
            fmtstr = ''
        elif self.summary_type is Summary.AVERAGE:
            fmtstr = '{name} {avg:.3f}'
        elif self.summary_type is Summary.SUM:
            fmtstr = '{name} {sum:.3f}'
        elif self.summary_type is Summary.COUNT:
            fmtstr = '{name} {count:.3f}'
        else:
            raise ValueError('invalid summary type %r' % self.summary_type)
        return fmtstr.format(**self.__dict__)
class ProgressMeter(object):
    def __init__(self, num_batches, meters, prefix=""):
        self.batch_fmtstr = self._get_batch_fmtstr(num_batches)
        self.meters = meters
        self.prefix = prefix
    def display(self, batch):
        entries = [self.prefix + self.batch_fmtstr.format(batch)]
        entries += [str(meter) for meter in self.meters]
        print('\t'.join(entries))
    def display_summary(self):
        entries = [" *"]
        entries += [meter.summary() for meter in self.meters]
        print(' '.join(entries))
    def _get_batch_fmtstr(self, num_batches):
        num_digits = len(str(num_batches // 1))
        fmt = '{:' + str(num_digits) + 'd}'
        return '[' + fmt + '/' + fmt.format(num_batches) + ']'


def accuracy(output, target, topk=(1,)):
    """Computes the accuracy over the k top predictions for the specified values of k"""
    with torch.no_grad():
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
        correct = pred.eq(target.view(1, -1).expand_as(pred))
```

```
        res = []
        for k in topk:
            correct_k = correct[:k].reshape(-1).float().sum(0, keepdim=True)
            res.append(correct_k.mul_(100.0 / batch_size))
        return res
if __name__ == '__main__':
    main()
```

10.2.5.1.4 使用 Notebook 进行代码调试

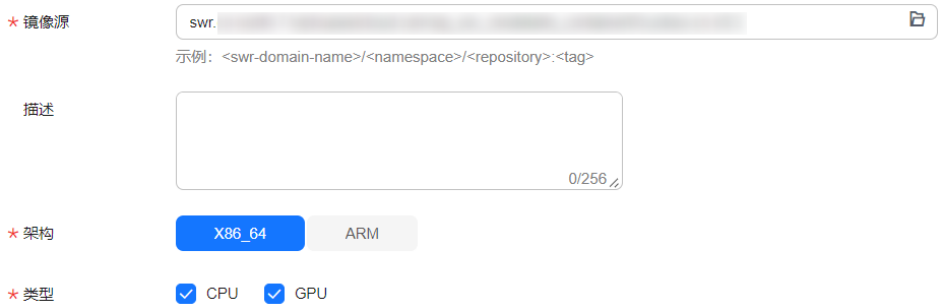
背景信息

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产品不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个帐户最多创建10个Notebook。

创建 Notebook 实例

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如**图10-17**所示。

   图 10-17 注册镜像

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见**表10-5**。

表 10-5 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

b. 填写Notebook详细参数，如镜像、资源规格等。

- 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
- 资源类型：按实际情况选择已创建的专属资源池。
- 规格：选择1 GPU规格。
- 存储配置：选择“云硬盘EVS”作为存储位置。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接Notebook方式介绍**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。

   如果创建Notebook启动失败，建议参考**调试要点**进行检查。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 挂载OBS并行文件系统：在Notebook实例详情页面，选择“存储配置”页签，单击“添加数据存储”，设置挂载参数。

   a. 设置本地挂载目录，在“/data/”目录下输入一个文件夹名称，例如：demo。挂载时，后台自动会在Notebook容器“的/data/”目录下创建该文件夹，用来挂载OBS文件系统。

   b. 选择存放OBS并行文件系统下的文件夹，单击“确定”。
8. 挂载成功后，可以在Notebook实例详情页查看到挂载结果。
9. 代码调试。

   打开Notebook，打开Terminal，进入步骤7中挂载的目录。

   ```
   cd /data/demo
   ```

   执行训练命令：

   ```
   /home/ma-user/anaconda3/envs/pytorch/bin/python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
   ```

告警"RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.12) doesn't match a supported version!"不影响训练，可忽略。

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

10.2.5.1.5 创建训练任务

说明

针对专属池场景，应注意挂载的目录设置和调试时一致。

1. 登录ModelArts管理控制台，检查当前帐号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd ${MA_JOB_DIR}/demo && python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
     ```
     此处的“demo”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择单GPU规格。
4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

10.2.5.1.6 监控资源

用户可以通过资源占用情况窗口查看计算节点的资源使用情况，最多可显示最近三天的数据。在资源占用情况窗口打开时，会定期向后台获取最新的资源使用率数据并刷新。

操作一：如果训练作业使用多个计算节点，可以通过实例名称的下拉框切换节点。

操作二：单击图例“cpuUsage”、“gpuMemUsage”、“gpuUtil”、“memUsage”“npuMemUsage”、“npuUtil”、可以添加或取消对应参数的使用情况图。

操作三：鼠标悬浮在图片上的时间节点，可查看对应时间节点的占用率情况。

表 10-6 参数说明

参数	说明
cpuUsage	cpu使用率。
gpuMemUsage	gpu内存使用率。
gpuUtil	gpu使用情况。
memUsage	内存使用率。
npuMemUsage	npu内存使用率。
npuUtil	npu使用情况。

10.2.5.2 单机多卡

10.2.5.2.1 线下容器镜像构建及调试

镜像构建及调试与单机单卡相同，请参考**线下容器镜像构建及调试**。

10.2.5.2.2 上传镜像

请参考**上传镜像**。

10.2.5.2.3 上传数据和算法至 SFS（首次使用时需要）

前提条件

- ECS服务器已挂载SFS，请参考**ECS服务器挂载SFS Turbo存储**。
- 已经创建好，请参考**在ECS中创建ma-user和ma-group**。
- 已经安装obsutil，请参考**下载和安装obsutil**。

准备数据

1. 登录coco数据集下载官网地址：**https://cocodataset.org/#download**
2. 下载coco2017数据集的Train（18GB）、Val images（1GB）、Train/Val annotations（241MB），分别解压后并放入coco文件夹中。
3. 下载完成后，将数据上传至SFS相应目录中。由于数据集过大，推荐先通过obsutil工具将数据集传到OBS桶后，再将数据集迁移至SFS。
   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。

```
# 将本地数据传至OBS中
# ./obsutil cp ${数据集所在的本地文件夹路径} ${存放数据集的obs文件夹路径} -f -r
# 例如
./obsutil cp ./coco obs://your_bucket/ -f -r
```

   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：

```
# 将OBS数据传至SFS中
# ./obsutil cp ${数据集所在的obs文件夹路径} ${SFS文件夹路径} -f -r
# 例如
./obsutil cp obs://your_bucket/coco/ /mnt/sfs_turbo/ -f -r
```

/mnt/sfs_turbo/coco文件夹内目录结构如下：

```
coco
|---annotations
|---train2017
|---val2017
```

更多obsutil的操作，可参考**obsutil简介**。

c. 将文件设置归属为ma-user：

```
chown -R ma-user:ma-group coco
```

代码云上适配

1. 下载YOLOX代码。代码仓地址：**https://github.com/Megvii-BaseDetection/YOLOX.git**。

```
git clone https://github.com/Megvii-BaseDetection/YOLOX.git
cd YOLOX
git checkout 4f8f1d79c8b8e530495b5f183280bab99869e845
```

2. 修改“requirements.txt”中的onnx版本，改为“onnx>=1.12.0”。
3. 将“yolox/data/datasets/coco.py”第59行的“data_dir = os.path.join(get_yolox_datadir(), "COCO")”改为“data_dir = '/home/ma-user/coco'”。

```
# data_dir = os.path.join(get_yolox_datadir(), "COCO")
data_dir = '/home/ma-user/coco'
```

4. 在“tools/train.py”的第13行前加两句代码。

```
# 加上这两句代码，防止运行时找不到yolox module
import sys
sys.path.append(os.getcwd())

# line13
from yolox.core import launch
from yolox.exp import Exp, get_exp
```

5. 将“yolox/layers/jit_ops.py”第122行的“fast_cocoeval”改为“fast_coco_eval_api”。

```
# def __init__(self, name="fast_cocoeval"):
def __init__(self, name="fast_coco_eval_api"):
```

6. 将“yolox\evaluators\coco_evaluator.py”第294行的“from yolox.layers import COCOeval_opt as COCOeval”改为“from pycocotools.cocoeval import COCOeval”。

```
try:
  # from yolox.layers import COCOeval_opt as COCOeval
  from pycocotools.cocoeval import COCOeval
except ImportError:
  from pycocotools.cocoeval import COCOeval

  logger.warning("Use standard COCOeval.")
```

7. 在tools目录下新建一个“run.sh”作为启动脚本，“run.sh”内容可参考：

```
#!/usr/bin/env sh
set -x
set -o pipefail

export NCCL_DEBUG=INFO

DEFAULT_ONE_GPU_BATCH_SIZE=32
BATCH_SIZE=$((${MA_NUM_GPUS:-8} * ${VC_WORKER_NUM:-1} * $
{DEFAULT_ONE_GPU_BATCH_SIZE}))
if [ ${VC_WORKER_HOSTS} ];then
    YOLOX_DIST_URL=tcp://$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1):6666
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                            -n yolox-s \
                            --devices ${MA_NUM_GPUS:-8} \
                            --batch-size ${BATCH_SIZE} \
```

```
                        --fp16 \
                        --occupy \
                        --num_machines ${VC_WORKER_NUM:-1} \
                        --machine_rank ${VC_TASK_INDEX:-0} \
                        --dist-url ${YOLOX_DIST_URL}
else
    /home/ma-user/anaconda3/envs/pytorch/bin/python -u tools/train.py \
                        -n yolox-s \
                        --devices ${MA_NUM_GPUS:-8} \
                        --batch-size ${BATCH_SIZE} \
                        --fp16 \
                        --occupy \
                        --num_machines ${VC_WORKER_NUM:-1} \
                        --machine_rank ${VC_TASK_INDEX:-0}
fi
```

说明

部分环境变量在Notebook环境中不存在，因此需要提供默认值。

8. 将代码放到OBS上，然后通过OBS将代码传至SFS相应目录中。
   a. 在本机机器上运行，通过obsutil工具将本地数据集传到OBS桶。
   ```
   # 将本地代码传至OBS中
   ./obsutil cp ./YOLOX obs://your_bucket/ -f -r
   ```
   b. 登录ECS服务器，通过obsutil工具将数据集迁移至SFS，样例代码如下：
   ```
   # 将OBS的代码传到SFS中
   ./obsutil cp obs://your_bucket/YOLOX/ /mnt/sfs_turbo/code/ -f -r
   ```

说明

本案例中以obsutils方式上传文件，除此之外也可通过SCP方式上传文件，具体操作步骤可参考**本地Linux主机使用SCP上传文件到Linux云服务器**。

9. 在SFS中将文件设置归属为ma-user。
   ```
   chown -R ma-user:ma-group YOLOX
   ```
10. 执行以下命令，去除Shell脚本的\r字符。
    ```
    cd YOLOX
    sed -i 's/\r//' run.sh
    ```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

10.2.5.2.4 使用 Notebook 进行代码调试

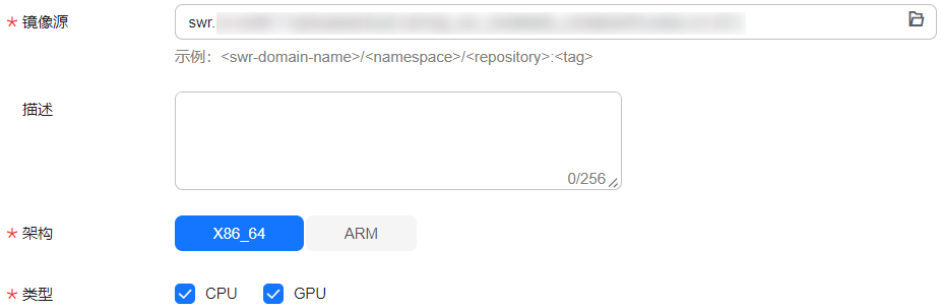
背景信息

- Notebook使用涉及到计费，具体收费项如下：
  - 处于“运行中”状态的Notebook，会消耗资源，产生费用。根据您选择的资源不同，收费标准不同，价格详情请参见**产品价格详情**。当您不需要使用Notebook时，建议停止Notebook，避免产生不必要的费用。
  - 创建Notebook时，如果选择使用云硬盘EVS存储配置，云硬盘EVS会一直收费，建议及时停止并删除Notebook，避免产品不必要的费用。
- 在创建Notebook时，默认会开启自动停止功能，在指定时间内停止运行Notebook，避免资源浪费。
- 只有处于“运行中”状态的Notebook，才可以执行打开、停止操作。
- 一个帐户最多创建10个Notebook。

创建 Notebook 实例

1. 注册镜像。登录ModelArts控制台，在左侧导航栏选择“镜像管理”，进入镜像管理页面。单击“注册镜像”，镜像源即为推送到SWR中的镜像。请将完整的SWR地址复制到这里即可，或单击 可直接从SWR选择自有镜像进行注册，类型加上“GPU”，如图10-18所示。

   图 10-18 注册镜像

2. 登录ModelArts管理控制台，在左侧导航栏中选择“开发空间 > Notebook”，进入“Notebook”列表页面。
3. 单击“创建”，进入“创建Notebook”页面，请参见如下说明填写参数。
   a. 填写Notebook基本信息，包含名称、描述、是否自动停止，详细参数请参见表10-7。

      表 10-7 基本信息的参数描述

参数名称	说明
“名称”	Notebook的名称。只能包含数字、大小写字母、下划线和中划线，长度不能大于64位且不能为空。
“描述”	对Notebook的简要描述。
“自动停止”	默认开启，且默认值为“1小时”，表示该Notebook实例将在运行1小时之后自动停止，即1小时后停止规格资源计费。 开启自动停止功能后，可选择“1小时”、“2小时”、“4小时”、“6小时”或“自定义”几种模式。选择“自定义”模式时，可指定1~24小时范围内任意整数。

   b. 填写Notebook详细参数，如镜像、资源规格等。
      - 镜像：在“自定义镜像”页签选择已上传的自定义镜像。
      - 资源类型：按实际情况选择已创建的专属资源池。
      - 规格：选择8卡GPU规格，“run.sh”文件中默认MA_NUM_GPUS为8卡，因此选择notebook规格时需要与MA_NUM_GPUS默认值相同。
      - 存储配置：选择“弹性文件服务SFS”作为存储位置。子目录挂载可不填写，如果需挂载SFS指定目录，则在子目录挂载处填写具体路径。

说明

如果需要通过VS Code连接Notebook方式进行代码调试，则需开启“SSH远程开发”并选择密钥对，请参考**VS Code连接N**。

4. 参数填写完成后，单击“立即创建”进行规格确认。
5. 参数确认无误后，单击“提交”，完成Notebook的创建操作。

   进入Notebook列表，正在创建中的Notebook状态为“创建中”，创建过程需要几分钟，请耐心等待。当Notebook状态变为“运行中”时，表示Notebook已创建并启动完成。
6. 在Notebook列表，单击实例名称，进入实例详情页，查看Notebook实例配置信息。
7. 在Notebook中打开Terminal，输入启动命令调试代码。

```
# 建立数据集软链接
# ln -s /home/ma-user/work/${coco数据集在SFS上的路径} /home/ma-user/coco
# 进入到对应目录
# cd /home/ma-user/work/${YOLOX在SFS上的路径}
# 安装环境并执行脚本
# /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh

# 例如
ln -s /home/ma-user/work/coco /home/ma-user/coco
cd /home/ma-user/work/code/YOLOX/
/home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
```

说明

Notebook中调试完后，如果镜像有修改，可以保存镜像用于后续训练，具体操作请参见**保存Notebook镜像环境**。

10.2.5.2.5 创建训练任务

1. 登录ModelArts管理控制台，检查当前帐号是否已完成访问授权的配置。如果未完成，请参考**使用委托授权**针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。单击“创建训练作业”进入创建训练作业页面。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：

     ```
     ln -s /home/ma-user/work/coco /home/ma-user/coco && cd /home/ma-user/work/code/YOLOX/ && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择8卡GPU规格。
   - 计算节点：1。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，因此云上挂载路径需要填写为“/home/ma-user/work”。

4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

10.2.5.3 多机多卡

10.2.5.3.1 线下容器镜像构建及调试

镜像构建及调试与单机单卡相同，请参考**线下容器镜像构建及调试**。

10.2.5.3.2 上传镜像

请参考**上传镜像**。

10.2.5.3.3 上传数据至 OBS（首次使用时需要）

前提条件

- 已经在OBS上创建好普通OBS桶，请参见**创建普通OBS桶**。
- 已经安装obsutil，请参考**下载和安装obsutil**。

操作步骤

1. 登录Imagenet数据集下载官网地址，下载Imagenet21k数据集：**http://image-net.org/**
2. 下载格式转换后的annotation文件：
   ILSVRC2021winner21k_whole_map_train.txt和
   ILSVRC2021winner21k_whole_map_val.txt。
3. 下载完成后将上述3个文件数据上传至OBS桶中的imagenet21k_whole文件夹中。上传方法请参考**上传数据和算法至OBS（首次使用时需要）**。

10.2.5.3.4 上传算法至 SFS

1. 下载Swin-Transformer代码。
   ```
   git clone --recursive https://github.com/microsoft/Swin-Transformer.git
   ```
2. 修改lr_scheduler.py文件，把第27行：t_mul=1. 注释掉。
3. 修改data文件夹下imagenet22k_dataset.py，把第28行：print("ERROR IMG LOADED: ", path) 注释掉。
4. 修改data文件夹下的build.py文件，把第112行：prefix = 'ILSVRC2011fall_whole'，改为prefix = 'ILSVRC2021winner21k_whole'。
5. 在Swin-Transformer目录下创建requirements.txt指定python依赖库：
   ```
   # requirements.txt内容如下

   timm==0.4.12
   termcolor==1.1.0
   yacs==0.1.8
   ```
6. 准备run.sh文件中所需要的obs文件路径。

a. 准备imagenet数据集的分享链接

勾选要分享的imagenet21k_whole数据集文件夹，单击分享按钮，选择分享链接有效期，自定义提取码，例如123456，单击“复制链接”，记录该链接。

b. 准备obsutil_linux_amd64.tar.gz的分享链接

单击**此处**下载obsutil_linux_amd64.tar.gz，将其上传至OBS桶中，设置为公共读。单击属性，单击复制链接。

链接样例如下：

```
https://${bucketname_name}.obs.cn-north-4.myhuaweicloud.com/${folders_name}/pytorch.tar.gz
```

7. 在Swin-Transformer目录下，创建运行脚本run.sh。

说明

- 脚本中的"SRC_DATA_PATH=${imagenet数据集在obs中分享链接}"，需要替换为上一步中的imagenet21k_whole文件夹分享链接。
- 脚本中的"https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz"，需要替换为上一步中obsutil_linux_amd64.tar.gz在OBS上的路径（需将文件设置为公共读）。

单机单卡运行脚本：

```
# 在代码主目录下创建一个run.sh，内容如下

#!/bin/bash

# 从obs中下载数据到本地SSD盘
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -

IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
MASTER_PORT="6061"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nproc_per_node=1 --master_addr localhost --master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml --local_rank 0
```

多机多卡运行脚本：

```
# 创建run.sh

#!/bin/bash

# 从obs中下载数据到本地SSD盘
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${imagenet数据集在obs中分享链接}
OBSUTIL_PATH=https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz
mkdir -p $DIS_DATA_PATH && cd $DIS_DATA_PATH && wget $OBSUTIL_PATH && tar -xzvf obsutil_linux_amd64.tar.gz && $DIS_DATA_PATH/obsutil_linux_amd64*/obsutil share-cp $SRC_DATA_PATH $DIS_DATA_PATH/ -ac=123456 -r -f -j 256 && cd -
IMAGE_DATA_PATH=$DIS_DATA_PATH/imagenet21k_whole
MASTER_ADDR=$(echo ${VC_WORKER_HOSTS} | cut -d "," -f 1)

MASTER_PORT="6060"
NNODES="$VC_WORKER_NUM"
NODE_RANK="$VC_TASK_INDEX"
NGPUS_PER_NODE="$MA_NUM_GPUS"

/home/ma-user/anaconda3/envs/pytorch/bin/python -m torch.distributed.launch --nnodes=$NNODES --node_rank=$NODE_RANK --nproc_per_node=$NGPUS_PER_NODE --master_addr $MASTER_ADDR --
```

```
master_port=$MASTER_PORT main.py --data-path $IMAGE_DATA_PATH --cfg ./configs/swin/
swin_base_patch4_window7_224_22k.yaml
```

说明

- 推荐先使用单机单卡运行脚本，待正常运行后再改用多机多卡运行脚本。
- 多机多卡run.sh中的“VC_WORKER_HOSTS”、“VC_WORKER_NUM”、“VC_TASK_INDEX”、“MA_NUM_GPUS”为ModelArts训练容器中预置的环境变量。训练容器环境变量详细介绍可参考**查看训练容器环境变量**。

8. 通过obsutils，将代码文件夹放到OBS上，然后通过OBS将代码传至SFS相应目录中。
9. 在SFS中将代码文件Swin-Transformer-main设置归属为ma-user。
   ```
   chown -R ma-user:ma-group Swin-Transformer
   ```
10. 执行以下命令，去除Shell脚本的\r字符。
    ```
    cd Swin-Transformer
    sed -i 's/\r//' run.sh
    ```

说明

Shell脚本在Windows系统编写时，每行结尾是\r\n，而在Linux系统中行每行结尾是\n，所以在Linux系统中运行脚本时，会认为\r是一个字符，导致运行报错“$'\r': command not found”，因此需要去除Shell脚本的\r字符。

10.2.5.3.5 使用 Notebook 进行代码调试

由于Notebook的/cache目录只能支持500G的存储，超过后会导致实例重启，ImageNet数据集大小超过该限制，因此建议用线下资源调试、或用小批量数据集在Notebook调试（Notebook调试方法与**使用Notebook进行代码调试**、**使用Notebook进行代码调试**相同）。

10.2.5.3.6 创建训练任务

1. 登录ModelArts管理控制台，检查当前帐号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“模型训练 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”。
   - 启动方式：选择“自定义”。
   - 镜像：选择上传的自定义镜像。
   - 启动命令：
     ```
     cd /home/ma-user/work/code/Swin-Transformer && /home/ma-user/anaconda3/envs/
     pytorch/bin/pip install -r requirements.txt && /bin/sh run.sh
     ```
   - 资源池：在“专属资源池”页签选择GPU规格的专属资源池。
   - 规格：选择所需GPU规格。
   - 计算节点个数：选择需要的节点个数。
   - SFS Turbo：增加挂载配置，选择SFS名称，云上挂载路径为“/home/ma-user/work”。

     说明

     为了和Notebook调试时代码路径一致，保持相同的启动命令，云上挂载路径需要填写为“/home/ma-user/work”。

4. 单击“提交”，在“信息确认”页面，确认训练作业的参数信息，确认无误后单击“确定”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

10.2.6 FAQ

10.2.6.1 CUDA 和 CUDNN

10.2.6.1.1 Vnt1 机型软件版本建议

gpu driver version : 440.95.01

- gpu driver version : 440.95.01（GPU驱动在宿主机中安装，镜像中无需安装）
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6.x（PyTorch自带，无需关心）
- pytorch version : 1.x.x+cu102

gpu driver version : 470.57.02

- gpu driver version : 470.57.02（GPU驱动在宿主机中安装，镜像中无需安装）
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6（PyTorch自带，无需关心）
- pytorch version : 1.X.X-cu102

gpu driver version : 510.65.01

- gpu driver version :510.65.01
- cuda runtime version : 10.2（PyTorch自带，无需关心）
- cudnn version : 7.6（PyTorch自带，无需关心）
- pytorch version : 1.X.X-cu102

10.2.6.1.2 CUDA Compatibility 如何使用?

当CUDA 10.2与低版本GPU驱动（440.33以下）配合使用时，可能会出现兼容问题，此时需要使用CUDA Compatibility。在创建训练页面添加以下环境变量：

```
export LD_LIBRARY_PATH=/usr/local/cuda/compat
```

说明

训练时默认不需要加此环境变量，仅当发现驱动版本不够时才使用此方法。

10.2.6.1.3 专属池驱动版本如何升级?

当专属资源池中的节点含有GPU/Ascend资源时，用户基于自己的业务，可能会有自定义GPU/Ascend驱动的需求，ModelArts面向此类客户提供了自助升级专属资源池GPU/Ascend驱动的能力，具体操作请参见**资源池驱动升级**。

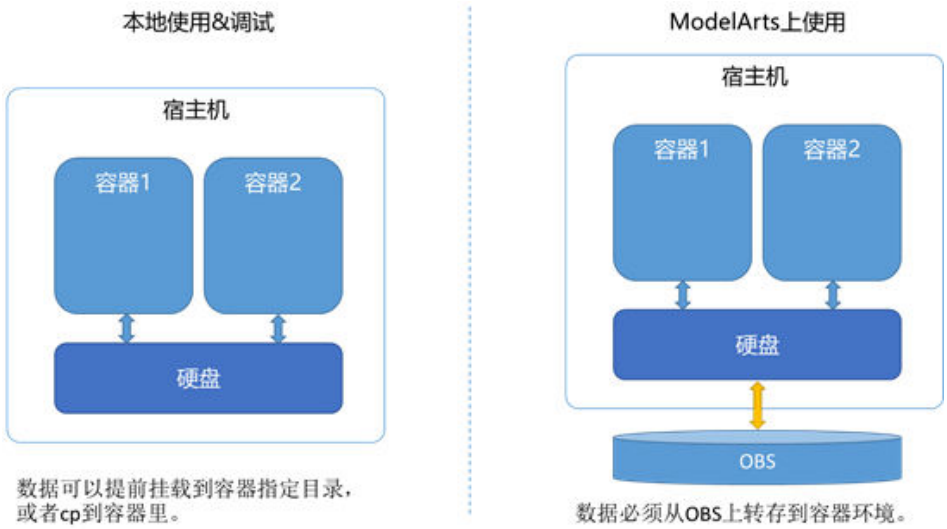
10.2.6.2 CloudShell 调试方法

当使用专属资源池时，允许用户使用ModelArts控制台提供的CloudShell登录运行中的训练容器。CloudShell调试方法请参见**CloudShell使用指导**。

10.2.6.3 run.sh 脚本测试 ModelArts 训练整体流程

自定义容器在ModelArts上训练和本地训练的区别如下图：

图 10-19 本地与 ModelArts 上训练对比



ModelArts上进行训练比本地训练多了一步OBS和容器环境的数据迁移工作。

增加了和OBS交互工作的整个训练流程如下：

说明

建议使用OBSutil作为和OBS交互的工具，如何在本机安装obsutil可以参考**obsutil安装和配置**。

1. 训练数据、代码、模型下载。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）
2. 启动脚本，用法无切换，一般就是到达执行目录，然后python xxx.py。
3. 训练结果、日志、checkpoints上传。（本地使用硬盘挂载或者docker cp，在ModelArts上使用OBSutil）

可以用一个run脚本把整个流程包起来。run.sh脚本的内容可以参考如下示例：

```
#!/bin/bash

##认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
##本示例以AK和SK保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
```

```
##安装obsutil，完成AKSK配置。建议在基础镜像里做好。
#mkdir -p /opt && cd /opt
#wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/obsutil_linux_amd64.tar.gz
#tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils
#alias obsutil='/opt/utils/obsutil'
#obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-
north-4.myhuaweicloud.com

##训练输入复制到容器镜像本地。
#/cache目录的容量较大。

DATA_URL=`echo ${DLS_DATA_URL} | sed /s/s3/obs/`
mkdir –p /cache/data
/opt/utils/obsutil cp –r –f ${DATA_URL} /cache/data

##执行训练任务。
#涉及conda env切换时。
source /xxxxx/etc/profile.d/conda.sh
conda activate xxxenv
conda info --envs
#启动训练脚本。
cd xxxx
python xxx.py

##复制输出结果到OBS目录。
TRAIN_URL=`echo ${DLS_TRAIN_URL} | sed /s/s3/obs/`
/opt/utils/obsutil cp –r –f /cache/out ${TRAIN_URL}
```

把run.sh放到/opt目录，在实际启动任务的时候，使用以下命令启动任务即可：

```
bash –x /opt/run.sh
```

把run.sh放到/root目录，可以在原镜像里增加一层，这一层就只是COPY这个run脚本。在基础镜像里可以一起把obsutil安装、配置好。参考如下dockerfile：

```
FROM $your_docker_image_tag

RUN mkdir -p /opt && cd /opt && \
        wget https://obs-community.obs.cn-north-1.myhuaweicloud.com/obsutil/current/
obsutil_linux_amd64.tar.gz && \
        tar -xzvf obsutil_linux_amd64.tar.gz && mv obsutil_linux_amd64_*/ utils && \
        /opt/utils/obsutil config -i=${HUAWEICLOUD_SDK_AK} -k=${HUAWEICLOUD_SDK_SK} -e=obs.cn-
north-4.myhuaweicloud.com

COPY run.sh /opt/run.sh
```

须知

ModelArts的容器会有一个/cache目录，这个目录挂载的硬盘容量最大。建议下载数据和中间数据都存到这个目录中，防止因硬盘占满导致任务失败。

10.2.6.4 ModelArts 环境挂载目录说明

本小节介绍Notebook开发环境、训练任务实例的目录挂载情况（以下挂载点在保存镜像的时候不会保存）。详情如下：

Notebook

表 10-8 Notebook 挂载点介绍

挂载点	是否只读	备注
/home/ma-user/work/	否	客户数据的持久化目录。
/data	否	客户PFS的挂载目录。
/cache	否	裸机规格时支持，用于挂载宿主机NVMe的硬盘。
/train-worker1-log	否	兼容训练任务调试过程。
/dev/shm	否	用于PyTorch引擎加速。
/modelarts	是	/
/etc/secret-volume	是	/
/etc/sudoers	是	/
/etc/localtime	是	/
var/run/secrets/kubernetes.io/serviceaccount	是	/

训练任务

表 10-9 训练任务挂载点介绍

挂载点	是否只读	备注
/xxx	否	专属池使用SFS盘挂载的目录，路径由客户自己指定。
/home/ma-user/modelarts	否	空文件夹，建议用户主要用这个目录。
/cache	否	裸机规格支持，挂载宿主机NVMe的硬盘。
/dev/shm	否	用于PyTorch引擎加速。
/usr/local/nvidia	是	宿主机的nvidia库。

10.2.6.5 如何查看训练环境变量

在创建训练作业时，“启动命令”输入为“env”，其他参数保持不变。

当训练任务执行完成后，在训练作业详情页面中查看“日志”。日志中即为所有的环境变量信息。

图 10-20 查看日志

```
NV_LIBCUBLAS_DEV_VERSION=11.3.1.68-1
NV_CUDA_COMPAT_PACKAGE=cuda-compat-11-2
NV_CUDNN_PACKAGE_DEV=libcudnn8-dev=8.1.1.33-1+cuda11.2
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64
NV_LIBNCCL_DEV_PACKAGE=libnccl-dev=2.8.4-1+cuda11.2
MA_ENGINE_VERSION=
MA_NUM_HOSTS=1
VC_WORKER_HOSTS=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0.modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
VK_TASK_INDEX=0
_=/usr/bin/env
MA_SCRIPT_INTERPRETER=
NV_LIBNPP_DEV_PACKAGE=libnpp-dev-11-2=11.2.1.68-1
MA_MAX_BACKOFF=0
HOSTNAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47-worker-0
MA_IAM_USER_ID=79098163dd814fd986eca7ef0325d086
MA_CURRENT_IP=10.0.0.62
NV_LIBNPP_VERSION=11.2.1.68-1
NV_NVPROF_DEV_PACKAGE=cuda-nvprof-11-2=11.2.67-1
MA_MOUNT_PATH=/home/ma-user/modelarts
NVIDIA_VISIBLE_DEVICES=all
MA_ENGINE_TYPE=
NV_NVPROF_VERSION=11.2.67-1
NV_LIBCUSPARSE_VERSION=11.3.1.68-1
MODELARTS_SCC_SERVICE_PORT=60687
MA_HOME=/home/ma-user
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.247.0.1
NV_LIBCUBLAS_DEV_PACKAGE=libcublas-dev-11-2=11.3.1.68-1
MA_VJ_NAME=modelarts-job-5f8e4b52-630b-4c15-9bb6-c3f68a48ac47
MA_PIP_URL=http://repo.myhuaweicloud.com/repository/pypi/simple/
NCCL_VERSION=2.8.4-1
KUBERNETES_PORT=tcp://10.247.0.1:443
PWD=/
NVARCH=x86_64
HOME=/home/ma-user
```

10.2.6.6 infiniband 驱动的安装

infiniband 驱动的安装

如果安装了libibverbs-dev库后仍然无法使能infiniband网卡，您可以直接安装infiniband官方驱动，以使用infiniband网卡进行分布式通信，提升训练性能。infiniband驱动需要在制作镜像时安装。

操作步骤

1. 下载MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz。

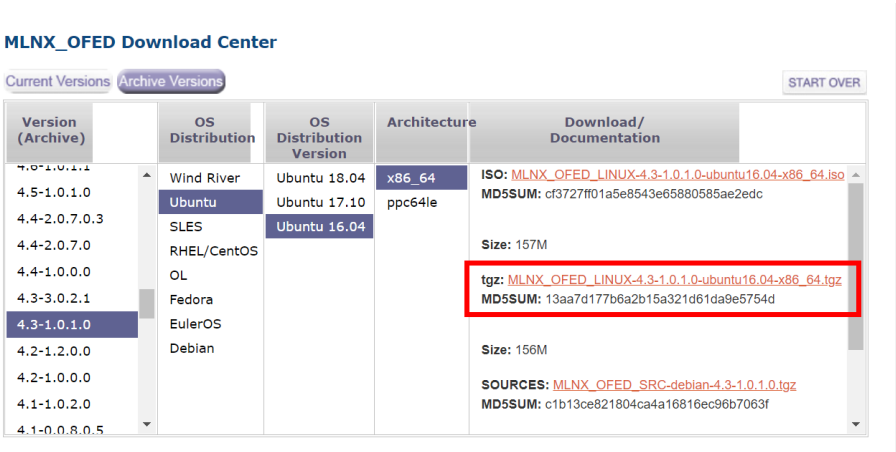
   进入**地址**，单击“Download”，选择“Archive Versions”，“Version”选择“4.3-1.0.1.0”，“OS Distribution”选择“Ubuntu”，“OS Distribution Version”选择“Ubuntu 16.04”，“Architecture”选择“x86_64”，下载MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz。

   说明

   宿主机安装的infiniband驱动版本为4.3-1.0.1.0，容器镜像中安装的infiniband驱动版本需要与宿主机版本匹配，即同为4.3-1.0.1.0。

   可能部分区域的网卡较新，会出现更高版本的infiniband驱动版本，如果您遇到了infiniband驱动安装后，仍然无法使能infiniband网卡的问题，可以咨询相关运维人员以确认宿主机的实际infiniband驱动版本。

图 10-21 下载驱动



2. 参考如下Dockerfile中，以在容器镜像中安装infiniband驱动。

```
USER root

# copy MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz to docker image

RUN tar xzvf MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz && \
    cd MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64 && \
    chmod +x mlnxofedinstall && \
    ./mlnxofedinstall --user-space-only --without-fw-update --force && \
    cd - && \
    rm MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64.tgz && \
    rm -rf MLNX_OFED_LINUX-4.3-1.0.1.0-ubuntu16.04-x86_64

USER ma-user
```

3. 验证infiniband驱动是否安装成功。

   在训练代码中执行以下命令，如果无报错则infiniband驱动安装成功：

```
os.system("ofed_info")
```

10.2.6.7 Tensorboard 的使用

ModelArts支持在开发环境中开启TensorBoard可视化工具。TensorBoard是TensorFlow的可视化工具包，提供机器学习实验所需的可视化功能和工具。

TensorBoard是一个可视化工具，能够有效地展示TensorFlow在运行过程中的计算图、各种指标随着时间的变化趋势以及训练中使用到的数据信息。TensorBoard相关概念请参考**TensorBoard官网**。

TensorBoard可视化训练作业，当前仅支持基于TensorFlow、PyTorch版本镜像，CPU/GPU规格的资源类型。请根据实际局点支持的镜像和资源规格选择使用。

前提条件

为了保证训练结果中输出Summary文件，在编写训练脚本时，您需要在脚本中添加收集Summary相关代码。

TensorFlow引擎的训练脚本中添加Summary代码，具体方式请参见**TensorFlow官方网站**。

注意事项

- 运行中的可视化作业不单独计费，当停止Notebook实例时，计费停止。

- Summary文件数据如果存放在OBS中，由OBS单独收费。任务完成后请及时停止Notebook实例，清理OBS数据，避免产生不必要的费用。

在开发环境中创建 TensorBoard 可视化作业流程

Step1 创建开发环境并在线打开

Step2 上传Summary数据

Step3 启动TensorBoard

Step4 查看训练看板中的可视化数据

Step1 创建开发环境并在线打开

在ModelArts控制台，进入“开发空间 > Notebook”页面，创建TensorFlow或者PyTorch镜像的开发环境实例。创建成功后，单击开发环境实例操作栏右侧的“打开”，在线打开运行中的开发环境。

TensorBoard可视化训练作业，当前仅支持基于TensorFlow、PyTorch镜像，CPU/GPU规格的资源类型。请根据实际局点支持的镜像和资源规格选择使用。

Step2 上传 Summary 数据

在开发环境中使用TensorBoard可视化功能，需要用到Summary数据。

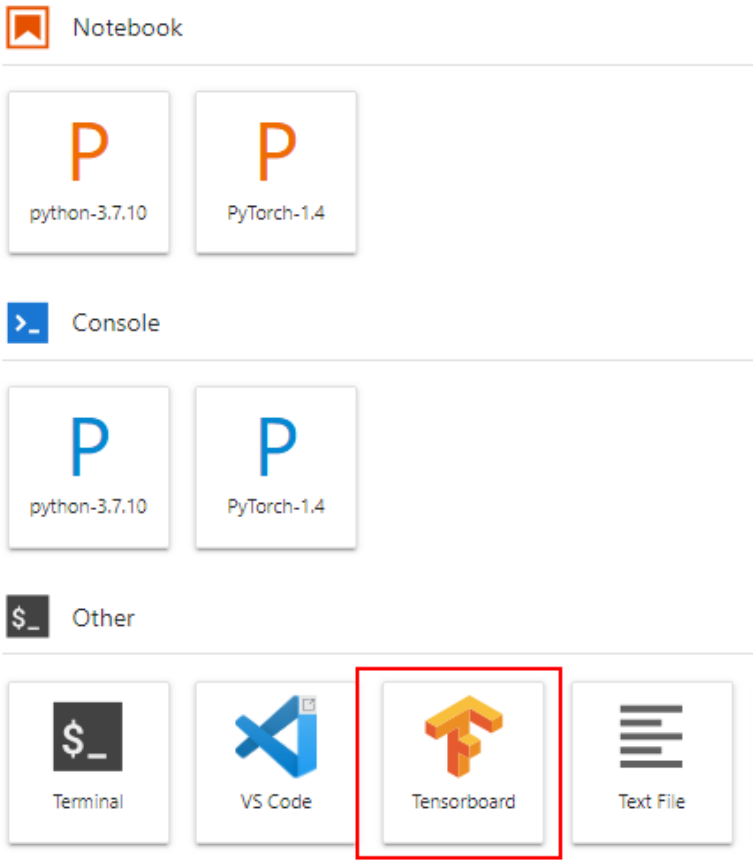
Summary数据可以直接传到开发环境的这个路径下/home/ma-user/work/，也可以放到OBS并行文件系统中。

- Summary数据上传到Notebook路径/home/ma-user/work/下的方式，请参见**上传本地文件至JupyterLab**。
- Summary数据如果是通过OBS并行文件系统挂载到Notebook中，请将模型训练时产生的Summary文件先上传到OBS并行文件系统，并确保OBS并行文件系统与ModelArts在同一区域。在Notebook中启动TensorBoard时，Notebook会自动从挂载的OBS并行文件系统目录中读取Summary数据。

Step3 启动 TensorBoard

在开发环境的JupyterLab中打开TensorBoard。

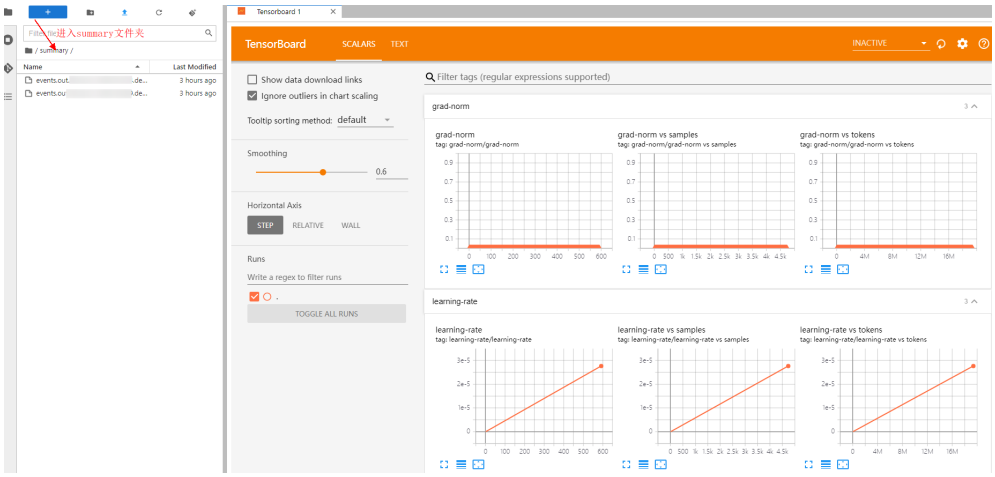
图 10-22 JupyterLab 中打开 TensorBoard



1. 在JupyterLab左侧导航创建名为“summary”的文件夹，将数据上传到“/home/ma-user/work/summary”路径。注：文件夹命名只能为summary否则无法使用。

2. 进入“summary”文件夹，单击方式1 ，直接进入TensorBoard可视化界面。如图10-23所示。

图 10-23 TensorBoard 界面（1）



Step4 查看训练看板中的可视化数据

训练看板是TensorBoard的可视化组件的重要组成部分，而训练看板的标签包含：标量可视化、图像可视化和计算图可视化等。

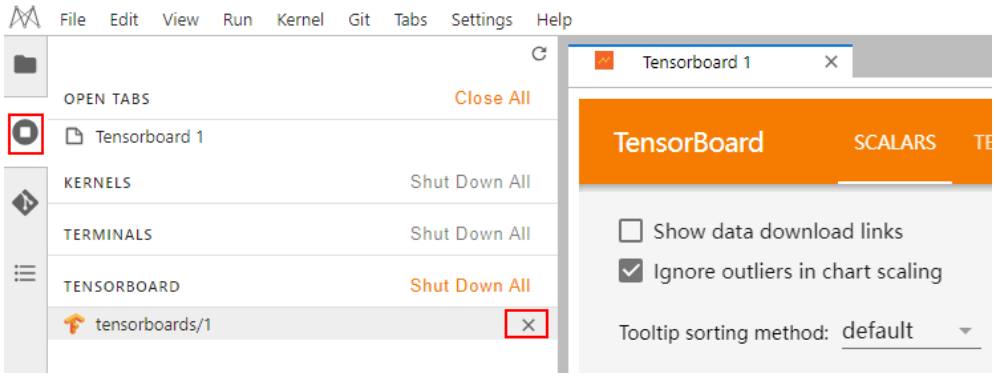
更多功能介绍请参见**TensorBoard官网资料**。

关闭 TensorBoard

关闭TensorBoard方式如下：

单击下图所示的，进入TensorBoard实例管理界面，该界面记录了所有启动的TensorBoard实例，单击对应实例后面的SHUT DOWN即可停止该实例。

图 10-24 单击 SHUT DOWN 停该实例



10.2.6.8 如何保证训练和调试时文件路径保持一致

云上挂载路径

Notebook中挂载SFS后，SFS默认在“/home/ma-user/work”路径下。在创建训练作业时，设置SFS Turbo的“云上挂载路径”为“/home/ma-user/work”，使得训练环境下SFS也在“/home/ma-user/work”路径下。

ln -s 建立软连接

如果代码中涉及文件绝对路径，由于Notebook调试与训练作业环境不同，可能会导致文件绝对路径不一致，需要修改代码内容。推荐使用软链接的方式解决该问题，用户只需提前建立好软链接，代码中的地址可保持不变。

新建软链接：

```
# ln -s 源目录/文件 目标目录/文件
# 例如
ln -s /mnt/sfs_turbo/data/coco /coco
```

删除软链接：

```
# rm 目标目录/文件
rm /coco
```

11 Standard 推理部署

11.1 ModelArts Standard 推理服务访问公网方案

本章节提供了推理服务访问公网的方法。

应用场景

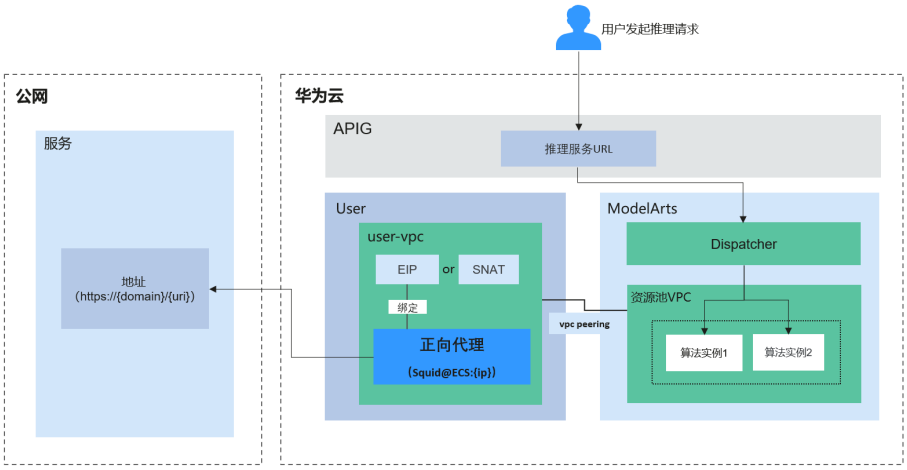
推理服务访问公网地址的场景，如：

- 输入图片，先进行公网OCR服务调用，然后进行NLP处理；
- 进行公网文件下载，然后进行分析；
- 分析结果回调给公网服务终端。

方案设计

从推理服务的算法实例内部，访问公网服务地址的方案。如下图所示：

图 11-1 推理服务访问公网



操作事项

- **ModelArts：设置资源池的网络**

- **用户VPC：安装和配置正向代理**
- **算法镜像：设置DNS代理和公网地址调用**

步骤1 ModelArts：设置资源池的网络

专属资源池的创建作业类型包含推理服务，选择的网络需打通VPC网络，如下图所示：

图 11-2 创建专属资源池

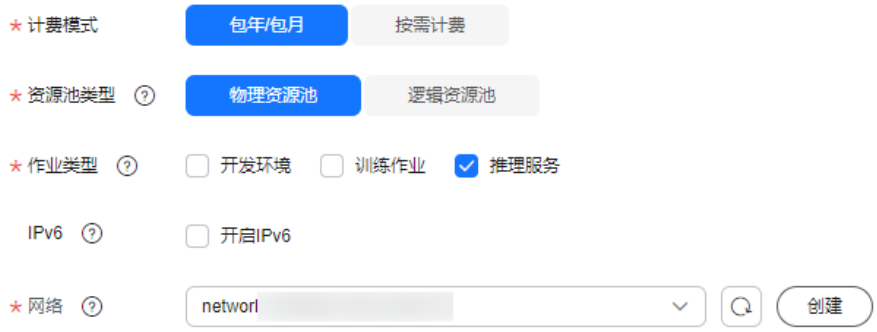


图 11-3 打通 VPC

打通VPC可实现ModelArts资源池和用户VPC的网络打通。打通VPC前需要提前创建好VPC和子网，具体步骤请参考**创建虚拟私有云和子网**。

步骤2 用户VPC：安装和配置正向代理

在安装正向代理前，需要先购买一台弹性云服务器ECS（镜像可选择Ubuntu最新版本），并配置好弹性EIP，然后登录ECS进行正向代理Squid的安装和配置，步骤如下：

1. 如果没有安装Docker，执行以下命令进行Docker安装
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
2. 拉取Squid镜像
   ```
   docker pull ubuntu/squid
   ```
3. 创建主机目录，配置whitelist.conf和squid.conf

   先创建主机目录：
   ```
   mkdir –p /etc/squid/
   ```
   添加whitelist.conf配置文件，内容为安全控制可访问的地址，如：
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   添加squid.conf配置文件，内容如下：
   ```
   # An ACL named 'whitelist'
   acl whitelist dstdomain '/etc/squid/whitelist.conf'

   # Allow whitelisted URLs through
   http_access allow whitelist

   # Block the rest
   http_access deny all
   ```

```
# Default port
http_port 3128
```

然后设置主机目录和配置文件权限如下：

```
chmod 640 -R /etc/squid
```

4. 启动squid实例

```
docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
```

5. 如果whitelist.conf或squid.conf有更新，则进入容器刷新squid

```
docker exec –it squid bash
root@{container_id}:/# squid -k reconfigure
```

步骤3 算法镜像：设置DNS代理和公网地址调用

1. 设置代理

   在代码中设置代理指向代理服务器私有IP和端口，如下所示：

```
proxies = {
  "http": "http://{proxy_server_private_ip}:3128",
  "https": "http://{proxy_server_private_ip}:3128"
}
```

   服务器私有IP获取如下图所示：

   图 11-4 ECS 私有 IP

2. 地址调用

   在推理代码中，使用服务URL进行业务请求，如：

```
https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
```

----结束

11.2 端到端运维 ModelArts Standard 推理服务方案

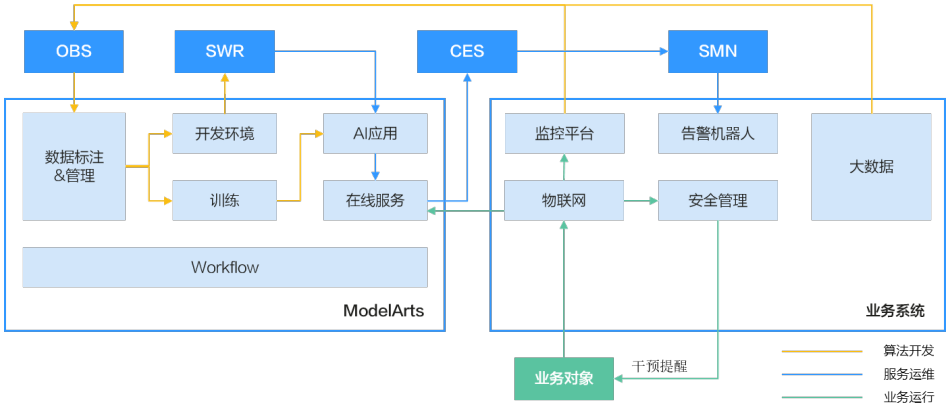
ModelArts推理服务的端到端运维覆盖了算法开发、服务运维和业务运行的整个AI流程。

方案概述

推理服务的端到端运维流程

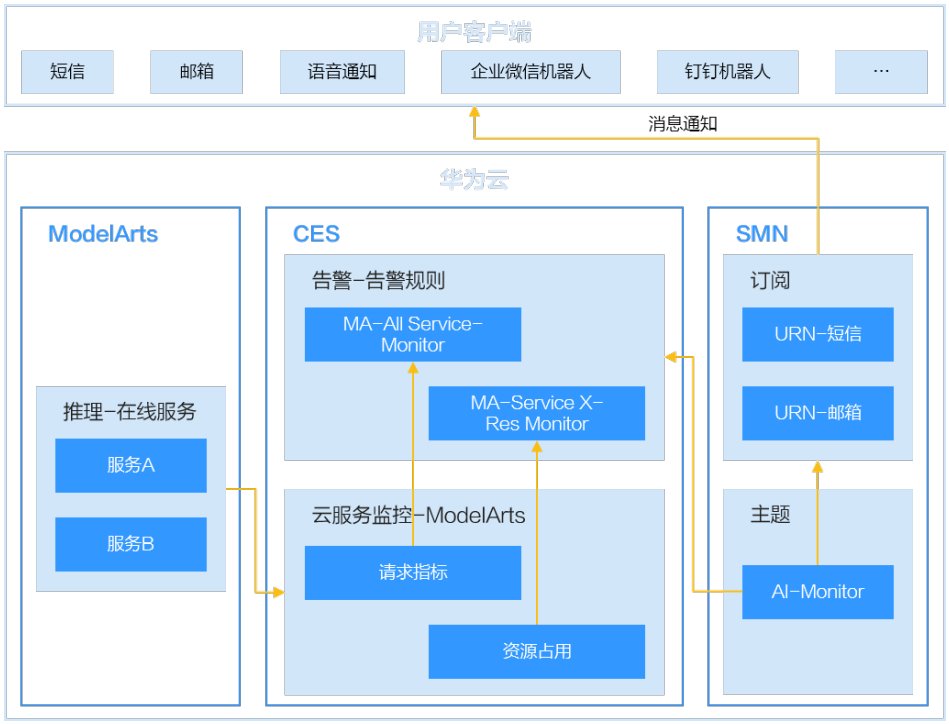
- 算法开发阶段，先将业务AI数据存放到对象存储服务（OBS）中，接着通过ModelArts数据管理进行标注和版本管理，然后通过训练获得AI模型结果，最后通过开发环境构建AI应用镜像。
- 服务运维阶段，先利用镜像构建AI应用，接着部署AI应用为在线服务，然后可在云监控服务（CES）中获得ModelArts推理在线服务的监控数据，最后可配置告警规则实现实时告警通知。
- 业务运行阶段，先将业务系统对接在线服务请求，然后进行业务逻辑处理和监控设置。

图 11-5 推理服务的端到端运维流程图



整个运维过程会对服务请求失败和资源占用过高的场景进行监控，当超过阈值时发送告警通知。

图 11-6 监控告警流程图



方案优势

通过端到端的服务运维配置，可方便地查看业务运行高低峰情况，并能够实时感知在线服务的健康状态。

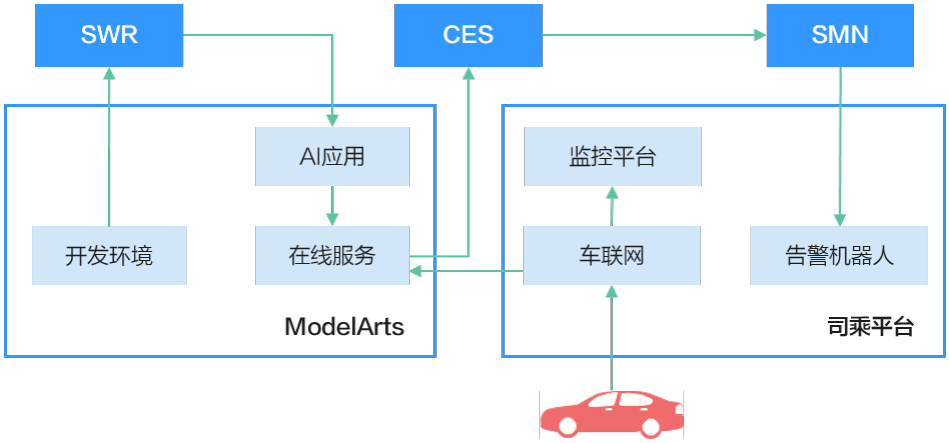
约束限制

端到端服务运维只支持在线服务，因为推理的批量服务和边缘服务无CES监控数据，不支持完整的端到端服务运维设置。

实施步骤

以出行场景的司乘安全算法为例，介绍使用ModelArts进行流程化服务部署和更新、自动化服务运维和监控的实现步骤。

图 11-7 司乘安全算法



步骤1 将用户本地开发完成的模型，使用自定义镜像在ModelArts构建成AI应用。具体操作请参考**从0-1制作自定义镜像并创建AI应用**。

步骤2 在ModelArts管理控制台，使用创建好的AI应用部署为在线服务。

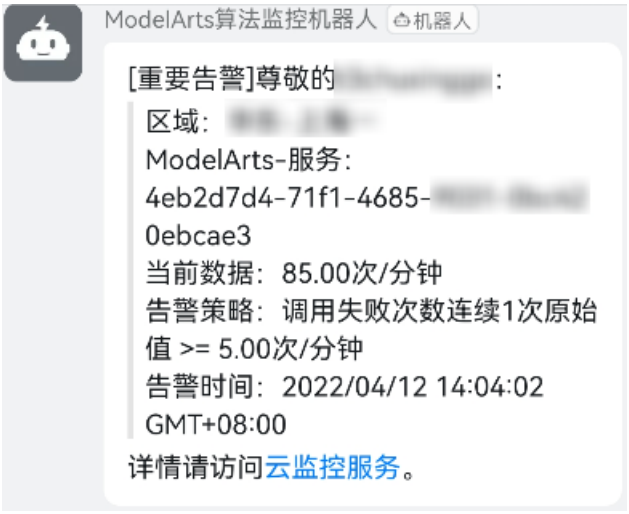
步骤3 登录云监控服务CES管理控制台，设置ModelArts服务的告警规则并配置主题订阅方式发送通知。具体操作请参考**设置告警规则**。

当配置完成后，在左侧导航栏选择“云服务监控 > ModelArts”即可查看在线服务的请求情况和资源占用情况，如下图所示。

图 11-8 查看服务的监控指标

当监控信息触发告警时，主题订阅对象将会收到消息通知。

图 11-9 告警消息通知



----结束

11.3 使用自定义引擎在 ModelArts Standard 创建 AI 应用

使用自定义引擎创建AI应用，用户可以通过选择自己存储在SWR服务中的镜像作为AI应用的引擎，指定预先存储于OBS服务中的文件目录路径作为模型包来创建AI应用，轻松地应对ModelArts平台预置引擎无法满足个性化诉求的场景。

自定义引擎创建 AI 应用的规范

使用自定义引擎创建AI应用，用户的SWR镜像、OBS模型包和文件大小需要满足以下规范：

- SWR镜像规范：
  - 镜像必须内置一个用户名为“ma-user”，组名为“ma-group”的普通用户，且必须确保该用户的uid=1000、gid=100。内置用户的dockerfile指令如下：

    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```

  - 明确设置镜像的启动命令。在dockerfile文件中指定cmd，dockerfile指令示例如下：

    ```
    CMD sh /home/mind/run.sh
    ```

    启动入口文件run.sh需要自定义。示例如下：

    ```
    #!/bin/bash

    # 自定义脚本内容
    ...

    # run.sh调用app.py启动服务器，app.py请参考https示例
    python app.py
    ```

    说明

    除了按上述要求设置启动命令，您也可以在镜像中自定义启动命令，在创建AI应用时填写与您镜像中相同的启动命令。

  - 提供的服务可使用HTTPS/HTTP协议和监听的容器端口，端口和协议可根据镜像实际使用情况自行填写，ModelArts提供的请求协议和端口号的缺省值是HTTPS和8080。请参考**https示例**。
  - （可选）健康检查的URL路径必须为"/health"。
- OBS模型包规范

  模型包的名字必须为model。模型包规范请参见**模型包规范介绍**。
- 文件大小规范

  当使用公共资源池时，SWR的镜像大小（指下载后的镜像大小，非SWR界面显示的压缩后的镜像大小）和OBS模型包大小总和不大于30G。

https 示例

使用Flask启动https，Webserver代码示例如下：

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("----------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

在本地机器调试

自定义引擎的规范可以在安装有docker的本地机器上通过以下步骤提前验证：

1. 将自定义引擎镜像下载至本地机器，假设镜像名为custom_engine:v1。
2. 将模型包文件夹复制到本地机器，假设模型包文件夹名字为model。
3. 在模型包文件夹的同级目录下验证如下命令拉起服务：

   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   说明

   该指令无法完全模拟线上，主要是由于-v挂载进去的目录是root权限。在线上，模型文件从OBS下载到/home/mind/model目录之后，文件owner将统一修改为ma-user。

4. 在本地机器上启动另一个终端，执行以下验证指令，得到符合预期的推理结果。
```
curl https://127.0.0.1:8080/${推理服务的请求路径}
```

推理部署示例

本节将详细说明以自定义引擎方式创建AI应用的步骤。

1. 创建AI应用并查看AI应用详情

   登录ModelArts管理控制台，进入“AI应用”页面中，单击“创建应用”，进入AI应用创建页面，设置相关参数如下：

   - 元模型来源：选择“从对象存储服务（OBS）中选择”。
   - 选择元模型：从OBS中选择一个模型包。
   - AI引擎：选择“Custom”。
   - 引擎包：从容器镜像中选择一个镜像。
   - 容器调用接口：端口和协议可根据镜像实际使用情况自行填写。

     其他参数保持默认值。

   单击“立即创建”，跳转到AI应用列表页，查看AI应用状态，当状态变为“正常”，AI应用创建成功。

   图 11-10 创建 AI 应用

   单击AI应用名称，进入AI应用详情页面，查看AI应用详情信息。

2. 部署服务并查看详情

   在AI应用详情页面，单击右上角“部署>在线服务”，进入服务部署页面，AI应用和版本默认选中，选择合适的“计算节点规格”（例如CPU：2核 8GB），其他参数可保持默认值，单击“下一步”，跳转至服务列表页，当服务状态变为“运行中”，服务部署成功。

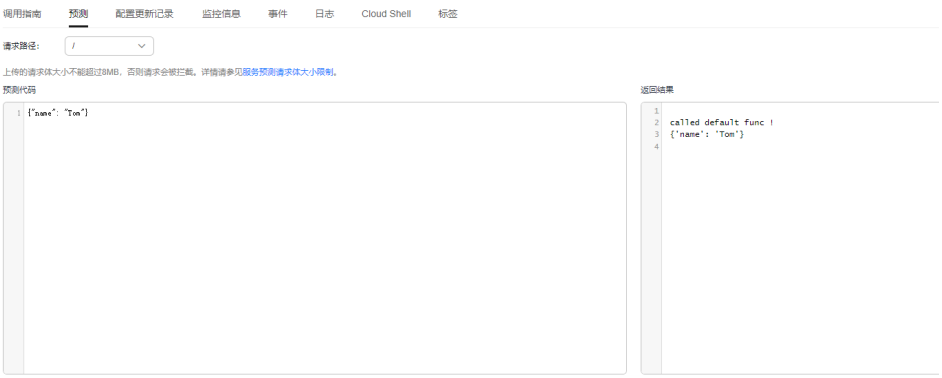
   单击服务名称，进入服务详情页面，查看服务详情信息，单击“日志”页签，查看服务日志信息。

图 11-11 查看服务日志信息

3. 服务预测

   在服务详情页面，单击“预测”页签，进行服务预测。

图 11-12 服务预测



11.4 使用大模型在 ModelArts Standard 创建 AI 应用部署在线服务

背景说明

目前大模型的参数量已经达到千亿甚至万亿，随之大模型的体积也越来越大。千亿参数大模型的体积超过200G，在版本管理、生产部署上对平台系统产生了新的要求。例如：导入AI应用管理时，需要支持动态调整租户存储配额；模型加载、启动慢，部署时需要灵活的超时配置；当负载异常重启，模型需要重新加载，服务恢复时间长的问题亟待解决。

为了应对如上诉求，ModelArts推理平台针对性给出解决方案，用于支持大模型场景下的AI应用管理和服务部署。

约束与限制

- 需要申请单个AI应用大小配额和添加使用节点本地存储缓存的白名单。
- 需要使用自定义引擎Custom，配置动态加载。
- 需要使用专属资源池部署服务。
- 专属资源池磁盘空间需大于1T。

操作事项

1. 申请扩大AI应用的大小配额和使用节点本地存储缓存白名单
2. 上传模型数据并校验上传对象的一致性
3. 创建专属资源池
4. 创建AI应用
5. 部署在线服务

申请扩大 AI 应用的大小配额和使用节点本地存储缓存白名单

服务部署时，默认情况下，动态加载的模型包位于临时磁盘空间，服务停止时已加载的文件会被删除，再次启动时需要重新加载。为了避免反复加载，平台允许使用资源池节点的本地存储空间来加载模型包，并在服务停止和重启时仍有效（通过哈希值保证数据一致性）

使用大模型要求用户采用自定义引擎，并开启动态加载的模式导入模型。基于此，需要执行以下操作：

- 如果模型超过默认配额值，需要提工单申请扩大单个AI应用的大小配额。单个AI应用大小配额默认值为20GB。
- 需要提工单申请添加使用节点本地存储缓存的白名单。

上传模型数据并校验上传对象的一致性

为了动态加载时保证数据完整性，需要在上传模型数据至OBS时，进行上传对象的一致性校验。obsutil、OBS Browser+以及OBS SDK都支持在上传对象时进行一致性校验，您可以根据自己的业务选择任意一种方式进行校验。详见**校验上传对象的一致性**。

以OBS Browser+为例，如**图11-13**。使用OBS Browser+上传数据，开启MD5校验，动态加载并使用节点本地的持久化存储时，检查数据一致性。

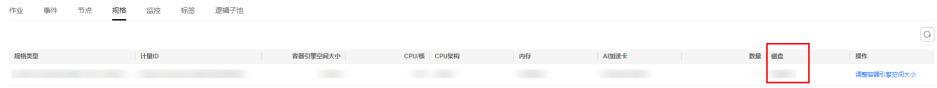
图 11-13 OBS Browser+配置 MD5 校验

创建专属资源池

使用本地的持久化存储功能，需使用专属资源池，且专属资源池磁盘空间大小必须超过1T。您可以通过专属资源池详情页面，规格页签，查看专属资源池磁盘信息。当服

务部署失败，提示磁盘空间不足时，请参考**服务部署、启动、升级和修改时，资源不足如何处理?**

图 11-14 查看专属资源池磁盘信息



创建 AI 应用

使用大模型创建AI应用，选择从对象存储服务（OBS）中导入，需满足以下参数配置：

1. 采用自定义引擎，开启动态加载

   使用大模型要求用户使用自定义引擎，并开启动态加载的模式导入模型。用户可以制作自定义引擎，满足大模型场景下对镜像依赖包、推理框架等的特殊需求。自定义引擎的制作请参考**使用自定义引擎在ModelArts Standard创建AI应用**。

   当用户使用自定义引擎时，默认开启动态加载，模型包与镜像分离，在服务部署时动态将模型加载到服务负载。

2. 配置健康检查

   大模型场景下导入的AI应用，要求配置健康检查，避免在部署时服务显示已启动但实际不可用。

图 11-15 采用自定义引擎，开启动态加载并配置健康检查示例图

部署在线服务

部署服务时，需满足以下参数配置：

1. 自定义部署超时时间

   大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

2. 添加环境变量

   部署服务时，增加如下环境变量，会将负载均衡的请求亲和策略配置为集群亲和，避免未就绪的服务实例影响预测成功率。

```
MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
```

图 11-16 自定义部署超时时间和添加环境变量示例图

建议部署多实例，增加服务可靠性。

11.5 第三方推理框架迁移到 ModelArts Standard 推理自定义引擎

背景说明

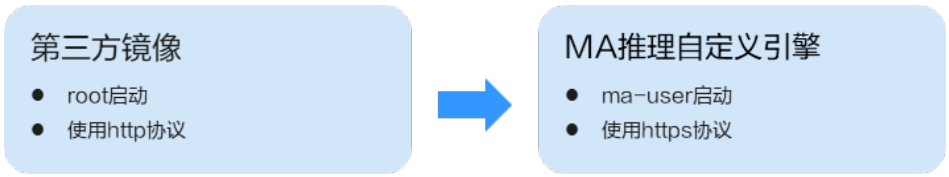
ModelArts支持第三方的推理框架在ModelArts上部署，本文以TFServing框架、Triton框架为例，介绍如何迁移到推理自定义引擎。

- TensorFlow Serving是一个灵活、高性能的机器学习模型部署系统，提供模型版本管理、服务回滚等能力。通过配置模型路径、模型端口、模型名称等参数，原生TFServing镜像可以快速启动提供服务，并支持gRPC和HTTP Restful API的访问方式。
- Triton是一个高性能推理服务框架，提供HTTP/gRPC等多种服务协议，支持TensorFlow、TensorRT、PyTorch、ONNXRuntime等多种推理引擎后端，并且支持多模型并发、动态batch等功能，能够提高GPU的使用率，改善推理服务的性能。

当从第三方推理框架迁移到使用ModelArts推理的AI应用管理和服务管理时，需要对原生第三方推理框架镜像的构建方式做一定的改造，以使用ModelArts推理平台的模型版本管理能力和动态加载模型的部署能力。本案例将指导用户完成原生第三方推理框架镜像到ModelArts推理自定义引擎的改造。自定义引擎的镜像制作完成后，即可以通过AI应用导入对模型版本进行管理，并基于AI应用进行部署和管理服务。

适配和改造的主要工作项如下：

图 11-17 改造工作项



针对不同框架的镜像，可能还需要做额外的适配工作，具体差异请见对应框架的操作步骤。

- TFServing框架迁移操作步骤
- Triton框架迁移操作步骤

TFServing 框架迁移操作步骤

步骤1 增加用户ma-user。

基于原生"tensorflow/serving:2.8.0"镜像构建，镜像中100的用户组默认已存在，Dockerfile中执行如下命令增加用户ma-user。

```
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

协议转换为https之后，对外暴露的端口从tfserving的8501变为8080。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
```



```
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8501;
    }
}
```

5. 准备启动脚本。

说明

启动前先创建ssl证书，然后启动TFServing的启动脚本。

启动脚本run.sh示例代码如下：

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText
sh /usr/bin/tf_serving_entrypoint.sh
```

步骤3 修改模型默认路径，支持ModelArts推理模型动态加载。

Dockerfile中执行如下命令修改默认的模型路径。

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----结束

完整的Dockerfile参考：

```
FROM tensorflow/serving:2.8.0
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean
RUN mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
ADD nginx /etc/nginx
ADD run.sh /home/mind/
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
ENTRYPOINT []
CMD /bin/bash /home/mind/run.sh
```

Triton 框架迁移操作步骤

本教程基于nvidia官方提供的nvcr.io/nvidia/tritonserver:23.03-py3镜像进行适配，使用开源大模型llama7b进行推理任务。

步骤1 增加用户ma-user。

Triton镜像中默认已存在id为1000的triton-server用户，需先修改triton-server用户名id后再增加用户ma-user，Dockerfile中执行如下命令。

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

步骤2 通过增加nginx代理，支持https协议。

1. Dockerfile中执行如下命令完成nginx的安装和配置。

```
RUN apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
```



```
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx
```

2. 准备nginx目录如下：

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. 准备nginx.conf文件内容如下：

```
user ma-user 100;
worker_processes 2;
pid /home/ma-user/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 768;
}
http {
    ##
    # Basic Settings
    ##
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    types_hash_max_size 2048;
    fastcgi_hide_header X-Powered-By;
    port_in_redirect off;
    server_tokens off;
    client_body_timeout 65s;
    client_header_timeout 65s;
    keepalive_timeout 65s;
    send_timeout 65s;
    # server_names_hash_bucket_size 64;
    # server_name_in_redirect off;
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    ##
    # Logging Settings
    ##
    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;
    ##
    # Gzip Settings
    ##
    gzip on;
    ##
    # Virtual Host Configs
    ##
    include /etc/nginx/conf.d/modelarts-model-server.conf;
}
```

4. 准备modelarts-model-server.conf配置文件内容如下：

```
server {
    client_max_body_size 15M;
    large_client_header_buffers 4 64k;
    client_header_buffer_size 1k;
    client_body_buffer_size 16k;
    ssl_certificate /etc/nginx/ssl/server/server.crt;
    ssl_password_file /etc/nginx/keys/fifo;
    ssl_certificate_key /etc/nginx/ssl/server/server.key;
    # setting for mutual ssl with client
    ##
    # header Settings
```

```
    ##
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header Strict-Transport-Security "max-age=31536000; includeSubdomains;";
    add_header Content-Security-Policy "default-src 'self'";
    add_header Cache-Control "max-age=0, no-cache, no-store, must-revalidate";
    add_header Pragma "no-cache";
    add_header Expires "-1";
    server_tokens off;
    port_in_redirect off;
    fastcgi_hide_header X-Powered-By;
    ssl_session_timeout 2m;
    ##
    # SSL Settings
    ##
    ssl_protocols TLSv1.2;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256;
    listen    0.0.0.0:8080 ssl;
    error_page 502 503 /503.html;
    location /503.html {
        return 503 '{"error_code": "ModelArts.4503","error_msg": "Failed to connect to backend service,
please confirm your service is connectable. "}';
    }
    location / {
#       limit_req zone=mylimit;
#       limit_req_status 429;
        proxy_pass http://127.0.0.1:8000;
    }
}
```

5. 准备启动脚本run.sh。

   说明

   启动前先创建ssl证书，然后启动Triton的启动脚本。

```
#!/bin/bash
mkdir -p /etc/nginx/ssl/server && cd /etc/nginx/ssl/server
cipherText=$(openssl rand -base64 32)
openssl genrsa -aes256 -passout pass:"${cipherText}" -out server.key 2048
openssl rsa -in server.key -passin pass:"${cipherText}" -pubout -out rsa_public.key
openssl req -new -key server.key -passin pass:"${cipherText}" -out server.csr -subj "/C=CN/ST=GD/L=SZ/
O=Huawei/OU=ops/CN=*.huawei.com"
openssl genrsa -out ca.key 2048
openssl req -new -x509 -days 3650 -key ca.key -out ca-crt.pem -subj "/C=CN/ST=GD/L=SZ/O=Huawei/
OU=dev/CN=ca"
openssl x509 -req -days 3650 -in server.csr -CA ca-crt.pem -CAkey ca.key -CAcreateserial -out server.crt
service nginx start &
echo ${cipherText} > /etc/nginx/keys/fifo
unset cipherText

bash /home/mind/model/triton_serving.sh
```

步骤3 编译安装tensorrtllm_backend。

1. Dockerfile中执行如下命令获取tensorrtllm_backend源码，安装tensorrt、cmake和pytorch等相关依赖，并进行编译安装。

```
# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt
```



```
# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver
```

2. 准备triton serving的启动脚本triton_serving.sh，llama模型的参考样例如下：

```
MODEL_NAME=llama_7b
MODEL_DIR=/home/mind/model/${MODEL_NAME}
OUTPUT_DIR=/tmp/llama/7B/trt_engines/fp16/1-gpu/
MAX_BATCH_SIZE=1
export LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH}

# build tensorrt_llm engine
cd /opt/tritonserver/tensorrtllm_backend/tensorrt_llm/examples/llama
python build.py --model_dir ${MODEL_DIR} \
                --dtype float16 \
                --remove_input_padding \
                --use_gpt_attention_plugin float16 \
                --enable_context_fmha \
                --use_weight_only \
                --use_gemm_plugin float16 \
                --output_dir ${OUTPUT_DIR} \
                --paged_kv_cache \
                --max_batch_size ${MAX_BATCH_SIZE}

# set config parameters
cd /opt/tritonserver/tensorrtllm_backend
mkdir triton_model_repo
cp all_models/inflight_batcher_llm/* triton_model_repo/ -r

python3 tools/fill_template.py -i triton_model_repo/preprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},preprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/postprocessing/config.pbtxt tokenizer_dir:$
{MODEL_DIR},tokenizer_type:llama,triton_max_batch_size:$
{MAX_BATCH_SIZE},postprocessing_instance_count:1
python3 tools/fill_template.py -i triton_model_repo/ensemble/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE}
python3 tools/fill_template.py -i triton_model_repo/tensorrt_llm/config.pbtxt triton_max_batch_size:$
{MAX_BATCH_SIZE},decoupled_mode:False,max_beam_width:1,engine_dir:$
{OUTPUT_DIR},max_tokens_in_paged_kv_cache:2560,max_attention_window_size:2560,kv_cache_free_
gpu_mem_fraction:0.5,exclude_input_in_output:True,enable_kv_cache_reuse:False,batching_strategy:V1,
max_queue_delay_microseconds:600

# launch tritonserver
python3 scripts/launch_triton_server.py --world_size 1 --model_repo=triton_model_repo/
while true; do sleep 10000; done
```

部分参数说明：

- MODEL_NAME：HuggingFace格式模型权重文件所在OBS文件夹名称。
- OUTPUT_DIR：通过TensorRT-LLM转换后的模型文件在容器中的路径。

完整的Dockerfile如下：

```
FROM nvcr.io/nvidia/tritonserver:23.03-py3

# add ma-user and install nginx
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash
```



```
ma-user && \
    apt-get update && apt-get -y --no-install-recommends install nginx && apt-get clean && \
    mkdir /home/mind && \
    mkdir -p /etc/nginx/keys && \
    mkfifo /etc/nginx/keys/fifo && \
    chown -R ma-user:100 /home/mind && \
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# get tensortllm_backend source code
WORKDIR /opt/tritonserver
RUN apt-get install -y --no-install-recommends rapidjson-dev python-is-python3 git-lfs && \
    git config --global http.sslVerify false && \
    git config --global http.postBuffer 1048576000 && \
    git clone -b v0.5.0 https://github.com/triton-inference-server/tensorrtllm_backend.git --depth 1 && \
    cd tensorrtllm_backend && git lfs install && \
    git config submodule.tensorrt_llm.url https://github.com/NVIDIA/TensorRT-LLM.git && \
    git submodule update --init --recursive --depth 1 && \
    pip3 install -r requirements.txt

# build tensorrtllm_backend
WORKDIR /opt/tritonserver/tensorrtllm_backend/tensorrt_llm
RUN sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_tensorrt.sh && \
    bash docker/common/install_tensorrt.sh && \
    export  LD_LIBRARY_PATH=/usr/local/tensorrt/lib:${LD_LIBRARY_PATH} && \
    sed -i "s/wget/wget --no-check-certificate/g" docker/common/install_cmake.sh && \
    bash docker/common/install_cmake.sh && \
    export PATH=/usr/local/cmake/bin:$PATH && \
    bash docker/common/install_pytorch.sh pypi && \
    python3 ./scripts/build_wheel.py --trt_root /usr/local/tensorrt && \
    pip install ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    rm -f ./build/tensorrt_llm-0.5.0-py3-none-any.whl && \
    cd ../inflight_batcher_llm && bash scripts/build.sh && \
    mkdir /opt/tritonserver/backends/tensorrtllm && \
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

完成镜像构建后，将镜像注册至华为云容器镜像服务SWR中，用于后续在ModelArts上部署推理服务。

步骤4 使用适配后的镜像在ModelArts部署在线推理服务。

1. 在obs中创建model目录，并将triton_serving.sh文件和llama_7b文件夹上传至model目录下，如下图所示。

图 11-18 上传至 model 目录

2. 创建AI应用，源模型来源选择“从对象存储服务（OBS）中选择”，元模型选择至model目录，AI引擎选择Custom，引擎包选择步骤3构建的镜像。

图 11-19 创建 AI 应用

3. 将创建的AI应用部署为在线服务，大模型加载启动的时间一般大于普通的模型创建的服务，请配置合理的“部署超时时间”，避免尚未启动完成被认为超时而导致部署失败。

图 11-20 部署为在线服务

4. 调用在线服务进行大模型推理，请求路径填写/v2/models/ensemble/infer，调用样例如下：

```
{
  "inputs": [
    {
      "name": "text_input",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": ["what is machine learning"]
    },
    {
      "name": "max_tokens",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [64]
    },
    {
      "name": "bad_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "stop_words",
      "shape": [1, 1],
      "datatype": "BYTES",
      "data": [""]
    },
    {
      "name": "pad_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    },
    {
      "name": "end_id",
      "shape": [1, 1],
      "datatype": "UINT32",
      "data": [2]
    }
  ],
  "outputs": [
    {
      "name": "text_output"
    }
  ]
}
```

说明

- "inputs"中"name"为"text_input"的元素代表输入，"data"为具体输入语句，本示例中为"what is machine learning"。
- "inputs"中"name"为"max_tokens"的元素代表输出最大tokens数，"data"为具体数值，本示例中为64。

图 11-21 调用在线服务

----结束

11.6 ModelArts Standard 推理服务支持 VPC 直连的高速访问通道配置

背景说明

访问在线服务的实际业务中，用户可能会存在如下需求：

- 高吞吐量、低时延
- TCP或者RPC请求

因此，ModelArts提供了VPC直连的高速访问通道功能以满足用户的需求。

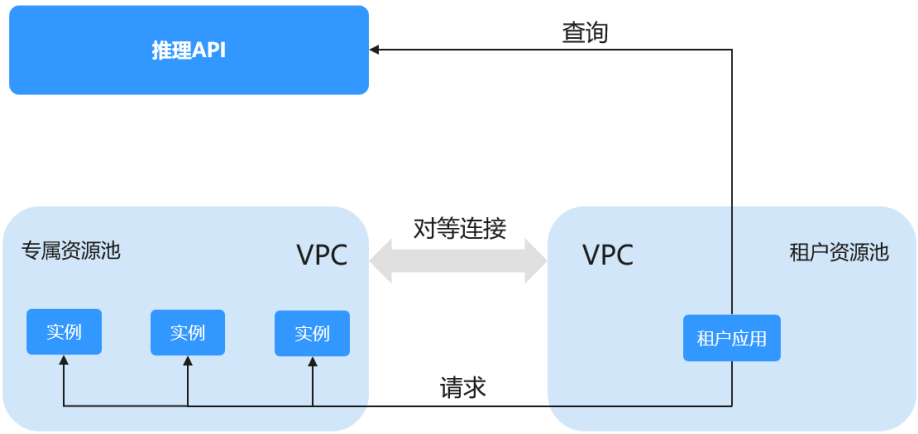
使用VPC直连的高速访问通道，用户的业务请求不需要经过推理平台，而是直接经VPC对等连接发送到实例处理，访问速度更快。

说明

由于请求不经过推理平台，所以会丢失以下功能：

- 认证鉴权
- 流量按配置分发
- 负载均衡
- 告警、监控和统计

图 11-22 VPC 直连的高速访问通道示意图



准备工作

使用专属资源池部署在线服务，服务状态为“运行中”。

须知

- 需使用新版专属资源池部署服务，详情请参见**ModelArts资源池管理功能全面升级**。
- 只有专属资源池部署的服务才支持VPC直连的高速访问通道。
- VPC直连的高速访问通道，目前只支持访问在线服务。
- 因流量限控，获取在线服务的IP和端口号次数有限制，每个主账号租户调用次数不超过2000次/分钟，每个子账号租户不超过20次/分钟。
- 目前仅支持自定义镜像导入模型，部署的服务支持高速访问通道。

操作步骤

使用VPC直连的高速访问通道访问在线服务，基本操作步骤如下：

1. **将专属资源池的网络打通VPC**
2. **VPC下创建弹性云服务器**
3. **获取在线服务的IP和端口号**
4. **通过IP和端口号直连应用**

步骤1 将专属资源池的网络打通VPC

登录ModelArts控制台，进入“AI专属资源池 > 弹性集群 Cluster”找到服务部署使用的专属资源池，单击“名称/ID”，进入资源池详情页面，查看网络配置信息。返回专属资源池列表，选择“网络”页签，找到专属资源池关联的网络，打通VPC。打通VPC网络后，网络列表和资源池详情页面将显示VPC名称，单击后可以跳转至VPC详情页面。

图 11-23 查看网络配置

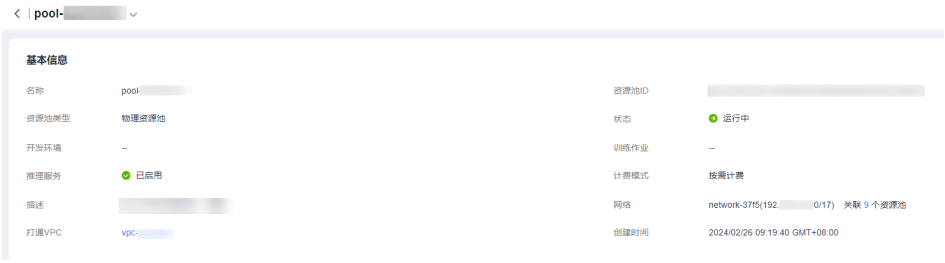


图 11-24 打通 VPC

步骤2 VPC下创建弹性云服务器

登录弹性云服务器ECS控制台，单击右上角“购买弹性云服务器”，进入购买弹性云服务器页面，完成基本配置后单击“下一步：网络配置”，进入网络配置页面，选择**步骤1**中打通的VPC，完成其他参数配置，完成高级配置并确认配置，下发购买弹性云服务器的任务。等待服务器的状态变为“运行中”时，弹性云服务器创建成功。单击“名称/ID”，进入服务器详情页面，查看虚拟私有云配置信息。

图 11-25 购买弹性云服务器时选择 VPC

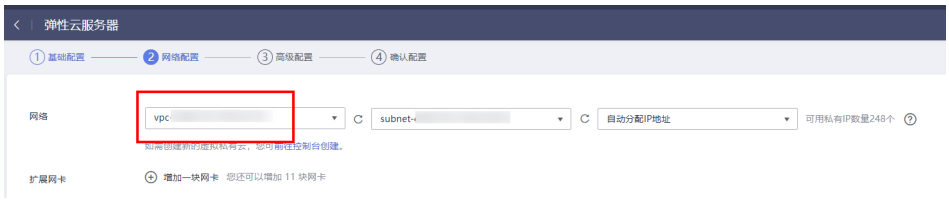
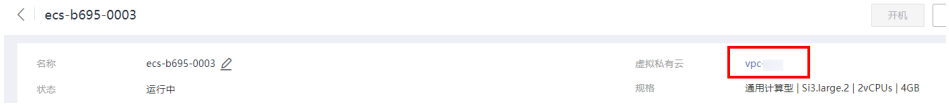


图 11-26 查看虚拟私有云配置信息



步骤3 获取在线服务的IP和端口号

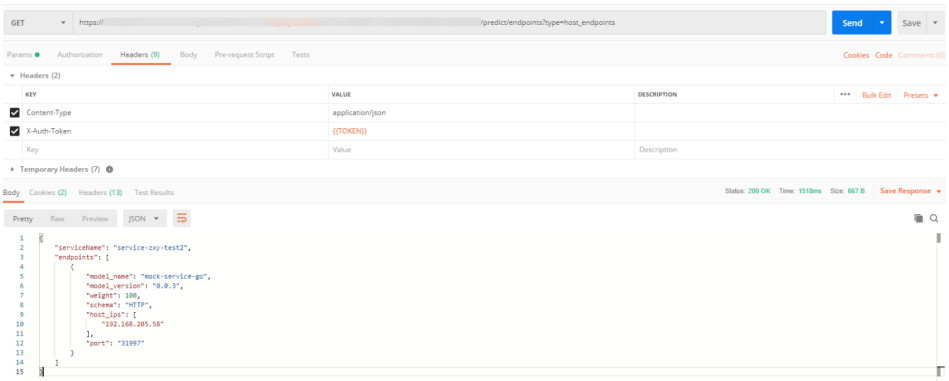
可以通过使用图形界面的软件（以Postman为例）获取服务的IP和端口号，也可以登录弹性云服务器（ECS），创建Python环境运行代码，获取服务IP和端口号。

API接口：

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- 方式一：图形界面的软件获取服务的IP和端口号

图 11-27 接口返回示例



- 方式二：Python语言获取IP和端口号

  Python代码如下，下述代码中以下参数需要手动修改：

  - project_id：用户项目ID，获取方法请参见**获取项目ID和名称**。
  - service_id：服务ID，在服务详情页可查看。
  - REGION_ENDPOINT：服务的终端节点，查询请参见**终端节点**。

```
def get_app_info(project_id, service_id):
    list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
    headers = {'X-Auth-Token': X_Auth_Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

步骤4 通过IP和端口号直连应用

登录弹性云服务器（ECS），可以通过Linux命令行访问在线服务，也可以创建Python环境运行Python代码访问在线服务。schema、 ip、port参数值从**步骤3**获取。

- 执行命令示例如下，直接访问在线服务。

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

图 11-28 访问在线服务

- 创建Python环境，运行Python代码访问在线服务。

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    url = infer_url.format(schema, ip, port)
    response = requests.post(url, data=body)
    print(response.content)
```

说明

由于高速通道特性会缺失负载均衡的能力，因此在多实例时需要自主制定负载均衡策略。

----结束

11.7 ModelArts Standard 的 WebSocket 在线服务全流程开发

背景说明

WebSocket是一种网络传输协议，可在单个TCP连接上进行全双工通信，位于OSI模型的应用层。WebSocket协议在2011年由IETF标准化为RFC 6455，后由RFC 7936补充规范。Web IDL中的WebSocket API由W3C标准化。

WebSocket使得客户端和服务器之间的数据交换变得更加简单，允许服务端主动向客户端推送数据。在WebSocket API中，浏览器和服务器只需要完成一次握手，两者之间就可以建立持久性的连接，并进行双向数据传输。

前提条件

- 用户需有一定的Java开发经验，熟悉jar打包流程。
- 用户需了解WebSocket协议的基本概念及调用方法。
- 用户需熟悉Docker制作镜像的方法。

约束与限制

- WebSocket协议只支持部署在线服务。
- 只支持自定义镜像导入AI应用部署的在线服务。

准备工作

ModelArts使用WebSocket完成推理需要用户自己准备自定义镜像，该自定义镜像需要在单机环境下能够提供完整的WebSocket服务，如完成WebSocket的握手，client向server发送数据，server向client发送数据等。模型的推理过程在自定义镜像中完成，如下载模型，加载模型，执行预处理，完成推理，拼装响应体等。

操作步骤

WebSocket在线服务开发操作步骤如下：

- **上传镜像至容器镜像服务**
- **使用镜像创建AI应用**
- **使用AI应用部署在线服务**
- **WebSocket在线服务调用**

上传镜像至容器镜像服务

将准备好的本地镜像上传到容器镜像服务（SWR）。

使用镜像创建 AI 应用

1. 登录ModelArts管理控制台，进入“ AI应用”页面，单击“创建”，跳转至创建AI应用页面。

2. 完成AI应用配置，部分配置如下：
   - 元模型来源：选择“从容器镜像中选择”。
   - 容器镜像所在的路径：选择**上传镜像至容器镜像服务**上传的路径。
   - 容器调用接口：根据实际情况配置容器调用接口。
   - 健康检查：保持默认。如果镜像中配置了健康检查则按实际情况配置健康检查。

   图 11-29 AI 应用配置参数

3. 单击“立即创建”，进入AI应用列表页，等AI应用状态变为“正常”，表示AI应用创建成功。

使用 AI 应用部署在线服务

1. 登录ModelArts管理控制台，进入“部署上线 >在线服务”页面，单击“部署”，跳转至在线服务部署页面。
2. 完成服务的配置，部分配置如下：
   - 选择AI应用及版本：选择**使用镜像创建AI应用**创建完成的AI应用及版本
   - 升级为WebSocket：打开开关

   图 11-30 升级为 WebSocket

3. 单击“下一步”，确认配置后“提交”，完成在线服务的部署。返回在线服务列表页，查看服务状态变为“运行中”，表示服务部署成功。

WebSocket 在线服务调用

WebSocket协议本身不提供额外的认证方式。不管自定义镜像里面是ws还是wss，经过ModelArts平台出去的WebSocket协议都是wss的。同时wss只支持客户端对服务端的单向认证，不支持服务端对客户端的双向认证。

可以使用ModelArts提供的以下认证方式：

- **token认证**
- **AK/SK**
- **APP认证**

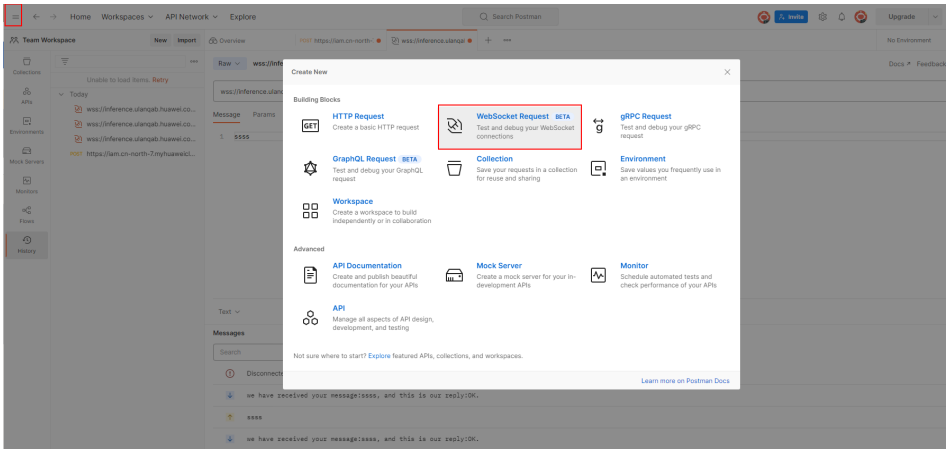
WebSocket服务调用步骤如下（以图形界面的软件Postman进行预测，token认证为例）：

1. **WebSocket连接的建立**
2. **WebSocket客户端和服务端双向传输数据**

步骤1 WebSocket连接的建立

1. 打开Postman（需选择8.5 以上版本，以10.12.0为例）工具，单击左上角 ，选择“File>New”，弹出新建对话框，选择“WebSocket Request”（当前为beta版本）功能：

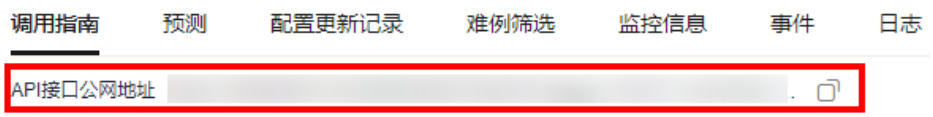
   图 11-31 选择 WebSocket Request 功能

2. 在新建的窗口中填入WebSocket连接信息：

   左上角选择Raw，不要选择Socket.IO（一种WebSocket实现，要求客户端跟服务端都要基于Socket.IO），地址栏中填入从服务详情页“调用指南”页签中获取“API接口调用公网地址”后面的地址。如果自定义镜像中有更细粒度的地址，则在地址后面追加该URL。如果有queryString，那么在params栏中添加参数。在header中添加认证信息（不同认证方式有不同header，跟https的推理服务相同）。选择单击右上的connect按钮，建立WebSocket连接。

   图 11-32 获取 API 接口调用公网地址

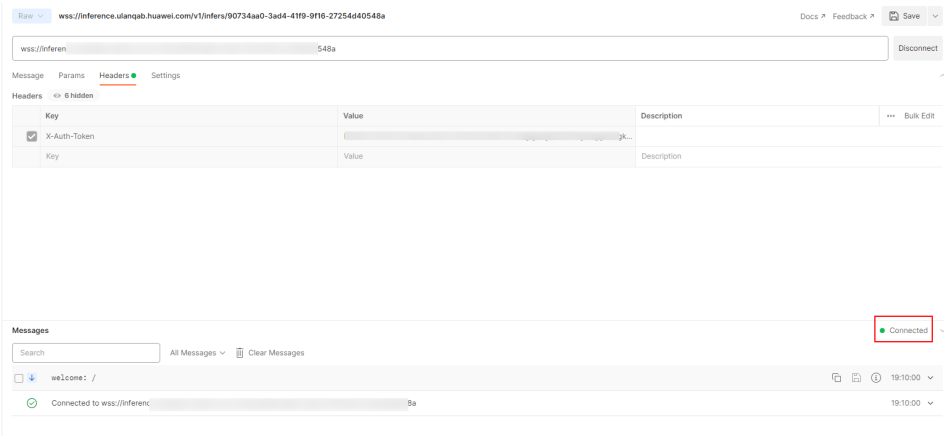
   

说明

- 如果信息正确，右下角连接状态处会显示：CONNECTED；
- 如果无法建立连接，如果是401状态码，检查认证信息；
- 如果显示WRONG_VERSION_NUMBER等关键字，检查自定义镜像的端口和ws跟wss的配置是否正确。

连接成功后结果如下：

图 11-33 连接成功



须知

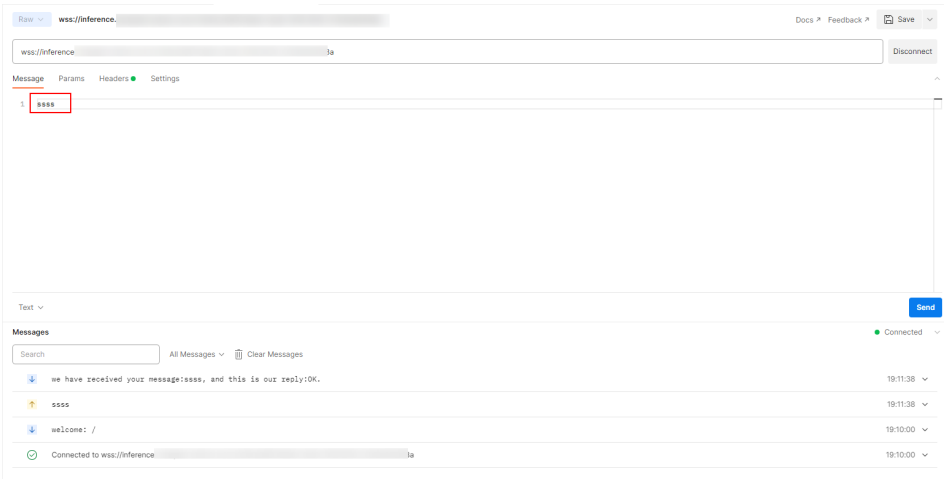
优先验证自定义镜像提供的websocket服务的情况，不同的工具实现的websocket服务会有不同，可能出现连接建立后维持不住，可能出现请求一次后连接就中断需要重新连接的情况，ModelArts平台只保证，未上ModelArts前自定义镜像的websocket的形态跟上了ModelArts平台后的websocket形态相同（除了地址跟认证方式不同）。

步骤2 WebSocket客户端和服务端双向传输数据

连接建立后，WebSocket使用TCP完成全双工通信。WebSocket的客户端可以往服务端发送数据，客户端有不同的实现，同一种语言也存在不同的lib包的实现，这里不考虑实现的不同种类。

客户端发送的内容在协议的角度不限定格式，Postman支持Text/Json/XML/HTML/Binary，以text为例，在输入框中输入要发送的文本，单击右侧中部的Send按钮即可将请求发往服务端，当文本内容过长，可能会导致postman工具卡住。

图 11-34 发送数据



----结束

12 历史待下线案例

12.1 使用 AI Gallery 的订阅算法实现花卉识别

本案例以“ResNet_v1_50”算法、花卉识别数据集为例，指导如何从AI Gallery下载数据集和订阅算法，然后使用算法创建训练模型，将所得的模型部署为在线服务。其他算法操作步骤类似，可参考“ResNet_v1_50”算法操作。

步骤1：准备训练数据

步骤2：订阅算法

步骤3：使用订阅算法创建训练作业

步骤4：创建AI应用

步骤5：部署为在线服务（CPU）

步骤6：清除资源

说明

费用说明：本案例使用过程中，从AI Gallery下载数据集和订阅算法免费，在ModelArts上运行训练作业推荐使用免费资源，将模型部署为在线服务推荐使用免费资源。但是数据集存储在OBS桶中会收取少量费用，具体计费请参见**OBS价格详情页**，案例使用完成后请及时清除资源和数据。

准备工作

- 注册华为账号并开通华为云、实名认证
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在弹出的“访问授权”窗口中，

  授权对象类型：**所有用户**

  委托选择：**新增委托**

权限配置：普通用户

选择完成后勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 1：准备训练数据

1. 从AI Gallery下载训练数据，单击链接**四类花卉图像分类小数据集**，进入数据集详情页。
2. 选择“数据集文件”页签后，单击“下载文件”跳转至下载详情页面。
3. 在下载详情页面，填写参数。
   - 下载方式：选择“对象存储服务（OBS）”
   - 目标区域：选择“华北-北京四”（即要部署服务的云服务区）
   - 目标位置：请选择一个空的OBS目录，本示例为“/test-modelartsz/dataset-flower/”

     说明

     此处从AI Gallery下载并使用数据集是限时免费的，但数据集存储在OBS，从OBS中读取数据需要根据OBS的计费原则收费。
4. 确认无误后，单击确定。页面自动跳转到“我的数据>我的下载”页面，请耐心等待，预计5分钟左右。
5. 下载完成后，您可以单击目标位置跳转至OBS桶中查看是否存在已下载的数据。

步骤 2：订阅算法

1. 在AI Gallery搜索“ResNet_v1_50”，进入算法详情页。
2. 单击右侧的“训练 > ModelArts”后，选择ModelArts的云服务区域（即要部署服务的云服务区），单击“确认”，跳转至ModelArts的“算法管理>我的订阅”中。

步骤 3：使用订阅算法创建训练作业

算法订阅成功后，算法将呈现在“算法管理>我的订阅”中，您可以使用订阅的“ResNet_v1_50”算法创建训练作业，获得模型。

1. 进入“算法管理 > 我的订阅”页面，选择订阅的“图像分类-ResNet_v1_50”算法，单击操作列的“创建训练作业”。
2. 在创建训练作业页面，参考如下说明填写关键参数。
   - “创建方式>我的订阅”：系统默认选择订阅的算法，请勿随意修改。
   - “训练输入”：选择数据存储位置，然后从弹出的窗口中选择**步骤1：准备训练数据**中下载好的数据dataset-flower。
   - “训练输出”：选择一个OBS空目录存储训练输出的模型，例如：“test-modelartsz/output-flower”。
   - “超参”：建议采用默认值。
   - “资源类型”：可以选择限时免费的GPU规格资源，如果希望训练效率更高，可以选择收费的GPU资源。
   - “计算节点个数”：建议采用默认值1。

3. 参数填写完成后，单击“提交”，根据界面提示确认规格，单击“确定”，完成训练作业创建。
4. 进入“训练管理 > 训练作业”页面，等待训练作业完成。

   训练作业运行需要几分钟时间，请耐心等待。根据经验，选择样例数据集，使用GPU资源运行，预计3分钟左右可完成。

   当训练作业的状态变更为“已完成”时，表示已运行结束。

   您可以单击训练作业名称，进入详情页面，了解训练作业的“配置信息”、“日志”、“资源占用情况”和“评估结果”等信息。您也可以在配置的“训练输出位置”对应的OBS目录下获得训练生成的模型。

步骤 4：创建 AI 应用

1. 在训练作业详情页的右上角单击“创建AI应用”，进入创建AI应用页面。

   也可以在ModelArts管理控制台，选择“AI应用管理 > AI应用”，在“我的AI应用”页面，单击“创建”，进入创建AI应用页面。
2. 在创建AI应用页面，系统会自动根据上一步训练作业填写参数，参考如下说明确认关键参数。

   “元模型来源”：系统自动选择“从训练中选择”。

   “选择训练作业”：系统自动选择上一步创建的训练作业。

   “AI引擎”：系统自动写入该模型的AI引擎，无需修改。

   “推理代码”：系统自动放置推理代码到OBS输出路径，无需修改。

   “部署类型”：默认选择“在线服务”。
3. 参数填写完成后，单击“立即创建”。页面自动跳转至AI应用列表页面，等待创建结果，预计2分钟左右。

   当AI应用的状态变为“正常”时，表示创建成功。

步骤 5：部署为在线服务（CPU）

AI应用创建成功后，可将其部署为在线服务，在部署时可使用CPU资源。

1. 单击AI应用名称左侧的单选按钮，在列表页底部展开“版本列表”，在版本的操作列中单击“部署 > 在线服务”。
2. 在部署页面，参考如下说明填写关键参数。
   - “资源池”：选择“公共资源池”。
   - “选择AI应用及版本”：AI应用来源及版本会自动选择前面创建的AI应用。
   - “计算节点规格”：在下拉框中选择限时免费的CPU资源，如果限时免费资源售罄，建议选择收费CPU资源进行部署。
   - “计算节点个数”，默认设置为“1”。
   - 其他参数可使用默认值。

   说明

   选择CPU资源部署模型会收取少量费用，具体费用以界面信息为准。

   如果需要使用GPU资源部署上线，需要进入模型所在位置，即**步骤3：使用订阅算法创建训练作业**步骤生成的“训练输出”路径，进入“model”目录，打开并编辑“config.json”文件，将“runtime”的配置修改为ModelArts支持的GPU规格，例如"runtime": "tf1.13-python3.6-gpu"。修改完成后，重新执行**导入模型**和**部署为在线服务**的操作。
3. 参数设置完成后，单击“下一步”，确认规格参数，单击“提交”，完成在线服务的部署。

4. 您可以进入“部署上线 > 在线服务”页面，等待服务部署完成，当服务状态变为“运行中”时，表示服务部署成功。预计时长2分钟左右。
5. 在线服务部署完成后，您可以单击操作列的预测，进入服务详情页的“预测”页面。
6. 在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”进行预测。此处提供一个预测样例图供使用。

步骤 6：清除资源

为避免产生不必要的费用，通过此示例学习订阅算法的使用后，建议您清除相关资源，避免造成资源浪费。

- 停止在线服务：在“在线服务”页面，单击对应服务操作列的“停止”。
- 删除训练作业：在“训练作业”页面，单击操作列的“删除”。
- 删除数据：前往OBS，删除数据，然后删除文件夹及OBS桶。

12.2 示例：从 0 到 1 制作自定义镜像并用于训练（Pytorch+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是PyTorch，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **前提条件**
2. **Step1 创建OBS桶和文件夹**
3. **Step2 准备训练脚本并上传至OBS**
4. **Step3 准备镜像主机**
5. **Step4 制作自定义镜像**

6. **Step5 上传镜像至SWR服务**
7. **Step6 在ModelArts上创建训练作业**

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如**表12-1**所示，示例中的桶名称“test-modelarts”和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见**创建桶**和**新建文件夹**。

请确保您使用的OBS与ModelArts在同一区域。

表 12-1 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/pytorch/demo-code/”	用于存储训练脚本文件。
“obs://test-modelarts/pytorch/log/”	用于存储训练日志文件。

Step2 准备训练脚本并上传至 OBS

准备本案例所需的训练脚本“pytorch-verification.py”文件，并上传至OBS桶的“obs://test-modelarts/pytorch/demo-code/”文件夹下。

“pytorch-verification.py”文件内容如下：

```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用Ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤 。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以执行以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 执行如下命令确认Docker Engine版本。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   ...
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 准备可用的pip源文件pip.conf 。本示例使用华为开源镜像站提供的pip源，其pip.conf文件内容如下。

   ```
   [global]
   index-url = https://repo.huaweicloud.com/repository/pypi/simple
   trusted-host = repo.huaweicloud.com
   timeout = 120
   ```

   说明

   在华为开源镜像站https://mirrors.huaweicloud.com/home中，搜索pypi ，也可以查看“pip.conf”文件内容。

5. 下载“torch*.whl ”文件。

   在网站“https://download.pytorch.org/whl/torch_stable.html”搜索并下载如下whl文件。

   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   说明

   “+”符号的URL编码为“%2B”，在上述网站中搜索目标文件名时，需要将原文件名中的“+”符号替换为“%2B”。

   例如“torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl”。

6. 下载Miniconda3安装文件。

   使用地址https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应python 3.7.13）。

7. 将上述pip源文件、torch*.whl文件、Miniconda3安装文件放置在context文件夹内，context文件夹内容如下。


```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

8. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

```
# 容器镜像构建主机需要连通公网

# 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

# 基础容器镜像的默认用户已经是 root
# USER root

# 使用华为开源镜像站提供的 pypi 配置
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# 复制待安装文件到基础容器镜像中的 /tmp 目录
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# 使用 Miniconda3 默认 python 环境 (即 /home/ma-user/miniconda3/bin/pip) 安装 torch*.whl
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim和curl 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
```



```
USER ma-user
WORKDIR /home/ma-user
```

关于Dockerfile文件编写的更多指导内容参见**Docker官方文档**。

9. 确认已创建完成Dockerfile文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像pytorch:1.8.1-cuda11.1。

```
docker build . -t pytorch:1.8.1-cuda11.1
```

构建过程结束时出现如下构建日志说明镜像构建成功。

```
Successfully tagged pytorch:1.8.1-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
#此处以华为云cn-north-4为例
sudo docker tag pytorch:1.8.1-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```

   b. 使用docker push命令上传镜像。

```
#region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
#此处以华为云cn-north-4为例
sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1
```

6. 完成镜像上传后，在容器镜像服务控制台的“我的镜像”页面可查看已上传的自定义镜像。

“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”

- 镜像地址：**Step5 上传镜像至SWR服务**中创建的镜像。“swr.cn-north-4.myhuaweicloud.com/deep-learning/pytorch:1.8.1-cuda11.1”
- 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/pytorch/demo-code/”，训练代码会被自动下载至训练容器的“${MA_JOB_DIR}/demo-code”目录中，“demo-code”为OBS存放代码路径的最后一级目录，可以根据实际修改。
- 启动命令：“/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py”，此处的“demo-code”为用户自定义的OBS存放代码路径的最后一级目录，可以根据实际修改。
- 资源池：选择公共资源池
- 类型：选择GPU或者CPU规格。
- 永久保存日志：打开
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/pytorch/log/”

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。训练作业执行成功后，日志信息如下所示。

12.3 示例：从 0 到 1 制作自定义镜像并用于训练（MPI+CPU/GPU）

本章节介绍如何从0到1制作镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MPI，训练使用的资源是CPU或GPU。

说明

本实践教程仅适用于新版训练作业。

场景描述

本示例使用Linux x86_64架构的主机，操作系统ubuntu-18.04，通过编写Dockerfile文件制作自定义镜像。

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用CPU/GPU规格资源运行训练任务。

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. 前提条件
2. Step1 创建OBS桶和文件夹
3. Step2 准备脚本文件并上传至OBS中
4. Step3 准备镜像主机
5. Step4 制作自定义镜像
6. Step5 上传镜像至SWR服务
7. Step6 在ModelArts上创建训练作业

前提条件

已注册华为账号并开通华为云，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。

Step1 创建 OBS 桶和文件夹

在OBS服务中创建桶和文件夹，用于存放样例数据集以及训练代码。需要创建的文件夹列表如表12-2所示，示例中的桶名称“test-modelarts” 和文件夹名称均为举例，请替换为用户自定义的名称。

创建OBS桶和文件夹的操作指导请参见创建桶和新建文件夹。

请确保您使用的OBS与ModelArts在同一区域。

表 12-2 OBS 桶文件夹列表

文件夹名称	用途
“obs://test-modelarts/mpi/demo-code/”	用于存储MPI启动脚本与训练脚本文件。
“obs://test-modelarts/mpi/log/”	用于存储训练日志文件。

Step2 准备脚本文件并上传至 OBS 中

准备本案例所需的MPI启动脚本run_mpi.sh文件和训练脚本mpi-verification.py文件，并上传至OBS桶的“obs://test-modelarts/mpi/demo-code/”文件夹下。

- MPI启动脚本run_mpi.sh文件内容如下：

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
```



```
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-
p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
```

```
    -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
    "$@"

RET_CODE=$?

if [ $RET_CODE -ne 0 ]; then
    echo "[run_mpi] exec command failed, exited with $RET_CODE"
else
    echo "[run_mpi] exec command successfully, exited with $RET_CODE"
fi

# stop 1...N worker by killing the sleep proc
sed -i '1d' ${MY_HOME}/hostfile
if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
    echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

    sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

    mpirun \
    --hostfile ${MY_HOME}/hostfile \
    --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
    -x PATH -x LD_LIBRARY_PATH \
    pkill sleep \
    > /dev/null 2>&1
fi

echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

说明

“run_mpi.sh”脚本需要以LF作为换行符。使用CRLF作为换行符会导致训练作业运行失败，日志中会打印“$'\r': command not found”的错误信息。

- 训练脚本mpi-verification.py文件内容如下：

```
import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

    # https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
    print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
    print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
    print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
```

Step3 准备镜像主机

准备一台Linux x86_64架构的主机，操作系统使用ubuntu-18.04。您可以准备相同规格的弹性云服务器ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录Linux弹性云服务器**。“CPU架构”选择“x86计算”，“镜像”选择“公共镜像”，推荐使用Ubuntu18.04的镜像。

Step4 制作自定义镜像

目标：构建安装好如下软件的容器镜像，并使用ModelArts训练服务运行。

- ubuntu-18.04

- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

此处介绍如何通过编写Dockerfile文件制作自定义镜像的操作步骤。

1. 安装Docker。

   以Linux x86_64架构的操作系统为例，获取Docker安装包。您可以使用以下指令安装Docker。关于安装Docker的更多指导内容参见**Docker官方文档**。

   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```

   如果**docker images**命令可以执行成功，表示Docker已安装，此步骤可跳过。

2. 确认Docker Engine版本。执行如下命令。

   ```
   docker version | grep -A 1 Engine
   ```

   命令回显如下。

   ```
   Engine:
    Version:          18.09.0
   ```

   说明

   推荐使用大于等于该版本的Docker Engine来制作自定义镜像。

3. 准备名为context的文件夹。

   ```
   mkdir -p context
   ```

4. 下载Miniconda3安装文件。

   使用地址 https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh，下载Miniconda3 py37 4.12.0安装文件（对应 python 3.7.13）。

5. 下载openmpi 3.0.0安装文件。

   使用地址https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz，下载 horovod v0.22.1已经编译好的openmpi 3.0.0文件。

6. 将上述Miniconda3安装文件、openmpi 3.0.0文件放置在context文件夹内，context文件夹内容如下。

   ```
   context
   ├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
   └── openmpi-3.0.0-bin.tar.gz
   ```

7. 编写容器镜像Dockerfile文件。

   在context文件夹内新建名为Dockerfile的空文件，并将下述内容写入其中。

   ```
   # 容器镜像构建主机需要连通公网

   # 基础容器镜像, https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
   #
   # https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
   # require Docker Engine >= 17.05
   #
   # builder stage
   FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

   # 基础容器镜像的默认用户已经是 root
   # USER root

   # 复制 Miniconda3 (python 3.7.13) 安装文件到基础容器镜像中的 /tmp 目录
   COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

   # 安装 Miniconda3 到基础容器镜像的 /home/ma-user/miniconda3 目录中
   # https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
   RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3
   ```



```
# 构建最终容器镜像
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# 安装 vim / curl / net-tools / ssh 工具（依然使用华为开源镜像站）
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# 安装 horovod v0.22.1 已经编译好的 openmpi 3.0.0 文件
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# 增加 ma-user 用户 (uid = 1000, gid = 100)
# 注意到基础容器镜像已存在 gid = 100 的组，因此 ma-user 用户可直接使用
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

# 从上述 builder stage 中复制 /home/ma-user/miniconda3 目录到当前容器镜像的同名目录
COPY --chown=ma-user:100 --from=builder /home/ma-user/miniconda3 /home/ma-user/miniconda3

# 设置容器镜像预置环境变量
# 请务必设置 PYTHONUNBUFFERED=1, 以免日志丢失
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

# 设置容器镜像默认用户与工作目录
USER ma-user
WORKDIR /home/ma-user

# 配置 sshd，使得 ssh 可以免密登录
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

关于Dockerfile文件编写的更多指导内容参见 **Docker 官方文档**。

8. 确认已创建完成Dockerfile 文件。此时context文件夹内容如下。

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. 构建容器镜像。在Dockerfile文件所在的目录执行如下命令构建容器镜像mpi:3.0.0-cuda11.1。
```
docker build . -t mpi:3.0.0-cuda11.1
```
构建过程结束时出现如下构建日志说明镜像构建成功。
```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step5 上传镜像至 SWR 服务

1. 登录容器镜像服务控制台，选择区域，要和ModelArts区域保持一致，否则无法选择到镜像。
2. 单击右上角“创建组织”，输入组织名称完成组织创建。请自定义组织名称，本示例使用“deep-learning”，下面的命令中涉及到组织名称“deep-learning”也请替换为自定义的值。
3. 单击右上角“登录指令”，获取登录访问指令，本文选择复制临时登录指令。
4. 以root用户登录本地环境，输入复制的SWR临时登录指令。
5. 上传镜像至容器镜像服务镜像仓库。
   a. 使用docker tag命令给上传镜像打标签。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. 使用docker push命令上传镜像。
   ```
   #region和domain信息请替换为实际值，组织名称deep-learning也请替换为自定义的值。
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
6. 完成镜像上传后，在“容器镜像服务控制台>我的镜像”页面可查看已上传的自定义镜像。
   “swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”即为此自定义镜像的“SWR_URL”。

Step6 在 ModelArts 上创建训练作业

1. 登录ModelArts管理控制台，检查当前账号是否已完成访问授权的配置。如未完成，请参考**使用委托授权**。针对之前使用访问密钥授权的用户，建议清空授权，然后使用委托进行授权。
2. 在ModelArts管理控制台，左侧导航栏中选择“训练管理 > 训练作业”，默认进入“训练作业”列表。
3. 在“创建训练作业”页面，填写相关参数信息，然后单击“提交”。
   - 创建方式：选择“自定义算法”
   - 启动方式：选择“自定义”
   - 镜像地址：“swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1”
   - 代码目录：设置为OBS中存放启动脚本文件的目录，例如：“obs://test-modelarts/mpi/demo-code/”
   - 启动命令：bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py
   - 环境变量：添加“MY_SSHD_PORT = 38888”
   - 资源池：选择公共资源池
   - 类型：选择GPU规格
   - 计算节点个数：选择“1”或“2”

- 永久保存日志：打开
- 作业日志路径：设置为OBS中存放训练日志的路径。例如：“obs://test-modelarts/mpi/log/”

4. 在“规格确认”页面，确认训练作业的参数信息，确认无误后单击“提交”。
5. 训练作业创建完成后，后台将自动完成容器镜像下载、代码目录下载、执行启动命令等动作。

   训练作业一般需要运行一段时间，根据您的训练业务逻辑和选择的资源不同，训练时长将持续几十分钟到几小时不等。

   计算节点个数选择为2，训练作业也可以运行。

12.4 示例：从 0 到 1 制作自定义镜像并用于训练（MindSpore+Ascend）

本案例介绍如何从0到1制作Ascend容器镜像，并使用该镜像在ModelArts平台上进行训练。镜像中使用的AI引擎是MindSpore，训练使用的资源是专属资源池的Ascend芯片。

约束限制

- 由于案例中需要下载商用版CANN，因此本案例仅面向有下载权限的渠道用户，非渠道用户建议参考其他自定义镜像制作教程。
- Mindspore版本与CANN版本，CANN版本与Ascend驱动/固件版本均有严格的匹配关系，版本不匹配会导致训练失败。

场景描述

目标：构建安装如下软件的容器镜像，并在ModelArts平台上使用Ascend规格资源运行训练任务。

- ubuntu-18.04
- cann-6.3.RC2 (商用版本)
- python-3.7.13
- mindspore-2.1.1

说明

- 本教程以cann-6.3.RC2.、mindspore-2.1.1为例介绍。
- 本示例仅用于示意Ascend容器镜像制作流程，且在匹配正确的Ascend驱动/固件版本的专属资源池上运行通过。

操作流程

使用自定义镜像创建训练作业时，需要您熟悉docker软件的使用，并具备一定的开发经验。详细步骤如下所示：

1. **Step1 创建OBS桶和文件夹**
2. **Step2 准备脚本文件并上传至OBS中**
3. **Step3 制作自定义镜像**

4. **Step4 上传镜像至SWR**
5. **Step5 在ModelArts上创建Notebook并调试**
6. **Step6 在ModelArts上创建训练作业**

12.5 使用 ModelArts Standard 一键完成商超商品识别模型部署

ModelArts的AI Gallery中提供了大量免费的模型供用户一键部署，进行AI体验学习。

本文以“商超商品识别”模型为例，完成从AI Gallery订阅模型，到ModelArts一键部署为在线服务的免费体验过程。

“商超商品识别”模型可以识别81类常见超市商品（包括蔬菜、水果和饮品），并给出置信度最高的5类商品的置信度得分。

步骤 1：准备工作

- 已注册华为账号并开通华为云，进行了实名认证，且在使用ModelArts前检查账号状态，账号不能处于欠费或冻结状态。
  - **注册华为账号并开通华为云**
  - **进行实名认证**
- 配置委托访问授权

  ModelArts使用过程中涉及到OBS、SWR、IEF等服务交互，首次使用ModelArts需要用户配置委托授权，允许访问这些依赖服务。

  a. 使用华为云账号登录**ModelArts管理控制台**，在左侧导航栏单击“权限管理”，进入“权限管理”页面，单击“添加授权”。

  b. 在“访问授权”页面，选择需要授权的“授权对象类型”，选择新增委托及其对应的权限“普通用户”，并勾选“我已经详细阅读并同意《ModelArts服务声明》”，然后单击“创建”。

  c. 完成配置后，在ModelArts控制台的权限管理列表，可查看到此账号的委托配置信息。

步骤 2：订阅模型

“商超商品识别”的模型共享在AI Gallery中。您可以前往AI Gallery，免费订阅此模型。

1. 单击案例链接**商超商品识别**，进入模型详情页。
2. 完成模型订阅。

   在模型详情页，单击“订阅”，阅读并勾选同意《数据安全与隐私风险承担条款》和《华为云AI Gallery服务协议》，单击“继续订阅”。订阅模型完成后，页面的“订阅”按钮显示为“已订阅”。
3. 从模型详情页进入ModelArts控制台的订阅列表。

   在模型详情页，单击“前往控制台”。在弹出的“选择云服务区域”页面选择ModelArts所在的云服务区域，单击“确定”跳转至ModelArts控制台的“AI应用 > 订阅应用”页面。
4. 在“订阅应用”列表，单击“版本数量”，在右侧展开版本列表，当订阅模型的版本列表的状态显示为“就绪”时表示模型可以使用。

步骤 3：使用订阅模型部署在线服务

模型订阅成功后，可将此模型部署为在线服务

1. 在展开的版本列表中，单击“部署 > 在线服务”跳转至部署页面。
2. 在部署页面，参考如下说明填写关键参数。

   “名称”：自定义一个在线服务的名称，也可以使用默认值，此处以“商超商品识别服务”为例。

   “资源池”：选择“公共资源池”。

   “AI应用来源”和“选择AI应用及版本”：会自动选择订阅模型。

   “计算节点规格”：在下拉框中选择“限时免费”资源，勾选并阅读免费规格说明。

   其他参数可使用默认值。

   说明

   如果限时免费资源售罄，建议选择收费CPU资源进行部署。当选择收费CPU资源部署在线服务时会收取少量资源费用，具体费用以界面信息为准。
3. 参数配置完成后，单击“下一步”，确认规格参数后，单击“提交”启动在线服务的部署。
4. 任务提交成功后，单击“查看任务详情”，等待服务状态变为“运行中”时，表示服务部署成功。预计时长4分钟左右。

步骤 4：预测结果

1. 在线服务部署完成后，单击“预测”页签。
2. 在“预测”页签，单击“上传”，上传一个测试图片，单击“预测”查看预测结果。此处提供一个样例图片供预测使用。

   说明

   本案例中使用的订阅模型可以识别**81类**常见超市商品，模型对预测图片有一定范围和要求，不满足条件的图片会影响预测结果的准确性。

步骤 5：清理资源

体验结束后，建议暂停或删除服务，避免占用资源，造成资源浪费。

- 停止在线服务：在“在线服务”列表，单击对应服务操作列的“更多 > 停止”。
- 删除在线服务：在“在线服务”列表，单击对应服务操作列的“更多 > 删除”。

12.6 从 0-1 制作自定义镜像并创建 AI 应用

针对ModelArts目前不支持的AI引擎，您可以针对该引擎构建自定义镜像，并将镜像导入ModelArts，创建为AI应用。本文详细介绍如何使用自定义镜像完成AI应用的创建，并部署成在线服务。

操作流程如下：

1. **本地构建镜像**：在本地制作自定义镜像包，镜像包规范可参考**创建AI应用的自定义镜像规范**。
2. **本地验证镜像并上传镜像至SWR服务**：验证自定义镜像的API接口功能，无误后将自定义镜像上传至SWR服务。

3. **将自定义镜像创建为AI应用**：将上传至SWR服务的镜像导入ModelArts的AI应用管理。
4. **将AI应用部署为在线服务**：将导入的模型部署上线。

本地构建镜像

以linux x86_x64架构的主机为例，您可以购买相同规格的ECS或者应用本地已有的主机进行自定义镜像的制作。

购买ECS服务器的具体操作请参考**购买并登录弹性云服务器**。镜像选择公共镜像，推荐使用ubuntu18.04的镜像。

图 12-1 创建 ECS 服务器-选择 X86 架构的公共镜像

1. 登录主机后，安装Docker，可参考**Docker官方文档**。也可执行以下命令安装docker。
   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```
2. 获取基础镜像。本示例以Ubuntu18.04为例。
   ```
   docker pull ubuntu:18.04
   ```
3. 新建文件夹“self-define-images”，在该文件夹下编写自定义镜像的“Dockerfile”文件和应用服务代码“test_app.py”。本样例代码中，应用服务代码采用了flask框架。

   文件结构如下所示
   ```
   self-define-images/
     --Dockerfile
     --test_app.py
   ```
   - “Dockerfile”
     ```
     From ubuntu:18.04
     # 配置华为云的源，安装 python、python3-pip 和 Flask
     RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
       sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
     && \
       sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list
     && \
       apt-get update && \
       apt-get install -y python3 python3-pip && \
       pip3 install  --trusted-host https://repo.huaweicloud.com -i https://repo.huaweicloud.com/
     repository/pypi/simple  Flask

     # 复制应用服务代码进镜像里面
     COPY test_app.py /opt/test_app.py

     # 指定镜像的启动命令
     CMD python3  /opt/test_app.py
     ```
   - “test_app.py”
     ```
     from flask import Flask, request
     import json
     ```

```
app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("---------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("---------- in goodbye func ----------")
    return '\nGoodbye!\n'


@app.route('/', methods=['POST'])
def default_func():
    print("---------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. 进入“self-define-images”文件夹，执行以下命令构建自定义镜像“test:v1”。
   ```
   docker build -t test:v1 .
   ```
5. 您可以使用“docker images”查看您构建的自定义镜像。

本地验证镜像并上传镜像至 SWR 服务

1. 在本地环境执行以下命令启动自定义镜像
   ```
   docker run -it -p 8080:8080 test:v1
   ```

   图 12-2 启动自定义镜像

   ```
   :/opt/file# docker run -it -p 8080:8080 test:v1
   * Serving Flask app "test_app" (lazy loading)
   * Environment: production
     WARNING: This is a development server. Do not use it in a production deployment.
     Use a production WSGI server instead.
   * Debug mode: off
   * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
   ```

2. 另开一个终端，执行以下命令验证自定义镜像的三个API接口功能。
   ```
   curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/
   curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
   curl -X GET 127.0.0.1:8080/goodbye
   ```

   如果验证自定义镜像功能成功，结果如下图所示。

   图 12-3 校验接口

   ```
   root@          :~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}'  127.0.0.1:8080/

    called default func !
    {'name': 'Tom'}
   root@          :~# curl -X POST -H "Content-Type: application/json" --data '{"name":"Tom"}' 127.0.0.1:8080/greet
   {
       "response": "Hello, Tom!"
   }root@          :~#curl -X GET 127.0.0.1:8080/goodbye

   Goodbye!
   ```

3. 上传自定义镜像至SWR服务。
4. 完成自定义镜像上传后，您可以在“容器镜像服务>我的镜像>自有镜像”列表中看到已上传镜像。

将自定义镜像创建为 AI 应用

参考**从容器镜像中选择元模型**导入元模型，您需要特别关注以下参数：

- 元模型来源：选择“从容器镜像中选择”
  - 容器镜像所在的路径：选择已制作好的自有镜像

    图 12-4 选择已制作好的自有镜像

  - 容器调用接口：指定模型启动的协议和端口号。请确保协议和端口号与自定义镜像中提供的协议和端口号保持一致。
  - 镜像复制：选填，选择是否将容器镜像中的模型镜像复制到ModelArts中。
  - 健康检查：选填，用于指定模型的健康检查。仅当自定义镜像中配置了健康检查接口，才能配置“健康检查”，否则会导致AI应用创建失败。
- apis定义：选填，用于编辑自定义镜像的apis定义。模型apis定义需要遵循ModelArts的填写规范，参见**模型配置文件说明**。

  本样例的配置文件如下所示：

```
[{
    "url": "/",
    "method": "post",
    "request": {
        "Content-type": "application/json"
    },
    "response": {
        "Content-type": "application/json"
    }
  },
{
    "url": "/greet",
    "method": "post",
    "request": {
        "Content-type": "application/json"
    },
    "response": {
        "Content-type": "application/json"
    }
  },
{
    "url": "/goodbye",
    "method": "get",
    "request": {
        "Content-type": "application/json"
    },
    "response": {
        "Content-type": "application/json"
    }
  }
]
```

将 AI 应用部署为在线服务

1. 参考**部署为在线服务**将AI应用部署为在线服务。
2. 在线服务创建成功后，您可以在服务详情页查看服务详情。

3. 您可以通过“预测”页签访问在线服务。

图 12-5 访问在线服务