ModelArts

Best Practices

Issue	01
Date	2025-12-11

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 ModelArts Best Practices

This document provides ModelArts samples concerning a variety of scenarios and AI engines to help you quickly understand the process and operations of using ModelArts for AI development.

LLM Training and Inference

Sample	Scenario	Description
Adapting Mainstream Open-Source Models to AscendFactory NPU Training	Pre-training, SFT full-parameter fine-tuning training, and LoRA fine-tuning training	Describes the training process of mainstream open-source models, such as DeepSeek, Llama, Qwen3, and Qwen-VL series, based on ModelArts. The AscendFactory framework and AI Compute Service NPUs are used for training.
Adapting Mainstream Open-Source Models for NPU Inference on Ascend-vLLM with PyTorch	Inference deployment, inference performance test, inference accuracy test, and inference model quantization	Describes the inference deployment process of mainstream open-source models, such as Llama, Qwen3, and Qwen-VL series, based on ModelArts. The Ascend-vLLM framework and AI Compute Service NPUs are used for inference.

Image Generation Model Training and Inference

Sample	Scenario	Description
Adapting Stable Diffusion for NPU Inference with Diffusers/ComfyUI and Lite Server (6.5.907)	SD1.5, SDXL, SD3.5, and Hunyuan model inference	Describes the inference process of mainstream image generation models based on ModelArts Lite Server. AI Compute Service NPUs are used for inference. After the inference service is started, it can be used in image generation scenarios.
Stable Diffusion XL Inference Guide Based on ModelArts Notebook (6.5.907)	SDXL model inference	Describes the inference process of mainstream image generation models based on ModelArts Standard Notebook. AI Compute Service NPUs are used for inference. After the inference service is started, it can be used in image generation scenarios.

Video Generation Model Training and Inference

Sample	Scenario	Description
Inference Guide for Wan Series Video Generation Models Adapted to NPU via Lite Server	Wan series model inference	Describes the inference process of Wan series models based on ModelArts Lite Server. The PyTorch framework and AI Compute Service NPUs are used for inference.

Configuring ModelArts Standard Permissions

Sample	Function	Scenario	Description
ModelArts Standard Permission Management	IAM permission configuration and management	Permission assignment for IAM users	Assign specific ModelArts operation permissions to the IAM users under a Huawei Cloud account. This prevents exceptions from occurring due to permissions when the IAM users access ModelArts.

ModelArts Standard Model Training

Table 1-1 Custom algorithm samples

Sample	Image	Function	Scenario	Description
Building a Handwritten Digit Recognition Model with ModelArts Standard	PyTorch	Algorithm customization	Handwritten digit recognition	Use your customized algorithm to train a handwritten digit recognition model and deploy the model for prediction.

ModelArts Standard Inference Deployment

Table 1-2 Inference deployment samples

Sample	Function	Scenario	Description
Migrating a Third-Party Inference Framework to a Custom Inference Engine	Third-party frameworks Inference deployment	-	ModelArts allows the deployment of third-party inference frameworks. This section describes how to migrate TF Serving and Triton to a custom inference engine.

2 AI Compute Service Capability Map

ModelArts supports the training and inference of the following open-source models using AI Compute Service NPUs.

LLMs

ModelArts now supports several leading LLMs on AI Compute Service NPUs. These adapted models enable both inference and training tasks directly on NPUs.

Table 2-1 LLM inference capabilities

Supported Model	Supported Model Parameter	Use Case	Software Technology Stack	Documentation
Llama3	Llama3-8B Llama3-70B llama3.1-8B llama3.1-70B llama-3.2-1B llama-3.2-3B	Inference	Ascend-vLLM	LLM Inference
Qwen2	Qwen2-0.5B Qwen2-1.5B Qwen2-7B Qwen2-72B	Inference	Ascend-vLLM	
Qwen2.5	Qwen2.5-0.5B Qwen2.5-1.5B Qwen2.5-3B Qwen2.5-7B Qwen2.5-14B Qwen2.5-32B Qwen2.5-72B	Inference	Ascend-vLLM	

Supported Model	Supported Model Parameter	Use Case	Software Technology Stack	Documentation
Qwen3	Qwen3-0.6B Qwen3-1.7B Qwen3-4B Qwen3-8B Qwen3-14B Qwen3-30B-A3B Qwen3-32B Qwen3-235B-A22B Qwen3-235B-A22B-Thinking-2507 Qwen3-235B-A22B-Instruct-2507 Qwen3-Coder-480B-A35B Qwen3-Embedding-0.6B Qwen3-Embedding-4B Qwen3-Embedding-8B Qwen3-Reranker-0.6B Qwen3-Reranker-4B Qwen3-Reranker-8B	Inference	Ascend-vLLM	
GLMv4	GLM-4-9B	Inference	Ascend-vLLM	
BGE	bge-reranker-v2-m3 bge-base-en-v1.5 bge-base-zh-v1.5 bge-large-en-v1.5 bge-large-zh-v1.5 bge-m3	Inference	Ascend-vLLM	

Supported Model	Supported Model Parameter	Use Case	Software Technology Stack	Documentation
DeepSeek-R1-Distill	DeepSeek-R1-Distill-Llama-8B DeepSeek-R1-Distill-Llama-70B DeepSeek-R1-Distill-Qwen-1.5B DeepSeek-R1-Distill-Qwen-7B DeepSeek-R1-Distill-Qwen-14B DeepSeek-R1-0528-Qwen3-8B	Inference	Ascend-vLLM	

Table 2-2 LLM training capabilities

Supported Model	Supported Model Parameter	Use Case	Documentation
DeepSeek	DeepSeek-R1-671B DeepSeek-V3-671B DeepSeek-V2-Lite 16B	Pre-training and fine-tuning	LLM Training
Llama	Llama3.1-8B/70B Llama3.2-1B/3B	Pre-training and fine-tuning	
Qwen2	Qwen2-0.5B Qwen2-1.5B Qwen2-7B Qwen2-72B	Pre-training and fine-tuning	
Qwen2.5	Qwen2.5-0.5B Qwen2.5-1.5B Qwen2.5-7B Qwen2.5-14B Qwen2.5-32B Qwen2.5-72B	Pre-training and fine-tuning	

Supported Model	Supported Model Parameter	Use Case	Documentation
Qwen3	Qwen3-0.6B Qwen3-1.7B Qwen3-4B Qwen3-8B Qwen3-14B Qwen3-32B Qwen3-30B-A3B Qwen3-235B-A22B	Pre-training and fine-tuning	
GLM-4	GLM-4-9B-Chat	Pre-training and fine-tuning	
Mistral AI	Mixtral-8x7B-Instruct-v0.1	Pre-training and fine-tuning	

Multimodal Models

ModelArts now supports several leading multimodal models on AI Compute Service NPUs. These adapted models enable both inference and training tasks directly on NPUs.

Table 2-3 Multimodal model inference based on the Ascend-vLLM framework

Supported Model	Supported Model Parameter	Use Case	Software Technology Stack	Documentation
Qwen2-VL	Qwen2-VL-2B Qwen2-VL-7B Qwen2-VL-72B	Inference	Ascend-vLLM	LLM Inference
Qwen2.5-VL	Qwen2.5-VL-2B Qwen2.5-VL-7B Qwen2.5-VL-72B	Inference	Ascend-vLLM	

Supported Model	Supported Model Parameter	Use Case	Software Technology Stack	Documentation
InternVL	InternVL2.5-26B InternVL2-llama3-76B-AWQ InternVL3-8B InternVL3-14B InternVL3-38B InternVL3-78B	Inference	Ascend-vLLM	
Gemma	GEMMA-3-27B	Inference	Ascend-vLLM	

Image Generation Models

ModelArts now supports several leading AIGC image generation models on AI Compute Service NPUs. These adapted models enable both inference and training tasks directly on NPUs.

Table 2-4 Text-to-image models

Model	Use Case	Software Technology Stack	Documentation
Stable Diffusion XL (SDXL)	Diffusers inference ComfyUI inference	PyTorch	**Adapting Stable Diffusion for NPU Inference with Diffusers/ComfyUI and Lite Server (6.5.907)** **Stable Diffusion XL Inference Guide Based on ModelArts Notebook (6.5.907)**
Stable Diffusion 1.5 (SD1.5)	Diffusers inference ComfyUI inference	PyTorch	
Stable Diffusion 3.5 (SD3.5)	Diffusers inference ComfyUI inference	PyTorch	
HUNYUAN	Diffusers inference	PyTorch	

Video Generation Models

Table 2-5 Video generation models

Model	Use Case	Software Technology Stack	Documentation
Wan series	Inference Training	PyTorch	**Video Generation Model Training and Inference**

3 DeepSeek Inference Applications on MaaS

3.1 Quickly Building Your Own AI Assistant Using a ModelArts Studio (MaaS) DeepSeek API and Cherry Studio

Operation Scenarios

Large language models (LLMs) are now essential for advancing natural language processing applications. The DeepSeek series offers LLMs designed for efficiency and strong performance.

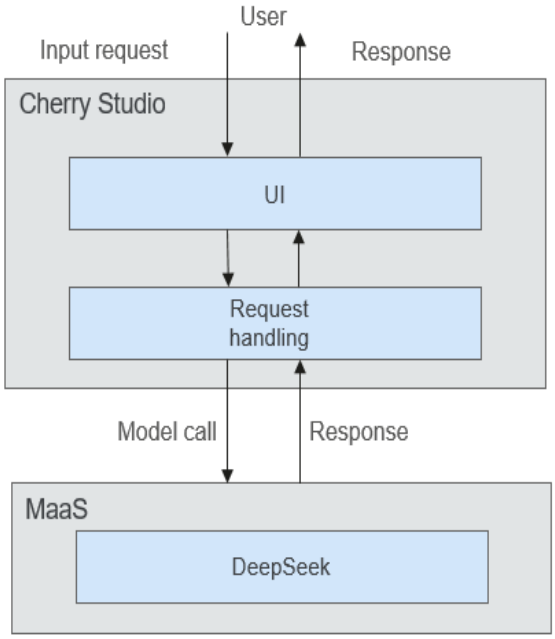
To simplify their use, several tools and platforms have been developed. Among them, Cherry Studio stands out as a versatile open-source desktop application compatible with Windows, macOS, and Linux. It combines popular LLMs like OpenAI, DeepSeek, and Gemini, enabling both cloud-based and local operations. Its features include dialog knowledge bases, AI painting, translation, and seamless model switching.

MaaS integrates DeepSeek models into its platform and enhances their accuracy and efficiency using AI Compute Service. Developers can test these models through API calls.

This guide explains how to access Cherry Studio with MaaS DeepSeek APIs to create your own AI assistant swiftly.

Solution Architecture

Figure 3-1 Solution architecture



- The user submits a request through Cherry Studio.
- Cherry Studio sends the request to MaaS DeepSeek.
- MaaS DeepSeek processes the request and returns the results to Cherry Studio.
- Cherry Studio optimizes the results and returns them to the user.

Billing

Model inference in MaaS uses compute and storage resources, which are billed. Compute resources are billed for running model services. Storage resources are billed for storing data in OBS. Using a built-in MaaS service charges you based on token count. For more information, see **Model Inference Billing Items**.

Prerequisites

- You have created a Huawei Cloud account. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.

Step 1: Downloading and Installing Cherry Studio

Download Cherry Studio from the **official website** or **open-source address**.

Step 2: Obtaining MaaS DeepSeek Interconnection Information

Obtain key details about the target MaaS DeepSeek model service for connecting with Cherry Studio.

1. Create an API key for authentication when calling the MaaS DeepSeek model service.

   You can create up to 30 keys. Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to the **ModelArts Studio (MaaS) console** and select **CN-Hong Kong** from the top navigation bar.

   b. In the navigation pane, choose **API Key Management**.

   c. On the **API Key Management** page, click **Create API Key**, enter the tag and description, and click **OK**.

      The tag and description cannot be modified after the key is created.

      Table 3-1 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

   d. In the **Your Key** dialog box, copy the key and store it securely.

   e. After the key is saved, click **Close**.

      After you click **Close**, the key cannot be viewed again.

2. Obtain the API URL and model name of the MaaS DeepSeek model service.

   The following uses a deployed service as an example. You can also use commercial services to call APIs. For details, see **Subscribing to a Built-in Commercial Service in ModelArts Studio (MaaS)**.

   a. In the navigation pane of **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.

   b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

   c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.

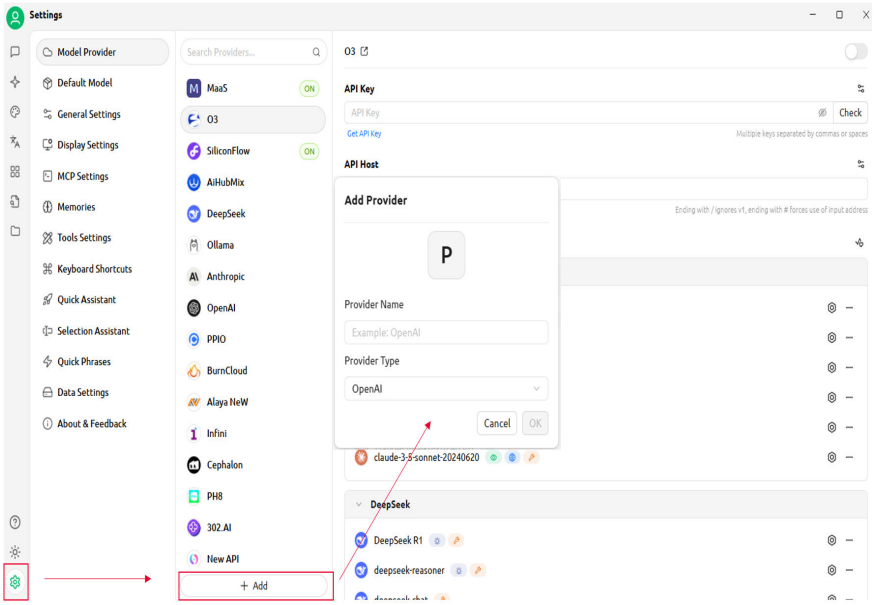
   d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cherry Studio.

Step 3: Configuring MaaS DeepSeek in Cherry Studio

1. Add a MaaS provider in Cherry Studio.

a. In the lower left corner of the Cherry Studio client, click the settings icon and click **Add** in **Model Provider**.

Figure 3-2 Adding a provider



b. Set the provider name and provider type and click **OK**.

Table 3-2 Parameters for adding a provider

Parameter	Description
Provider Name	Enter **MaaS**, which can be changed as needed.
Provider Type	Set this parameter to **OpenAI**.

2. Add the MaaS DeepSeek API key and API URL in Cherry Studio.
   a. In the lower left corner of the Cherry Studio client, click the settings icon.
   b. Click **MaaS** and configure the API key and API URL.

Parameter	Description
API Key	API key created in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**.
API Host	Basic API URL of the MaaS service obtained in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**. Remove **/v1/chat/completions** from the URL and enter it.

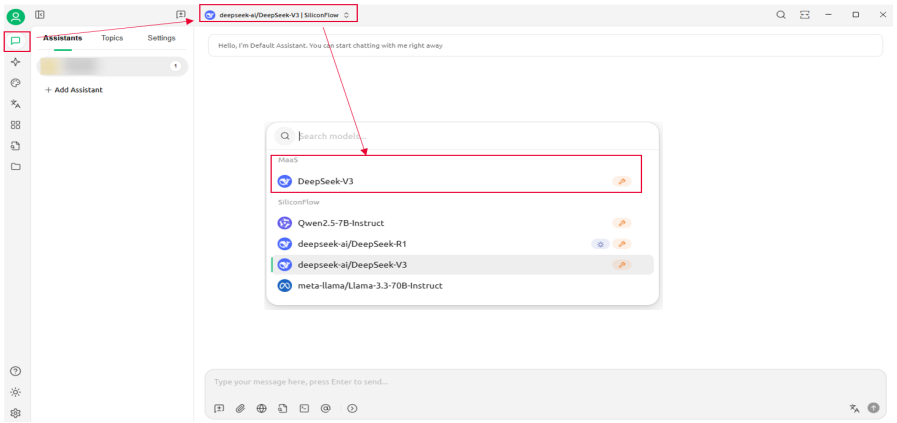
3. Add a model to Cherry Studio.
   a. In the **Models** area, click **Add**.
   b. Set the model ID, model name, and group name, and click **Add Model**.

Parameter	Description
Model ID	Model name obtained in **Step 2: Obtaining MaaS DeepSeek Interconnection Information**
Model	Custom model name
Group Name	Custom group name

Step 4: Using MaaS DeepSeek in Cherry Studio

1. In the navigation pane of Cherry Studio, click  and select the configured model.

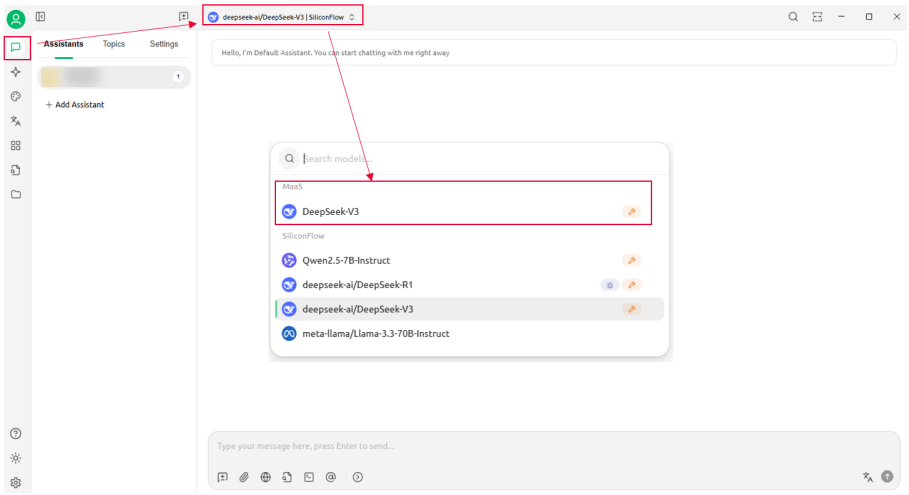
   Figure 3-3 Selecting a model

2. Enter message in the text box to start a conversation.

   Choose the model name in the top menu to switch models.

   Figure 3-4 Model Q&A example

FAQs

- **How Long Does It Take for an API Key to Become Valid After It Is Created in MaaS?**

  A MaaS API key becomes valid a few minutes after creation.
- **How Do I Connect MaaS to Cherry Studio, Cursor, and Cline?**

  MaaS integrates DeepSeek models and supports AI development on multiple platforms. For details, see **Using ModelArts Studio (MaaS) DeepSeek API to Build AI Applications**.

3.2 Quickly Building a Code Editor Using a ModelArts Studio (MaaS) DeepSeek API and Cursor

This guide describes how to use Cursor to call a DeepSeek model deployed on ModelArts Studio to build a code editor.

Operation Scenarios

The Cursor code editor uses AI to enhance developer productivity with its modern features. It combines the strong features of traditional editors like VS Code with AI-powered smart coding tools. These include smart code completion, natural language programming, and code library understanding, which boost development speed. Cursor also supports popular AI models like OpenAI's GPT-4 and DeepSeek, and offers flexible customization. This makes it ideal for users from beginners to professionals.

MaaS deploys DeepSeek models on its platform, allowing developers use them via API calls.

Prerequisites

- You have created a Huawei Cloud account. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.

Step 1: Downloading and Installing Cursor

Download and install Cursor from the **official website**.

Step 2: Preparing for MaaS Model API Access

1. Create an API key.

   You can create up to 30 keys. Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to the **ModelArts Studio (MaaS) console** and select **CN-Hong Kong** from the top navigation bar.

   b. In the navigation pane, choose **API Key Management**.

   c. On the **API Key Management** page, click **Create API Key**, enter the tag and description, and click **OK**.

The tag and description cannot be modified after the key is created.

Table 3-3 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

d. In the **Your Key** dialog box, copy the key and store it securely.

e. After the key is saved, click **Close**.

After you click **Close**, the key cannot be viewed again.

2. Use a model from **My Services**.

The following uses **My Services** as an example. You can also use commercial services to call APIs. For details, see **Subscribing to a Built-in Commercial Service in ModelArts Studio (MaaS)**.

a. In the navigation pane of **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.

b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

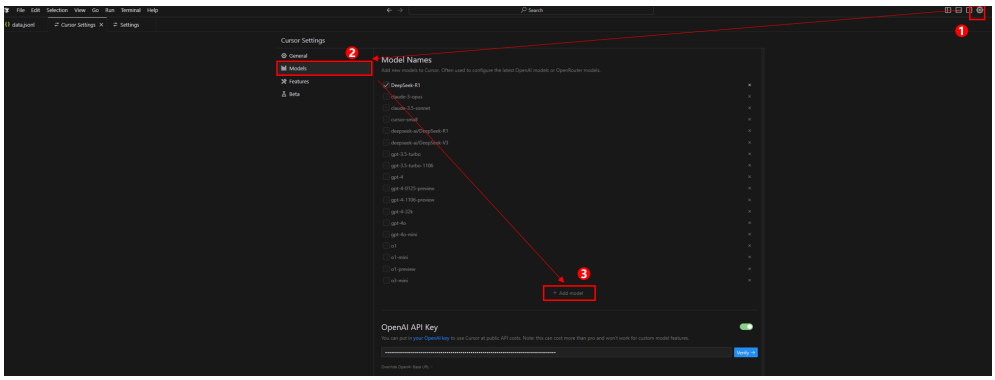
c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.

d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cursor.

Step 3: Configuring the MaaS API in Cursor

1. Click the settings icon in the upper right corner of the Cursor platform.
2. In the **Cursor Settings** area, choose **Models** and click **Add model**.

Figure 3-5 Adding a model



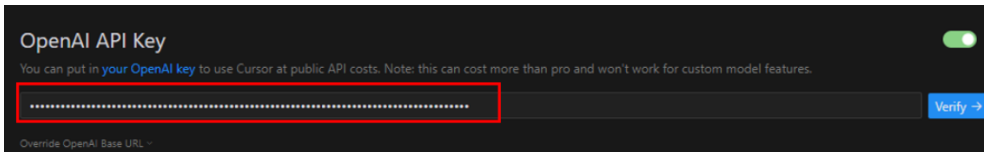
3. Enter the model name obtained in **Step 2.2** in the text box and click **Add model** on the right.

4. Select only the added MaaS model. Choosing others might cause the verification to fail.

Figure 3-6 Selecting a MaaS model

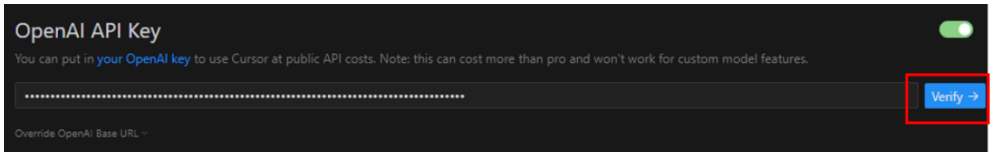
5. In the **OpenAI Key** area, enter the API key created in **Step 2.1**.

Figure 3-7 Entering the API key



6. Click **Override Openai Base URL**. Change the basic API URL to the one from **Step 2.2**, removing **/chat/completions** at the end. Then click **Save**.
7. Click **Verify** to verify the API connectivity. If no errors show, the setup is complete and you can start using the API.

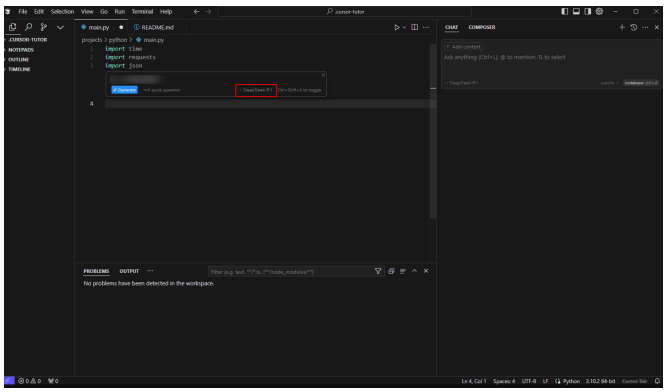
Figure 3-8 Verifying connectivity



Step 4: Using the MaaS API in Cursor to Generate Code

On the code editing page, choose the configured model in the red box for tasks like dialog, code generation, and code parsing.

Figure 3-9 Using the MaaS API



3.3 Quickly Building an AI Coding Assistant Using a ModelArts Studio (MaaS) DeepSeek API and Cline

This guide describes how to use Cline to call a DeepSeek model deployed on ModelArts Studio to build an AI coding assistant.

Operation Scenarios

Cline is a VS Code plugin that uses large language models (LLMs) to handle complex software development tasks. It offers a convenient and efficient coding experience. Advantages of Cline:

- Deep integration with MaaS: Cline connects to DeepSeek model services on MaaS.
- File management and code correction: You can easily create and edit files while monitoring real-time linter and editor errors. Cline detects issues like missing imports or syntax errors, suggests fixes, and improves your coding efficiency.
- Terminal interaction and instant response: Cline includes a terminal for running commands and viewing outputs in real time. It helps you quickly find and fix server issues, keeping development smooth.
- One-stop web solution: For web tasks, Cline starts websites in a headless browser, simulates user actions, and captures screenshots and logs. This helps identify and correct runtime and visual errors, ensuring high-quality web applications.

Prerequisites

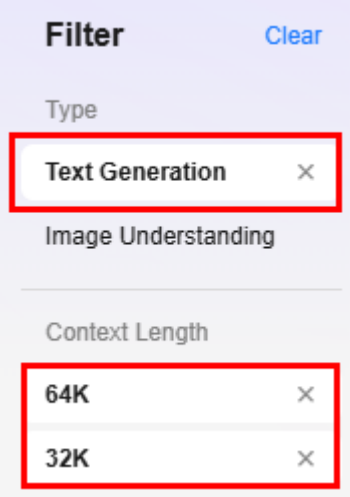
- You have created a Huawei Cloud account. For details, see **Signing Up for a HUAWEI ID and Enabling Huawei Cloud Services**.
- You have completed ModelArts agency authorization. For details, see **Configuring ModelArts Studio (MaaS) Access Authorization**.

Supported Models

The system supports text generation models with a context length of at least 32K.

You can log in to the **ModelArts Studio (MaaS) console**. In the **Model Filtering** area of Model Square, choose **Text Generation** for **Type** and **32K** and **64K** for **Context Length** to view the supported models.

Figure 3-10 Accessing supported models in the Model Square

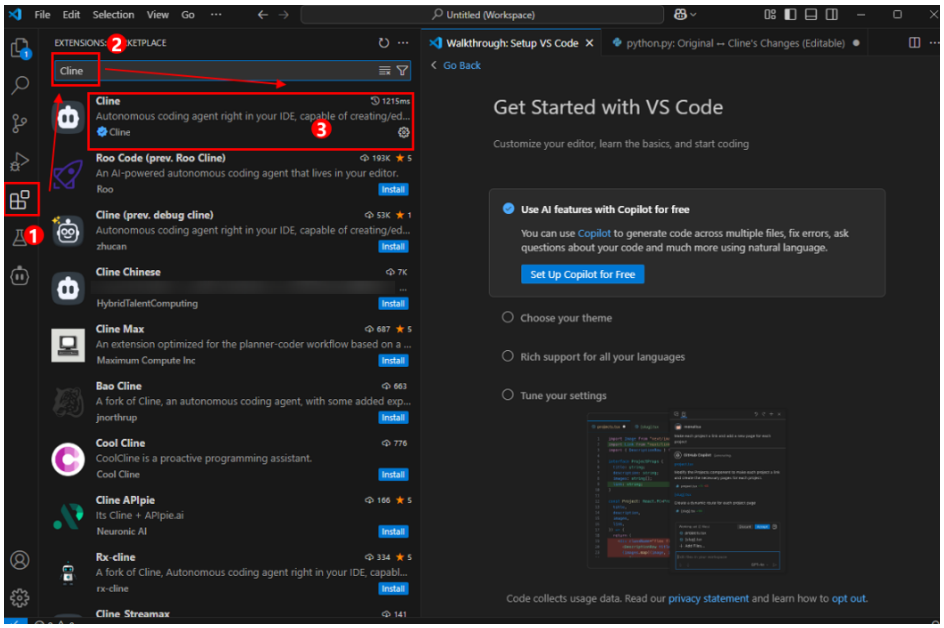


Step 1: Installing Cline in VS Code

1. Open VS Code. Click  on the navigation pane, enter **Cline** in the search box, and click **Install**.

   If you see a small robot icon  on the left, Cline is installed.

   Figure 3-11 Installing Cline



Step 2: Preparing for MaaS Model API Access

1. Create an API key.

   Each key is displayed only once after creation. Keep it secure. If the key is lost, it cannot be retrieved. In this case, create a new API key.

   a. Log in to the **ModelArts Studio (MaaS) console** and select **CN-Hong Kong** from the top navigation bar.

   b. In the navigation pane, choose **API Key Management**.

   c. On the **API Key Management** page, click **Create API Key**, enter the tag and description, and click **OK**.

      The tag and description cannot be modified after the key is created.

      Table 3-4 Parameters

Parameter	Description
Tag	Tag of the API key. The tag must be unique. The tag can contain 1 to 100 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Description	Description of the custom API key. The value can contain 1 to 100 characters.

   d. In the **Your Key** dialog box, copy the key and store it securely.

   e. After the key is saved, click **Close**.

      After you click **Close**, the key cannot be viewed again.

2. Use a model from **My Services**.

   The following uses **My Services** as an example. You can also use commercial services to call APIs. For details, see **Subscribing to a Built-in Commercial Service in ModelArts Studio (MaaS)**.

   a. In the navigation pane of **ModelArts Studio (MaaS) console**, choose **Real-Time Inference**.

   b. Click the **My Services** tab and click **Deploy Model** in the upper right corner to create a model service. For details, see **Deploying a Model Service in ModelArts Studio (MaaS)**.

   c. Choose **More** > **View Call Description** in the **Operation** column of the target running model service.

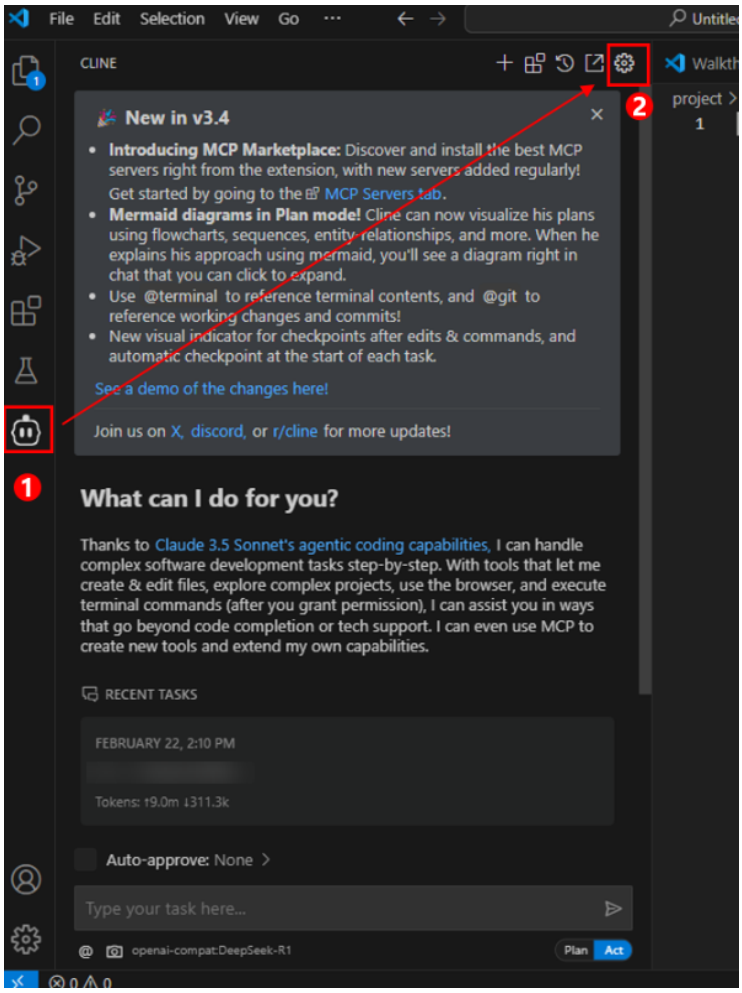
   d. Check the basic API URL and model name on the page. You will need these for the next steps in configuring Cline.

Step 3: Configuring the MaaS API in Cline

1. Configure the MaaS model service.

   a. Open VS Code, click  in the navigation pane to open the Cline plug-in, and click  in the upper right corner.

Figure 3-12 Opening the cline plug-in



b. On the **Settings** page, configure related information and click **Done**.

Table 3-5 Cline configuration

Parameter	Description
API Provider	Select **OpenAI Compatible**.
Base URL	Remove **/chat/completions** from the API URL obtained in **Step 2.2**
API Key	Select the API key created in **Step 2.1**.
Model ID	Enter the model name obtained in **Step 2.2**.

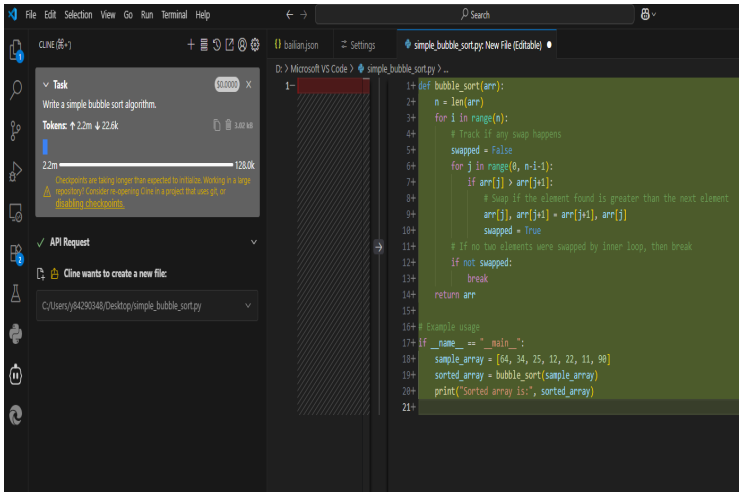
2. Use the Cline plug-in in VS Code to call the MaaS API for automatic code generation.

a. In the navigation pane of VS Code, click  .

b. Select the configured MaaS service shown in the red box in the lower left corner to generate code.

   Cline can generate, correct, and optimize code.

Figure 3-13 Code generation example



4 LLM Inference

4.1 Adapting Mainstream Open-Source Models to Ascend-vLLM for NPU Inference Based on Lite Server (New)

4.1.1 Introduction to Ascend-vLLM

Overview

vLLM is a well-known GPU-based framework for foundation model inference. Its popularity stems from features like continuous batching and PageAttention. Additionally, vLLM supports speculative inference and automatic prefix caching, making it valuable for both academic and industrial applications.

Ascend-vLLM is an inference framework optimized for NPUs. It inherits the advantages of vLLM with boosting the performance and usability through specific optimizations. This makes running foundation models on NPUs more efficient and convenient, significantly enhancing user convenience and performance. Ascend-vLLM can be extensively applied across various foundation model inference tasks, particularly in scenarios requiring high performance and efficiency, such as natural language processing and multimodal understanding.

Highlights

1. **Ease of use**: Ascend-vLLM makes deploying and running foundation models simpler for developers.
2. **Easy development**: An intuitive interface makes it easier to develop and debug models, making adjustments and optimizations straightforward.
3. **High performance**: Huawei's advanced features and optimizations for NPUs boost efficiency through techniques like PD aggregation, preprocessing, postprocessing, and sampling.

Supported Features

Table 4-1 Features supported by Ascend-vLLM

Feature		Description
Scheduling	Page-attention	Manages KV cache in chunks to improve throughput.
	Continuous batching	Iterative scheduling and dynamic batch adjustment reduce the latency and improve the throughput.
Quantification	W4A16-AWQ	INT4 weight quantization cuts down on video memory usage and speeds up processing. It boosts performance for low-concurrency tasks by 80% with minimal accuracy loss under 2%.
	W8A8-SmoothQuant	Int8 weight quantization cuts video memory usage, boosts throughput by 30%, and limits precision loss to under 1.5%.
Efficient decoding	Auto-prefix-caching	Caching the prefix speeds up generating the first token. This helps significantly when using lengthy system prompts or handling multi-turn conversations.
	Chunked-prefill	It is also called SplitFuse, which enhances resource efficiency and boosts performance by combining full and incremental inference.
	Speculative Decoding	It boosts inference speed by supporting both large and small model speculation and eager-mode speculative execution.
Graph mode	ascend-turbo-graph	It tracks how operators depend on each other during execution, removes Python host latency, and handles dynamic shapes. By default, it uses **acl_graph** if no setting is provided. If unsupported, it switches back to eager mode.
	acl-graph	It compares performance with cuda-graph's piece-wise graph. When eager mode is enabled, it takes precedence.
Output control	Guided Decoding	It controls model outputs in a specific mode.
	Beam search	It outputs multiple candidate results through beam search.

4.1.2 Supported Models

Table 4-2 Supported LLMs and their weight download addresses

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
1	DeepSeek-R1-Distill-Llama-8B	√	x	x	x	x	x	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-8B
2	DeepSeek-R1-Distill-Llama-70B	√	x	x	x	x	x	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B
3	DeepSeek-R1-Distill-Qwen-1.5B	√	x	x	x	√	√	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
4	DeepSeek-R1-Distill-Qwen-7B	√	x	x	x	√	√	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
5	DeepSeek-R1-Distill-Qwen-14B	√	x	x	x	√	√	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
6	DeepSeek-R1-0528-Qwen3-8B	√	x	x	x	√	√	v1	https://huggingface.co/deepseek-ai/DeepSeek-R1-0528-Qwen3-8B

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
7	glm-4-9b	√	x	x	x	x	x	v1	https://huggingface.co/THUDM/glm-4-9b-chat
8	llama3-8b	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
9	llama3-70b	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
10	llama3.1-8b	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
11	llama3.1-70b	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
12	llama-3.2-1B	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
13	llama-3.2-3B	√	x	x	x	x	x	v1	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
14	qwen2-0.5b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
15	qwen2-1.5b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
16	qwen2-7b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2-7B-Instruct
17	qwen2-72b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2-72B-Instruct
18	qwen2.5-0.5b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct
19	qwen2.5-1.5b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-1.5B-Instruct
20	qwen2.5-3b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-3B-Instruct

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
21	qwen2.5-7b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
22	qwen2.5-14b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
23	qwen2.5-32b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-32B-Instruct
24	qwen2.5-72b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
25	qwen3-0.6b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-0.6B
26	qwen3-1.7b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-1.7B
27	qwen3-4b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-4B
28	qwen3-8b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-8B

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
29	qwen3-14b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-14B
30	qwen3-30b-a3b	√	x	x	x	√	x	v1	https://huggingface.co/Qwen/Qwen3-30B-A3B
31	qwen3-32b	√	√	√	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-32B
32	qwen3-235b-a22b	√	x	x	x	√	x	v1	https://huggingface.co/Qwen/Qwen3-235B-A22B
33	QwQ-32B	√	x	x	x	√	√	v1	https://huggingface.co/Qwen/QwQ-32B
34	Qwen3-Coder-480B-A35B	√	x	x	x	√	√	v1	https://huggingface.co/Qwen/Qwen3-Coder-480B-A35B-Instruct
35	Qwen3-Embedding-0.6B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Embedding-0.6B
36	Qwen3-Embedding-4B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Embedding-4B
37	Qwen3-Embedding-8B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Embedding-8B

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
38	Qwen3-Reranker-0.6B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Reranker-0.6B
39	Qwen3-Reranker-4B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Reranker-4B
40	Qwen3-Reranker-8B	√	x	x	x	x	√	v0	https://huggingface.co/Qwen/Qwen3-Reranker-4B
41	bge-reranker-v2-m3	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-reranker-v2-m3
42	bge-base-en-v1.5	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-base-en-v1.5
43	bge-base-zh-v1.5	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-base-zh-v1.5
44	bge-large-en-v1.5	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-large-en-v1.5
45	bge-large-zh-v1.5	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-large-zh-v1.5

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	kv-cache-int8 Quantization Supported	Ascend_turbo graph Supported	Acl_graph Supported	v0/v1 Backend	Address for Obtaining the Open-Source Weight
46	bge-m3	√	x	x	x	x	√	v0	https://huggingface.co/BAAI/bge-m3

Table 4-3 Supported multimodal models and their weight download addresses

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	W8A16 Quantization Supported	kv-cache-int8 Quantization Supported	Address for Obtaining the Open-Source Weight	Remarks
1	qwen2-vl-2B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main	-
2	qwen2-vl-7B	√	x	x	x	x	https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main	-

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	W8A16 Quantization Supported	kv-cache-int8 Quantization Supported	Address for Obtaining the Open-Source Weight	Remarks
3	qwen2-vl-72B	√	√	x	x	x	**https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct/tree/main** **https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct-AWQ**	The awq version only supports the eager mode. --enforce-eager
4	qwen2.5-vl-7B	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct/tree/main**	-
5	qwen2.5-vl-32B	√	x	x	x	x	**https://huggingface.co/Qwen/Qwen2.5-VL-32B-Instruct/tree/main**	-

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	W8A16 Quantization Supported	kv-cache-int8 Quantization Supported	Address for Obtaining the Open-Source Weight	Remarks
6	qwen2.5-vl-72B	√	√	x	x	x	**https://huggingface.co/Qwen/Qwen2.5-VL-72B-Instruct/tree/main** **https://huggingface.co/Qwen/Qwen2.5-VL-72B-Instruct-AWQ/tree/main**	The awq version only supports the eager mode. --enforce-eager
7	internvl2.5-26B	√	x	x	x	x	**https://huggingface.co/OpenGVLab/InternVL2_5-26B/tree/main**	-

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	W8A16 Quantization Supported	kv-cache-int8 Quantization Supported	Address for Obtaining the Open-Source Weight	Remarks
8	internvl2-llama3-76B-awq	√	x	x	x	x	**https://huggingface.co/OpenGVLab/InternVL2-Llama3-76B-AWQ/tree/main**	The awq version only supports the eager mode. --enforce-eager
9	gemma3-27B	√	x	x	x	x	**https://huggingface.co/google/gemma-3-27b-it/tree/main**	-
10	internvl3-8B	√	x	x	x	x	**https://huggingface.co/OpenGVLab/InternVL3-8B/tree/main**	-
11	internvl3-14B	√	x	x	x	x	**https://huggingface.co/OpenGVLab/InternVL3-14B/tree/main**	-

No.	Model	FP16/BF16 Inference Supported	W4A16 Quantization Supported	W8A8 Quantization Supported	W8A16 Quantization Supported	kv-cache-int8 Quantization Supported	Address for Obtaining the Open-Source Weight	Remarks
12	internvl3-38B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL3-38B/tree/main	-
13	internvl3-78B	√	x	x	x	x	https://huggingface.co/OpenGVLab/InternVL3-78B/tree/main	-

NOTE

For details about the number of PUs supported by each model, see **Minimum Number of PUs and Maximum Sequence Length Supported by Each Model**.

4.1.3 Minimum Number of PUs and Maximum Sequence Length Supported by Each Model

The table below describes the minimum number of NPUs supported by different models and the max-model-len length when the inference service is deployed based on vLLM.

The values are obtained when **gpu-memory-utilization** is set to **0.95**. They are the minimum number of NPUs required for service deployment and the recommended maximum length of **max-model-len** based on the number of PUs, and do not represent the optimal performance.

For Qwen3-14b, if the NPU has 64 GB of memory, you need at least one NPU for inference. With a single NPU, set **max-model-len** to **32K** (where 1K equals 1,024). This means 32 × 1,024.

Test method: When the value of **gpu-memory-utilization** is **0.95**, set the **max-model-len** value to **4K**, then **8K**, and finally **16K** until reaching its highest possible limit in the static benchmark.

Table 4-4 Minimum number of PUs and maximum sequence length supported by vLLM-based inference

No.	Model	64 GB Video Memory	
		Minimum Number of PUs	Maximum Sequence Length (K) max-model-len
1	DeepSeek-R1-Distill-Llama-8B	1	32
2	DeepSeek-R1-Distill-Llama-70B	4	32
3	DeepSeek-R1-Distill-Qwen-1.5B	1	32
4	DeepSeek-R1-Distill-Qwen-7B	1	32
5	DeepSeek-R1-Distill-Qwen-14B	1	32
6	DeepSeek-R1-0528-Qwen3-8B	1	32
7	glm-4-9b	1	32
8	llama3-8b	1	32
9	llama3-70b	4	32
10	llama3.1-8b	1	32
11	llama3.1-70b	4	32
12	llama-3.2-1B	1	32
13	llama-3.2-3B	1	32
14	qwen2-0.5b	1	32
15	qwen2-1.5b	1	32
16	qwen2-7b	1	32
17	qwen2-72b	4	32
18	qwen2.5-0.5b	1	32
19	qwen2.5-1.5b	1	32
20	qwen2.5-3b	1	32
21	qwen2.5-7b	1	32
22	qwen2.5-14b	1	32
23	qwen2.5-32b	2	32
24	qwen2.5-72b	4	32

25	qwen3-0.6b	1	32
26	qwen3-1.7b	1	32
27	qwen3-4b	1	32
28	qwen3-8b	1	32
29	qwen3-14b	1	32
30	qwen3-30b-a3b	2	32
31	qwen3-32b	2	32
32	qwen3-235b-a22b	16	64
33	QwQ-32B	2	32
34	bge-reranker-v2-m3	1	8
35	bge-base-en-v1.5	1	0.5
36	bge-base-zh-v1.5	1	0.5
37	bge-large-en-v1.5	1	0.5
38	bge-large-zh-v1.5	1	0.5
39	bge-m3	1	8
40	qwen2-vl-2B	1	8
41	qwen2-vl-7B	1	32
42	qwen2-vl-72B	4	32
43	qwen2.5-vl-7B	1	**32**
44	qwen2.5-vl-32B	1	32
45	qwen2.5-vl-72B	4	**48**
46	internvl2.5-26B	1	8
47	InternVL2-Llama3-76B-AWQ	2	8
48	gemma3-27B	1	4
49	Qwen3-Embedding-0.6B	1	32
50	Qwen3-Embedding-4B	1	40
51	Qwen3-Embedding-8B	1	40
52	Qwen3-Reranker-0.6B	1	40
53	Qwen3-Reranker-4B	1	40
54	Qwen3-Reranker-8B	1	40
55	Qwen3-Coder-480B-A35B	32	64

56	internvl3-8B	1	16
57	internvl3-14B	1	16
58	internvl3-38B	2	16
59	internvl3-78B	4	32

4.1.4 Version Description and Requirements

Version Differences

Use this guide for ModelArts version 6.5.906 or newer. The latest version is 6.5.907. You are advised to use the latest software package and image.

Table 4-5 Version differences

Version	Description
6.5.907	Compared with 6.5.906, 6.5.907 has the following changes: 1. LLM inference framework: Qwen3-Embedding series, Qwen3-Reranker series, and Qwen3-Coder-480B-A35B are added. 2. Multimodal inference framework: InternVL3 series and Qwen2.5 VL support 128K sequences. 3. Some stability issues in version 6.5.906 are resolved.

Resource Specifications

In this document, the model runtime environment is ModelArts Lite Server. Snt9b and Snt9b23 resources are recommended.

Enable Lite Server resources and obtain passwords. Verify SSH access to all servers. Confirm proper network connectivity between them.

NOTE

If a container is used or shared by multiple users, you should restrict the container from accessing the OpenStack management address (169.254.169.254) to prevent host machine metadata acquisition. For details, see **Forbidding Containers to Obtain Host Machine Metadata**.

Ascend-vLLM Version

This solution supports vLLM v0.9.0.

Image Version

The table below lists the base image addresses and their versions for this tutorial.

Table 4-6 Base image addresses

Usage	Address	Version
Snt9b base image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	Cann: CANN 8.2.RC1 PyTorch: pytorch_2.5.1
Snt9b23 base image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129	

Software Version

Table 4-7 lists the supported software versions and dependency packages.

Table 4-7 Software packages

Software Package Name	Description	How to Obtain
AscendCloud-6.5.907-20250910155849.zip	Inference framework and operator code package (suitable for Snt9b)	Download ModelArts 6.5.907 from **Support-E**. **NOTE** If the software information does not appear when opening the download link, you lack access permissions. Contact your company's Huawei technical support for assistance with downloading.
AscendCloud-6.5.907-20250910161027.zip	Inference framework and operator code package (suitable for Snt9b23)	

Software Package Structure

In this tutorial, the **AscendCloud-LLM-*xxx*.zip** software package in **AscendCloud-*xxx*** stores the key files of the inference code.


```
├──AscendCloud-LLM
 ├──llm_inference  # Inference code
  ├──ascend_vllm
      ├── ascend_vllm             # Inference source code
      ├── install.sh              # Installation script
      ├── version.info            # Version information
      ├── Dockerfile              # Dockerfile for inference build image
      ├── vllm_list.patch         # Incremental patch for inference based on vLLM
      ├── vllm_service_profile.patch   # Incremental patch for inference based on vLLM
      ├── vllm_serving_chat.patch      # Incremental patch for inference based on vLLM
      ├── vllm-log-rotating.patch      # Incremental patch for inference based on vLLM
 ├──llm_tools          # Inference tool package
  ├──best_practices    # Best practice package
  ├──launch_server      # One-click startup script
  ├──llm_evaluation     # MME accuracy evaluation tool
  ├──PD_separate        # PD aggregation
  ├──simple_evals       # Accuracy evaluation tool
  ├──acs_bench-1.0.1-py3-none-any.whl     # Benchmark performance test tool package
  ├──acs_service_profiler-1.0.1-py3-none-any.whl    # Service profiling collection tool package
```

4.1.5 Inference Service Deployment

4.1.5.1 Preparing the Inference Environment

Prerequisites

- You have prepared Lite Server resources. For details, see **Resource Specifications**. Snt9b or Snt9b23 resources are recommended.
- The container connects to the public network. Git clone needs internet access during installation.

Step 1: Checking the Environment

1. Log in to the server via SSH and check the NPU status. Obtain the NPU device information:

   ```
   npu-smi info                 # Run this command on each instance node to view the NPU status.
   npu-smi info -l | grep Total    # Run this command on each instance node to view the total number of PUs and check whether these PUs have been mounted.
   npu-smi info -t board -i 1 | egrep -i "software|firmware"   # Check the driver and firmware versions.
   ```

   If an error occurs, the NPU devices on the server may not be properly installed, or the NPU image may be mounted to another container. Install the firmware and driver or release the mounted NPUs.

   Table 4-8 lists the driver version requirements. If the requirements are not met, upgrade the driver by referring to **Configuring the Software Environment on the NPU Server**.

   Table 4-8 HDK firmware & driver

NPU	HDK Firmware & Driver
Snt9b23 (standard)	25.2.1
Snt9b (standard)	25.2.1

2. Check whether Docker is installed.

   ```
   docker -v   # Check whether Docker is installed.
   ```

   If Docker is not installed, run this command:


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

3. Configure IP forwarding for intra-container network accesses. Run the following command to check the value of **net.ipv4.ip_forward**. Skip this step if the value is **1**.

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

If the value is not **1**, configure IP forwarding:

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

Step 2: Obtaining the Base Image

Use official images to deploy inference services. For details about the image path *{image_url}*, see **Table 4-6**.

```
docker pull {image_url}
```

Step 3: Uploading the Code Package and Weight File

1. Upload the inference code package *AscendCloud-LLM-xxx.zip* and operator package *AscendCloud-OPP-xxx.zip* to the host. For details about how to obtain the packages, see **Table 4-7**.
2. Upload the weight file to the Lite Server. The weight file must be in Hugging Face format. For details about how to obtain the open-source weight file, see **Supported Models**.
3. Save the weight file in the chosen folder on your disk and verify its size using this sample command:

```
df -h
```

Step 4: Creating an Inference Image

Run the script below to decompress the package. Ensure that the environment is connected to the network and supports git clone.

```
# Move the package to the build directory and decompress the package in sequence to build a production image.
unzip AscendCloud-*.zip -d AscendCloud && cd AscendCloud \
&& unzip AscendCloud-OPP-*.zip && unzip AscendCloud-OPP-*-torch-*-py*.zip -d ./AscendCloud-OPP \
&& unzip AscendCloud-LLM*.zip -d ./AscendCloud-LLM \
&& cp AscendCloud-LLM/llm_inference/ascend_vllm/Dockerfile . \
&& docker build -t ${image} --build-arg BASE_IMAGE=$base_image .
```

Parameters:

- *${base_image}*: base image name, that is, *{image_url}*. The value is obtained from **Table 4-6**.
- *$image*: name of the newly created image. You can customize this name, but it must follow the format "image_name:image_tag". For example:

  pytorch_ascend:pytorch_2.5.1-cann_8.1.rc2-py_3.11-hce_2.0.2503-aarch64-snt9b23

After the execution is complete, the image required for inference is generated. Run **docker images** to find the generated image.

Step 5: Starting the Container

Before starting the container image, modify the parameters in *${}* according to the parameter description. If Docker fails to start, there will be a corresponding error

prompt. If it starts successfully, the corresponding Docker ID will be generated without any errors.

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

Parameters:

- --device=/dev/davinci0, ..., --device=/dev/davinci7: mounts NPUs. In the example, eight NPUs (davinci0 to davinci7) are mounted.
- **-v** *${dir}*:*${container_model_path}*: host directory to be mounted to the container.

  NOTE

  - The **/home/ma-user** directory cannot be mounted to the container. This directory is the home directory of user **ma-user**. If the container is mounted to **/home/ma-user**, the container conflicts with the base image when being started. As a result, the base image is unavailable.
  - Both the driver and npu-smi must be mounted to the container.
  - Avoid assigning multiple containers to one NPU. Doing so will prevent its use in later containers.
- **--name** *${container_name}*: container name, which is used when you access the container. You can define a container name.
- *{image_id}* indicates the ID of the Docker image, that is, the ID of the new image generated in **Step 4: Creating an Inference Image**. You can run the **docker images** command on the host to query the ID.

Step 6: Entering the Container

1. Access the container.
   ```
   docker exec -it -u ma-user ${container_name} /bin/bash
   ```
2. Evaluate inference resources. Run the following command in the container to retrieve the number of available NPUs:
   ```
   npu-smi info      # Before starting the inference service, ensure that the NPUs are not occupied, ports are free, and relevant processes are running.
   ```
   If an error occurs, the NPU devices on the server may not be properly installed, or the NPU image may be mounted to another container. Install the firmware and driver or release the mounted NPUs.

If the requirements are not met, upgrade the driver by referring to **Configuring the Software Environment on the NPU Server**. After the container is started, the default port number is 8080.

3. Specify which NPUs within the container should be used. For example, if the first NPU in the container is being used, set it to **0**:

```
export ASCEND_RT_VISIBLE_DEVICES=0
```

If multiple NPUs are needed for the service, arrange them sequentially according to their numbering in the container. For example, if the first and second NPUs are used, set it to **0,1**:

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

Run the **npu-smi info** command to determine the order of NPUs in the container. For example, if two NPUs are detected and you wish to use the first and second ones, set **export ASCEND_RT_VISIBLE_DEVICES=0,1**. Be careful that the indices are not necessarily 4 or 5.

Figure 4-1 Query result

```
+------------------------------------------------------------------------------------------------+
| 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                   |
+===========================+===============+====================================================+
```

For details about how to start an inference service, see **Starting an LLM-powered Inference Service**.

4.1.5.2 Starting an LLM-powered Inference Service

This topic describes how to start an LLM-powered inference service, including offline inference and online inference.

CAUTION

You must set these common framework environment variables for both offline and online inference.

The common framework environment variables are as follows:

```
# VPC CIDR block
# You need to manually change the values. For details, see the following instructions.
VPC_CIDR="7.150.0.0/16"
VPC_PREFIX=$(echo "$VPC_CIDR" | cut -d'/' -f1 | cut -d'.' -f1-2)
POD_INET_IP=$(ifconfig | grep -oP "(?<=inet\s)$VPC_PREFIX\.\d+\.\d+" | head -n 1)
POD_NETWORK_IFNAME=$(ifconfig | grep -B 1 "$POD_INET_IP" | head -n 1 | awk '{print $1}' | sed 's/://')
echo "POD_INET_IP: $POD_INET_IP"
echo "POD_NETWORK_IFNAME: $POD_NETWORK_IFNAME"
# Specify the NIC.
export GLOO_SOCKET_IFNAME=$POD_NETWORK_IFNAME
```



```
export TP_SOCKET_IFNAME=$POD_NETWORK_IFNAME
export HCCL_SOCKET_IFNAME=$POD_NETWORK_IFNAME
# Set this parameter in the multi-node scenario.
export RAY_EXPERIMENTAL_NOSET_ASCEND_RT_VISIBLE_DEVICES=1

# Enable video memory optimization.
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
# Configure the expansion location of the communication algorithm to the AI Vector Core unit on the device side.
export HCCL_OP_EXPANSION_MODE=AIV
# Specify available PUs as required.
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3
# Specify CPU core binding as needed.
export CPU_AFFINITY_CONF=1
export LD_PRELOAD=/usr/local/lib/libjemalloc.so.2:${LD_PRELOAD}
# ascend-turbo-graph is enabled by default; specify the plug-in as ascend_vllm, otherwise ascend is used.
export VLLM_PLUGINS=ascend_vllm
# Specify whether to use ACLGraph mode: set the parameter to 1 to enable, and 0 to disable.
export USE_ACLGRAPH=0
# Set the vLLM backend version to v1.
export VLLM_USE_V1=1
# Specify the vLLM version.
export VLLM_VERSION=0.9.0
```

Offline Inference

Edit a Python script containing this code. Execute the script to run offline model inference using ascend-vllm.

```
from vllm import LLM, SamplingParams

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

    model_path = "/path/to/model"
    llm = LLM(
        model=model_path,
        tensor_parallel_size=2,
        block_size=128,
        max_num_seqs=256,
        max_model_len=8192,
        additional_config={
            "ascend_turbo_graph_config": {
                "enabled": True
            },
            "ascend_scheduler_config": {
                "enable": True,
                "chunked_prefill_enabled": False
            }
        },
        distributed_executor_backend='ray'
    )

    outputs = llm.generate(prompts, sampling_params)

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

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


Starting a Real-Time Inference Service

This section uses the OpenAI service API. For details, see **https://docs.vllm.ai/en/latest/getting_started/quickstart.html**.

Use the OpenAI API for inference. Below are the commands for single-node with one PU or multiple PUs. Adjust settings using the provided parameters.

NOTE

The table below shows key performance enhancement settings for Qwen series models, alongside common framework environment variables.

Qwen2, Qwen2.5, and Qwen3 LLMs

- The Qwen MoE model cannot use the Qwen series optimization settings listed in Table 1.
- The ACLGraph and eager modes cannot use the Qwen series optimization settings listed in Table 1.
- The Qwen series W4A16 quantization models work exclusively with the AscendTurbo graph mode and cannot be configured using Qwen series optimization environment variables.

1. For Qwen2, Qwen2.5, or Qwen3 models, use the ascend-turbo-graph mode as the default setting in the inference service startup parameters. You must also configure specific environment variables for Qwen models.
2. Use eager mode for Meta-Llama or Llama-like models.

Table 4-9 Environment variables for starting the Qwen Dense series

Variable	Description
ENABLE_QWEN_HYPERDRIVE_OPT	This function is disabled by default. The flashcomm communication optimization combined with the general fused operator optimization (TDynamicquant) applies to the entire Qwen model series. For BF16 scenarios, it must be used together with DISABLE_QWEN_DP_PROJ. W8A8 is not affected.
ENABLE_QWEN_MICROBATCH	This function is disabled by default. This enables micro-batch optimization and must be used together with ENABLE_QWEN_HYPERDRIVE_OPT. Applicable to all Qwen models in W8A8 and BF16 modes.

Variable	Description
ENABLE_PHASE_AWARE_QKVO_QUANT	This function is disabled by default. At runtime, it introduces BF16 weights and performs mixed-precision quantized inference, which may increase memory usage. Applicable to all Qwen models in W8A8 mode; not supported in BF16 mode. Must be used together with **ENABLE_QWEN_HYPERDRIVE_OPT**.
DISABLE_QWEN_DP_PROJ	This function is disabled by default. When **export ENABLE_QWEN_HYPERDRIVE_OPT=1** is set, **DISABLE_QWEN_DP_PROJ takes** effect. It disables full weight loading for mlp down_proj. Recommended to disable in Qwen BF16 scenarios; can be enabled in W8A8 scenarios.

Qwen Dense Models Supported	Environment Variable Example	Remarks
Qwen2 series – BF16 Qwen2.5 series – BF16 Qwen3 Dense series – BF16	export ENABLE_QWEN_HYPERDRIVE_OPT=1 export ENABLE_QWEN_MICROBATCH=1 export DISABLE_QWEN_DP_PROJ=1	-

Qwen Dense Models Supported	Environment Variable Example	Remarks
Qwen2 series – W8A8 Qwen2.5 series – W8A8 Qwen3 Dense series – W8A8	export ENABLE_QWEN_HYPERDRIVE_OPT=1 export ENABLE_QWEN_MICROBATCH=1 export ENABLE_PHASE_AWARE_QKVO_QUANT=0 export DISABLE_QWEN_DP_PROJ=0	• export ENABLE_PHASE_AWARE_QKVO_QUANT=1 (set it to 1 only for Qwen3-32b-w8a8 tp8) • When BF16 is used, set **export DISABLE_QWEN_DP_PROJ=1** to disable weight copy optimization. • When W8A8 is used, the default setting is **DISABLE_QWEN_DP_PROJ=0**. When the server log indicates prolonged GPU KV cache usage above 90% and a sharp rise in TTFT values, disable weight copy optimization by setting **export DISABLE_QWEN_DP_PROJ=1**.

Inference service startup parameters:

```
source /home/ma-user/AscendCloud/AscendTurbo/set_env.bash

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=0.95 \
--trust-remote-code \
--no-enable-prefix-caching \
--additional-config='{"ascend_turbo_graph_config": {"enabled": true}, "ascend_scheduler_config": {"enabled": true}}'
```

The basic inference service parameters are as follows:

- --model ${container_model_path}: path to the model weights within the container. It uses the Hugging Face directory format, which stores the model's weight files. If quantization is used, the weights converted in **Quantization**

are used. If the address for converting the trained model into the Hugging Face format is used, the original tokenizer file is required.

- --quantization, -q: weight quantization method taken from **quantization_config** in the model's configuration file. This parameter is required if the model uses quantization.
- **--max-num-seqs**: maximum number of concurrent requests that can be processed. Requests exceeding this limit will wait in the queue.
- **--max-model-len**: maximum number of input and output tokens during inference. The value of **max-model-len** must be less than that of **seq_length** in the **config.json** file. Otherwise, an error will be reported during inference and prediction. The **config.json** file is stored in the path corresponding to the model, for example, *${container_model_path}*/**chatglm3-6b/config.json**. The maximum length varies among different models. For details, see **Table 4-4**.
- **--max-num-batched-tokens**: maximum number of tokens that can be used in the prefill phase. The value must be greater than or equal to the value of **--max-model-len**. The value **4096** or **8192** is recommended.
- **--dtype**: data type for model inference, which can be FP16 or BF16. **float16** indicates FP16, and **bfloat16** indicates BF16. If unspecified, the data type is auto-matched based on input data. Using different **dtype** may affect model precision. When using open-source weights, you are advised not to specify **dtype** and instead use the default **dtype** of the open-source weights.
- **--tensor-parallel-size**: number of PUs you want to use. The product of model parallelism and pipeline parallelism must be the same as the number of started NPUs. For details, see **Table 4-4**. The value **1** indicates that the service is started using a single PU.
- --pipeline-parallel-size: number of parallel pipelines. The product of model parallelism and pipeline parallelism must be the same as the number of started NPUs. The default value is **1**. Currently, **pipeline-parallel-size** can only be set to **1**.
- **--block-size**: block size of kv-cache. The recommended value is **128**.
- **--host=${docker_ip}**: IP address for service deployment. Replace *${docker_ip}* with the actual IP address of the host machine. The default value is **None**. For example, the parameter can be set to **0.0.0.0**.
- **--port**: port where the service is deployed.
- **--gpu-memory-utilization**: ratio of the GPU memory used by the NPU. The input parameter name of the original vLLM is reused. The recommended value is **0.95**.
- **--trust-remote-code**: Specifies whether to trust the remote code.
- **--distributed-executor-backend**: backend for launching multi-PU inference. The options are **ray** and **mp**, where **ray** indicates that Ray is used for multi-PU inference, and **mp** indicates that Python multi-processing is used for multi-PU inference. The default value is **mp**.
- **--disable-async-output-proc**: Disables asynchronous post-processing, which reduces performance.
- **--speculative-config**: speculative inference parameter, which is a JSON string. The default value is **None**.
- **--no-enable-prefix-caching**: Disables prefix caching. For details about how to enable prefix caching, see **Prefix Caching**.

- **--enforce-eager**: If the **INFER_MODE** environment variable is not set, some models are started in ACLGraph mode by default to improve performance. After this parameter is set, the graph mode is disabled. You are advised to enable this function for non-Qwen series, such as Meta-Llama series.
- **--additional-config**: **{"ascend_turbo_graph_config": {"enabled": true}}** enables the ascend_turbo graph mode. If this function is enabled, the performance of the Qwen series is improved. If this function is disabled, acl_graph is used by default. Currently, acl_graph supports only BF16 and does not support compress_tensors, smoothquant, or awq.

  "ascend_scheduler_config": {"enabled": true} is the configuration option of the scheduler.

Deploying an Inference Service on Multiple Nodes

If the GPU memory of a single node cannot accommodate the model weights, you may choose multi-node deployment. For this deployment method, it is required that the nodes are within the same cluster and the IP addresses between NPUs can be pinged successfully. The specific steps are as follows:

1. Run the following command on one of the nodes to view the IP address of the NPUs:
   ```
   for i in $(seq 0 7);do hccn_tool -i $i -ip -g;done
   ```
2. Check whether the network connection between NPUs is normal.
   ```
   # Run the following command on another node. 29.81.3.172 is the value of ipaddr obtained in the previous step.
   hccn_tool -i 0 -ping -g address 29.81.3.172
   ```
3. Start the Ray cluster.
   ```
   # Specify the communication NIC. Run the ifconfig command to find the NIC name that matches the host IP address.
   export GLOO_SOCKET_IFNAME=enp67s0f5
   export TP_SOCKET_IFNAME=enp67s0f5
   export RAY_EXPERIMENTAL_NOSET_ASCEND_RT_VISIBLE_DEVICES=1

   # Specify the available NPUs.
   export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7

   # Set one node as the head node.
   ray start --head --num-gpus=8

   # Run the following command on other nodes:
   ray start --address='10.170.22.18:6379' --num-gpus=8
   ```
   - --num-gpus: The value must be the same as the number of available NPUs specified by **ASCEND_RT_VISIBLE_DEVICES**.
   - --address: IP address and port number of the head node. This will be printed after the head node is successfully created.

   NOTE
   - Environment variables must be set on each node.
   - To update environment variables, you must restart the Ray cluster.
4. Choose a node, add the distributed backend setting **--distributed-executor-backend=ray**, and configure all other settings to match standard service launch conditions. For details, see the single-node scenario in **Starting a Real-Time Inference Service**.

Inference Request Test

Use commands to test if the inference service has started properly. For details about the parameter settings in the service startup command, see **Starting a Real-Time Inference Service**.

Run the inference test commands below to start the service via the OpenAI service API. Replace *${docker_ip}* with the actual IP address of the host machine. When starting the service, if you omit the **served-model-name** parameter, *${container_model_path}* should match the **model** parameter's value. If you include **served-model-name**, use its value instead of *${container_model_path}*.

OpenAI Completions API with vLLM

```
curl http://${docker_ip}:8080/v1/completions \
-H "Content-Type: application/json" \
-d '{
      "model": "${container_model_path}",
      "prompt": "hello",
      "max_tokens": 32,
      "temperature": 0
}'
```

OpenAI Chat Completions API with vLLM

```
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": 32,
    "temperature": 0
}'
```

The service APIs are identical to those on the vLLM official website. Key parameters are introduced here. For detailed parameter descriptions, refer to the official website at **https://docs.vllm.ai/en/stable/api/vllm/vllm.sampling_params.html**.

For details about the OpenAI service request parameters, see **Table 4-10**.

Table 4-10 OpenAI service request parameters

Parameter	Mandatory	Default Value	Type	Description
model	Yes	None	Str	This parameter is mandatory for an inference request when the service is started through the OpenAI service API. The value must be the same as the value of **model ${container_model_path}** when the inference service is started. This parameter is not involved for an inference request when the service is started through the vLLM service API.
prompt	Yes	-	Str	Input question of the request.
max_tokens	No	16	Int	Maximum number of tokens to be generated for each output sequence.
top_k	No	-1	Int	Determines how many of the highest ranking tokens are considered. The value **-1** indicates that all tokens are considered. Decreasing the value can reduce the sampling time.
top_p	No	1.0	Float	A floating point number that controls the cumulative probability of the first several tokens to be considered. The value must be in the range (0, 1]. The value **1** indicates that all tokens are considered.
temperature	No	1.0	Float	A floating-point number that controls the randomness of sampling. Lower values make the model more deterministic, while higher values make it more random. The value **0** indicates greedy sampling.
stop	No	None	None/Str/List	A list of strings used to stop generation. The output does not contain the stop strings. For example, if "Hello" and "Hi" are configured as the stop sequences, the model stops generating text when "Hello" or "Hi" is encountered.
stream	No	False	Bool	Controls whether to enable streaming inference. The default value is **False**, indicating that streaming inference is disabled.

Parameter	Mandatory	Default Value	Type	Description
n	No	1	Int	Multiple normal results are returned. Constraints: If **beam_search** is not used, the recommended value range of n is 1 ≤ n ≤10. When n exceeds 1, avoid using **greedy_sample** by setting **top_k** greater than 1 and keeping **temperature** above 0. If **beam_search** is used, the recommended value range of n is 1 < n ≤ 10. If n is 1, the inference request will fail. **NOTE** For optimal performance, keep n at 10 or below. Large values of n can significantly slow down processing. Inadequate video RAM may cause inference requests to fail.
use_beam_search	No	False	Bool	Controls whether to use **beam_search** to replace sampling. Constraints: When this parameter is used, the following parameters must be set as required. n>1 top_p = 1.0 top_k = -1 temperature = 0.0 **WARNING** When **beam_search** is used, **max_tokens** must be explicitly set to ensure that the request can be stopped as expected.
presence_penalty	No	0.0	Float	Applies rewards or penalties based on the presence of new words in the generated text. The value range is [-2.0, 2.0].
frequency_penalty	No	0.0	Float	Applies rewards or penalties based on the frequency of each word in the generated text. The value range is [-2.0,2.0].

Parameter	Mandatory	Default Value	Type	Description
length_penalty	No	1.0	Float	Imposes a larger penalty on longer sequences in a beam search process. To use **length_penalty**, you must include the following three parameters: set **use_beam_search** to **true**, set **best_of** to a value greater than 1, and fix **top_k** at **-1**. "top_k": -1 "use_beam_search":true "best_of":2
ignore_eos	No	False	Bool	Indicates whether to ignore EOS and continue to generate tokens.
guided_json	No	None	Union[str, dict, BaseModel]	Use OpenAI to start the service. If JSON Schema is required, set **guided_json**. For details, see **Structured Outputs**.

4.1.5.3 Starting an Embedding or Reranking Model-powered Inference Service

This topic describes how to start an embedding or reranking model-powered inference service, including offline inference and online inference.

CAUTION

You must set these common framework environment variables for both offline and online inference.

The common framework environment variables are as follows:

```
# VPC CIDR block
# You need to manually change the values. For details, see the following instructions.
VPC_CIDR="7.150.0.0/16"
VPC_PREFIX=$(echo "$VPC_CIDR" | cut -d'/' -f1 | cut -d'.' -f1-2)
POD_INET_IP=$(ifconfig | grep -oP "(?<=inet\s)$VPC_PREFIX\.\d+\.\d+" | head -n 1)
POD_NETWORK_IFNAME=$(ifconfig | grep -B 1 "$POD_INET_IP" | head -n 1 | awk '{print $1}' | sed 's/://')
echo "POD_INET_IP: $POD_INET_IP"
echo "POD_NETWORK_IFNAME: $POD_NETWORK_IFNAME"
# Specify the NIC.
export GLOO_SOCKET_IFNAME=$POD_NETWORK_IFNAME
export TP_SOCKET_IFNAME=$POD_NETWORK_IFNAME
export HCCL_SOCKET_IFNAME=$POD_NETWORK_IFNAME
# Set this parameter in the multi-node scenario.
export RAY_EXPERIMENTAL_NOSET_ASCEND_RT_VISIBLE_DEVICES=1

# Enable video memory optimization.
```



```
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
# Configure the expansion location of the communication algorithm to the AI Vector Core unit on the device side.
export HCCL_OP_EXPANSION_MODE=AIV
# Specify available PUs as required.
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3
# Specify CPU core binding as needed.
export CPU_AFFINITY_CONF=1
export LD_PRELOAD=/usr/local/lib/libjemalloc.so.2:${LD_PRELOAD}
# By default, the ACLGraph mode is enabled. Specify the plug-in as ascend.
export VLLM_PLUGINS=ascend
# Specify whether to use ACLGraph mode: set the parameter to 1 to enable, and 0 to disable.
export USE_ACLGRAPH=1
# Set the vLLM backend version to v0.
export VLLM_USE_V1=0
# Specify the vLLM version.
export VLLM_VERSION=0.9.0
```

Offline Inference

Edit an embeddings Python script containing this code. Execute the script to run offline model inference using ascend-vllm.

```
from vllm import LLM, SamplingParams

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

    model_path = "/path/to/model"
    llm = LLM(
        model=model_path,
        tensor_parallel_size=2,
        block_size=128,
        max_num_seqs=256,
        max_model_len=8192,
        distributed_executor_backend='ray'
    )

    outputs = llm.encode(prompts)

    # Print the outputs.
    for output in outputs:
        print(output.outputs.embedding)

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

Starting a Real-Time Inference Service

This section uses the OpenAI service API. For details, see **https://docs.vllm.ai/en/latest/getting_started/quickstart.html**.

Use the OpenAI API for inference. Below are the commands for single-node with one PU or multiple PUs. Adjust settings using the provided parameters.

Inference service startup parameters (Qwen3-Reranker-0.6B is used as an example):

```
source /home/ma-user/AscendCloud/AscendTurbo/set_env.bash

python -m vllm.entrypoints.openai.api_server \
--model /model/Qwen3-Reranker-0.6B \
```



```
--served-model-name Qwen3-Reranker-0.6B \
--max-num-seqs=256 \
--max-model-len=32768 \
--max-num-batched-tokens=32768 \
--tensor-parallel-size=1 \
--block-size=128 \
--host=0.0.0.0 \
--port=9001 \
--gpu-memory-utilization=0.95 \
--trust-remote-code

# hf_overrides needs to be set only for Qwen3 reranker.
# --hf_overrides '{"architectures": ["Qwen3ForSequenceClassification"],"classifier_from_token": ["no", "yes"],"is_original_qwen3_reranker": true}'
```

The basic inference service parameters are as follows:

- --model ${container_model_path}: path to the model weights within the container. It uses the Hugging Face directory format, which stores the model's weight files.
- **--max-num-seqs**: maximum number of concurrent requests that can be processed. Requests exceeding this limit will wait in the queue.
- **--max-model-len**: maximum number of input and output tokens during inference. The value of **max-model-len** must be less than that of **seq_length** in the **config.json** file. Otherwise, an error will be reported during inference and prediction. The **config.json** file is stored in the path corresponding to the model, for example, *${container_model_path}*/**chatglm3-6b/config.json**. The maximum length varies among different models. For details, see **Table 4-4**.
- **--max-num-batched-tokens**: maximum number of tokens that can be used in the prefill phase. The value must be greater than or equal to the value of **--max-model-len**. The value **4096** or **8192** is recommended.
- **--dtype**: data type for model inference, which can be FP16 or BF16. **float16** indicates FP16, and **bfloat16** indicates BF16. If unspecified, the data type is auto-matched based on input data. Using different **dtype** may affect model precision. When using open-source weights, you are advised not to specify **dtype** and instead use the default **dtype** of the open-source weights.
- **--tensor-parallel-size**: number of PUs you want to use. The product of model parallelism and pipeline parallelism must be the same as the number of started NPUs. For details, see **Table 4-4**. The value **1** indicates that the service is started using a single PU.
- --pipeline-parallel-size: number of parallel pipelines. The product of model parallelism and pipeline parallelism must be the same as the number of started NPUs. The default value is **1**. Currently, **pipeline-parallel-size** can only be set to **1**.
- **--block-size**: block size of kv-cache. The recommended value is **128**.
- **--host=${docker_ip}**: IP address for service deployment. Replace *${docker_ip}* with the actual IP address of the host machine. The default value is **None**. For example, the parameter can be set to **0.0.0.0**.
- **--port**: port where the service is deployed.
- **--gpu-memory-utilization**: ratio of the GPU memory used by the NPU. The input parameter name of the original vLLM is reused. The recommended value is **0.95**.
- **--trust-remote-code**: Specifies whether to trust the remote code.

- **--distributed-executor-backend**: backend for launching multi-PU inference. The options are **ray** and **mp**, where **ray** indicates that Ray is used for multi-PU inference, and **mp** indicates that Python multi-processing is used for multi-PU inference. The default value is **mp**.
- **--disable-async-output-proc**: Disables asynchronous post-processing, which reduces performance.
- **--no-enable-prefix-caching**: Disables prefix caching. For details about how to enable prefix caching, see **Prefix Caching**.
- **--enforce-eager**: If the **INFER_MODE** environment variable is not set, some models are started in ACLGraph mode by default to improve performance. After this parameter is set, the graph mode is disabled. You are advised to enable this function for non-Qwen series, such as Meta-Llama series.
- **--hf_overrides**: Overrides the default settings of the Hugging Face model within the vLLM inference engine. Currently, only Qwen3 rerankers need to override some parameters (see the startup description).

Inference Request Test

Use commands to test if the inference service has started properly. For details about the parameter settings in the service startup command, see **Starting a Real-Time Inference Service**.

Run the inference test commands below to start the service via the OpenAI service API. Replace *${docker_ip}* with the actual IP address of the host machine. The **model** parameter is mandatory. The service's **served_model_name** parameter must match its own value if provided. When the service lacks a **served_model_name**, it uses the **model** parameter's value instead, that is the model's weight location within the container.

Example reranking API:

```
curl -X POST http://${docker_ip}:8080/v1/rerank \
    -H "Content-Type: application/json" \
    -d '{
        "model": "/container_model/bge-reranker-v2-m3",
        "query": "What is the capital of France?",
        "documents": [
            "The capital of France is Paris",
            "Reranking is fun!",
            "vLLM is an open-source framework for fast AI serving"
        ]
    }'
```


Table 4-11 Reranking service request parameters

Parameter	Mandatory	Default Value	Type	Description
model	Yes	None	Str	The service's **served_model_name** parameter must match its own value if provided. If the service does not provide **served_model_name**, the value is the same as the **model** parameter of the service.
query	Yes	None	Str	User query text
documents	Yes	None	Str	List of documents to be sorted (usually top-k results of embedding recall)

Use OpenAI to start the service (only V0 is supported). The following is an example of the embeddings API:

```
curl -X POST http://${docker_ip}:8080/v1/embeddings \
    -H "Content-Type: application/json" \
    -d '{
        "model": "/container_model/bge-base-en-v1.5",
        "input":"I love shanghai"
    }'
```

Table 4-12 Embedding service request parameters

Parameter	Mandatory	Default Value	Type	Description
model	Yes	None	Str	The service's **served_model_name** parameter must match its own value if provided. If the service does not provide **served_model_name**, the value is the same as the **model** parameter of the service.
input	Yes	None	Str	Strings are supported.

4.1.5.4 Starting a Multimodal Model-powered Inference Service

What Is Multimodality?

Multimodality refers to the methods and techniques for integrating and processing two or more different types of information or data. Specifically, in the fields of

machine learning and AI, multimodal data typically includes, but is not limited to, text, images, video, audio, and sensor data.

The primary goal of multimodality is to leverage information from multiple modalities to enhance task performance, deliver richer user experiences, or achieve more comprehensive data analysis. For example, in real-world applications, combining image and text data can lead to improved object recognition or sentiment analysis.

Furthermore, multimodality can be subdivided into the following areas:

- Multimodal understanding: How computers extract useful information from various types of data sources and synthesize it into meaningful knowledge.
- Vision models: These models are specifically designed for images and other visual data, helping computers better understand and interpret the visual world.
- Multimodal retrieval: Techniques that use multiple data modalities (such as text, images, video, and audio) for information retrieval, aiming to provide more accurate results by integrating different forms of data.

In summary, multimodality is not merely about simple feature fusion. It encompasses a broad theoretical foundation and practical applications. In this context, multimodality refers specifically to multimodal understanding.

Constraints

For details about models and their PUs, see **Minimum Number of PUs and Maximum Sequence Length Supported by Each Model**.

Setting Common Inference Environment Variables

This topic describes how to start an LLM-powered inference service, including offline inference and online inference.

CAUTION

You must set these common framework environment variables for both offline and online inference.

Common environment variables

```
export VLLM_IMAGE_FETCH_TIMEOUT=100
export VLLM_ENGINE_ITERATION_TIMEOUT_S=600

# Prioritize setting PYTORCH_NPU_ALLOC_CONF to expandable_segments:True.
# If an error related to virtual GPU memory is reported, set it to expandable_segments:False.
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True

export VLLM_USE_V1=1
export VLLM_PLUGINS=ascend_vllm

# When the model startup needs to support ultra-large input, VLLM_ALLOW_LONG_MAX_MODEL_LE needs to be set.
# For example, qwen2.5-vl-72B and qwen2.5-vl-32B support 128K long sequences.
export VLLM_ALLOW_LONG_MAX_MODEL_LEN=1
```



```
# This parameter needs to be set for offline inference.
export VLLM_WORKER_MULTIPROC_METHOD=spawn
```

Starting Offline Inference

Use **AscendCloud-LLM/llm_inference/ascend_vllm/vllm-gpu-0.9.0/examples/offline_inference/vision_language.py script** to perform multimodal offline inference.

1. Find the entry function of the corresponding model series and modify the model weight location, for example, **qwen2.5-vl**. Change the red box in the figure below to the model weight location.

   Figure 4-2 Modifying the model weight location

   ```
   534     # Qwen2.5-VL
   535     def run_qwen2_5_vl(question: str, modality: str):
   536
   537         model_name = "Qwen/Qwen2.5-VL-3B-Instruct"
   538
   539         llm = LLM(
   540             model=model_name,
   541             max_model_len=4096,
   542             max_num_seqs=5,
   543             mm_processor_kwargs={
   544                 "min_pixels": 28 * 28,
   545                 "max_pixels": 1280 * 28 * 28,
   546                 "fps": 1,
   547             },
   ```

2. Modify model parameters in **LLM** of the entry function of the corresponding model series.

   Parameters:

   - **model**: model address in Hugging Face format.
   - **max_num_seqs**: maximum number of requests that can be processed concurrently.
   - **max_model_len**: maximum number of input and output tokens during inference.
   - **max_num_batched_tokens**: maximum number of tokens that can be used in the prefill phase. The value must be greater than or equal to the value of **--max-model-len**. The value **4096** or **8192** is recommended.
   - **dtype**: data type for model inference.
   - **tensor_parallel_size**: number of PUs you want to use.
   - **block-size**: block size of PagedAttention. The recommended value is **128**.
   - **gpu-memory-utilization**: ratio of the GPU memory used by the NPU. The input parameter name of the original vLLM is reused. The default value is **0.9**.
   - **trust_remote_code**: Indicates whether to trust remote code.
   - **distributed_executor_backend="ray"**: Uses Ray for communication.

3. Modify model parameters in **SamplingParams**.

Figure 4-3 Setting parameters in **SamplingParams**

```
def main(args):
    model = args.model_type
    if model not in model_example_map:
        raise ValueError(f"Model type {model} is not supported.")

    modality = args.modality
    mm_input = get_multi_modal_input(args)
    data = mm_input["data"]
    question = mm_input["question"]

    llm, prompt, stop_token_ids = model_example_map[model](question, modality)

    # We set temperature to 0.2 so that outputs can be different
    # even when all prompts are identical when running batch inference.
    sampling_params = SamplingParams(temperature=0.2,
                                     max_tokens=64,
                                     stop_token_ids=stop_token_ids)
```

Table 4-13 Parameters

Parameter	Mandatory	Default Value	Type	Description
max_tokens	No	16	Int	Maximum number of tokens to be generated for each output sequence.
top_k	No	-1	Int	Determines how many of the highest ranking tokens are considered. The value **-1** indicates that all tokens are considered. Decreasing the value can reduce the sampling time.
top_p	No	1.0	Float	A floating point number that controls the cumulative probability of the first several tokens to be considered. The value must be in the range (0, 1]. The value **1** indicates that all tokens are considered.
temperature	No	1.0	Float	A floating-point number that controls the randomness of sampling. Lower values make the model more deterministic, while higher values make it more random. The value **0** indicates greedy sampling.
stream	No	False	Bool	Controls whether to enable streaming inference. The default value is **False**, indicating that streaming inference is disabled.

Parameter	Mandatory	Default Value	Type	Description
ignore_eos	No	False	Bool	Indicates whether to ignore EOS and continue to generate tokens.
repetition_penalty	No	1.0	Float	Reduces the likelihood of generating repetitive text.
stop_token_ids	No	None	Int	Stops the token list. This parameter needs to be input for InternVL 2.5. For details, see **stop_token_ids** in the offline inference script **AscendCloud-LLM/llm_inference/ascend_vllm/vllm-gpu-0.9.0/examples/offline_inference/vision_language.py**.

4. Specify the image path. To specify the local image path, add the code **from PIL import Image** to the beginning of the file, and add the code **get_multi_modal_input function: image = Image.open(***{image_path}***).convert('RGB')**.

Figure 4-4 Specifying the image path

```
def get_multi_modal_input(args):
    """
    return {
        "data": image or video,
        "question": question,
    }
    """
    if args.modality == "image":
        # Input image and question
        # image = ImageAsset("cherry_blossom") \
        #     .pil_image.convert("RGB")
        image = Image.open("{image_path}").convert('RGB')
        img_question = "What is the content of this image?"

        return {
            "data": image,
            "question": img_question,
        }
```

5. Specify the input text.

Figure 4-5 Specifying the input text

```
594  def get_multi_modal_input(args):
595      """
596      return {
597          "data": image or video,
598          "question": question,
599      }
600      """
601      if args.modality == "image":
602          # Input image and question
603          image = ImageAsset("cherry_blossom") \
604              .pil_image.convert("RGB")
605          img_question = "What is the content of this image?"
606
607          return {
608              "data": image,
609              "question": img_question,
610          }
```

6. Start the inference script.
   Commands for starting the Qwen2.5-VL series models

```
python vision_language.py --model-type qwen2_5_vl
```

   Script parameters:

--model-type: model type. Currently, the options are **internvl_chat** and **qwen2_5_vl**.

--num-prompts: number of prompts entered at a time. The default value is **4**.

--modality: input type. The options are **image** and **video**. The default value is **image**.

--num-frames: number of frames extracted from the video. The default value is **16**.

Starting Online Inference

```
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 \
--dtype ${dtype} \
--host=${docker_ip} \
--port=${port} \
--gpu-memory-utilization=0.9 \
--quantization=${quantization} \
--trust-remote-code \
--compilation_config '{"cudagraph_capture_sizes": [1, 2, 4, 8, 16, 32, 64]}' \
--enforce-eager
```

The following describes the parameters in the template for starting a multimodal inference service. For details about how to set other parameters, see basic parameters in **Starting an LLM-powered Inference Service**.

- **VLLM_IMAGE_FETCH_TIMEOUT**: environment variable for the image download time.
- **VLLM_ENGINE_ITERATION_TIMEOUT_S**: maximum service interval. Exceeding this duration will result in a timeout error.
- **PYTORCH_NPU_ALLOC_CONF=expandable_segments:True**: Allows the allocator to initially create a segment and later expand its size when more memory is needed. Enabling this may improve model performance. Disable it if errors occur.
- **VLLM_USE_V1=1**: Indicates whether to enable the v1 framework of vLLM.
- **VLLM_PLUGINS=ascend_vllm**: Activates the ascend_vllm optimization plug-in of **VLLM_PLUGINS**.
- **--model ${container_model_path}**: model address in Hugging Face format, which stores the model's weight files. If quantization is used, the weights converted in **Quantization** are used. If the address for converting the trained model into the Hugging Face format is used, the original tokenizer file is required.
- **--max-num-seqs**: maximum number of concurrent requests that can be processed. Requests exceeding this limit will wait in the queue.
- **--max-model-len**: maximum number of input and output tokens during inference. The value of **max-model-len** must be less than that of **seq_length** in the **config.json** file. Otherwise, an error will be reported during inference and prediction. The **config.json** file is stored in the path corresponding to the model, for example, *${container_model_path}*/**chatglm3-6b/config.json**. The maximum length varies among different models. For details, see **Table 4-4**.

- **--max-num-batched-tokens**: maximum number of tokens that can be used in the prefill phase. The value must be greater than or equal to the value of **--max-model-len**. The value **4096** or **8192** is recommended.
- **--dtype**: data type for model inference, which can be FP16 or BF16. **float16** indicates FP16, and **bfloat16** indicates BF16. If unspecified, the data type is auto-matched based on input data. Using different **dtype** may affect model precision. When using open-source weights, you are advised not to specify **dtype** and instead use the default **dtype** of the open-source weights.
- **--tensor-parallel-size**: number of PUs you want to use. The product of model parallelism and pipeline parallelism must be the same as the number of started NPUs. For details, see **Table 4-4**. The value **1** indicates that the service is started using a single PU.
- **--block-size**: block size of kv-cache. The recommended value is **128**.
- **--host=${docker_ip}**: IP address for service deployment. Replace *${docker_ip}* with the actual IP address of the host machine. The default value is **None**. For example, the parameter can be set to **0.0.0.0**.
- **--port**: port where the service is deployed.
- **--gpu-memory-utilization**: ratio of the GPU memory used by the NPU. The input parameter name of the original vLLM is reused. The default value is **0.9**.
- **--trust-remote-code**: Specifies whether to trust the remote code.
- **--chat-template**: (optional) chat building template.
- **--quantization**: quantization option. This parameter is optional. Currently, only **awq** is supported. If this parameter is not transferred, the default value **None** is used, indicating that quantization is disabled.
- **--compilation_config**: compilation options of the model. This parameter is optional. For example, **{"cudagraph_capture_sizes": [1, 2, 4, 8, 16, 32, 64]}** is used to enable graph capture in different batch sizes.
- **--enforce-eager**: Boots in eager mode instead of acl mode. This parameter is optional.

Multimodal Inference Request

Send a request using **online_serving.py** (single-image, single-turn dialog).

Since multimodal inference involves image encoding and decoding, service APIs are invoked via scripting. The parameters that need to be configured in the script are described in **Table 4-14**.

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

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



```
def post_img(args):
    # Path to your image
    image_path = args.image_path
    # Getting the base64 string
    image_base64 = encode_image(image_path)
    stop_token_ids = args.stop_token_ids if args.stop_token_ids is not None else
get_stop_token_ids(args.model_path)
    headers = {
      "Content-Type": "application/json"
    }
    payload = {
      "model": args.model_path,
      "messages": [
        {
          "role": "user",
          "content": [
            {
              "type": "text",
              "text": args.text
            },
            {
              "type": "image_url",
              "image_url": {
                "url": f"{image_base64}"
              }
            }
          ]
        }
      ],
      "max_tokens": args.max_tokens,
      "temperature": args.temperature,
      "ignore_eos": args.ignore_eos,
      "stream": args.stream,
      "top_k": args.top_k,
      "top_p": args.top_p,
      "stop_token_ids": stop_token_ids,
      "repetition_penalty": args.repetition_penalty,
    }
    response = requests.post(f"http://{args.docker_ip}:{args.served_port}/v1/chat/completions",
headers=headers, json=payload)
    print(response.json())

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # Mandatory
    parser.add_argument("--model-path", type=str, required=True)
    parser.add_argument("--image-path", type=str, required=True)
    parser.add_argument("--docker-ip", type=str, required=True)
    parser.add_argument("--served-port", type=str, required=True)
    parser.add_argument("--text", type=str, required=True)
    # Optional
    parser.add_argument("--temperature", type=float, default=0) # Randomness of the output result. This
parameter is optional.
    parser.add_argument("--ignore-eos", type=bool, default=False) # Specifies whether to ignore the end
symbol during generation and continue to generate tokens after an EOS token is generated. This parameter
is optional.
    parser.add_argument("--top-k", type=int, default=-1) # Controls result diversity. A lower value makes
text more unique but less coherent. A higher value improves coherence but reduces diversity. This
parameter is optional.
    parser.add_argument("--top-p", type=int, default=1.0) # The value ranges between 0 and 1. A smaller
value results in more unexpected outputs, but may sacrifice coherence. A larger value leads to more
coherent content, but reduces novelty. This parameter is optional.
    parser.add_argument("--stream", type=int, default=False) # Controls whether to enable streaming
inference. The default value is False, indicating that streaming inference is disabled.
    parser.add_argument("--max-tokens", type=int, default=16) # Maximum length of the generated
sequence. This parameter is mandatory.
    parser.add_argument("--repetition-penalty", type=float, default=1.0) # Reduces the likelihood of
generating repetitive text. This parameter is optional.
```



```
    parser.add_argument("--stop-token-ids", nargs='+', type=int, default=None) # Stops the token list. This parameter is optional.
    args = parser.parse_args()
    post_img(args)
```

Run this script:

```
python online_serving.py --model-path ${container_model_path} --image-path ${image_path} --docker-ip ${docker_ip} --served-port ${port} --text What is the image content?
```

For details about request parameters, see **Multimodal Request Parameters**.

Figure 4-6 Input example

Multimodal Request Parameters

Table 4-14 Script parameters

Parameter	Mandatory	Type	Description
container_model_path	Yes	str	Model weight file path
image_path	Yes	str	Path of the image passed to the model for inference
docker_ip	Yes	str	IP address of the host where the multimodal OpenAI service is started
served_port	Yes	str	Port number for starting the multimodal OpenAI service
text	Yes	str	Prompts passed to the model for inference

Table 4-15 JSON parameters in the payload of a POST request

Parameter	Mandatory	Default Value	Type	Description
model	Yes	None	Str	This parameter is mandatory for an inference request when the service is started through the OpenAI service API. The value must be the same as the value of **model ${container_model_path}** when the inference service is started.
messages	Yes	-	Dict	Input question and image of the request. **role**: indicates the message sender, which can only be a user here. **content**: indicates the message content, with the type being a list. For a single-image single-turn dialogue, the content must include two elements. The first element has its **type** field set to **text**, representing the text type, and the **text** field contains the string of the input question. The second element has its **type** field set to **image_url**, representing the image type, and the **image_url** field contains the Base64 encoding of the input image.
max_tokens	No	16	Int	Maximum number of tokens to be generated for each output sequence.
top_k	No	-1	Int	Determines how many of the highest ranking tokens are considered. The value **-1** indicates that all tokens are considered. Decreasing the value can reduce the sampling time.
top_p	No	1.0	Float	A floating point number that controls the cumulative probability of the first several tokens to be considered. The value must be in the range (0, 1]. The value **1** indicates that all tokens are considered.
temperature	No	1.0	Float	A floating-point number that controls the randomness of sampling. Lower values make the model more deterministic, while higher values make it more random. The value **0** indicates greedy sampling.
stream	No	False	Bool	Controls whether to enable streaming inference. The default value is **False**, indicating that streaming inference is disabled.
ignore_eos	No	False	Bool	Indicates whether to ignore EOS and continue to generate tokens.

Parameter	Mandatory	Default Value	Type	Description
repetition_penalty	No	1.0	Float	Reduces the likelihood of generating repetitive text.
stop_token_ids	No	None	Int	Stops the token list. This parameter needs to be input for InternVL2 and MiniCPM. For details, see **stop_token_ids** in the offline inference script **examples/offline_inference_vision_language.py**.

4.1.6 Usage of Key Inference Features

4.1.6.1 Quantization

4.1.6.1.1 W4A16 Quantification

In foundation model inference, the data types for model weights (weight), inference computations (activation), and kvcache are generally half-precision floating-point (FP16 or BF16). Quantization refers to the process of converting high-bit floating-point numbers to lower-bit data types, such as int4 or int8.

Model quantization includes weight-only quantization, weight-activation quantization, and kvcache quantization.

The general steps for quantization are: 1. Mirror and quantize the floating-point weights and save the quantized weights. 2. Use the quantized weights for inference deployment.

What Is W4A16 Quantization?

W4A16 quantization is a large model compression optimization technique. In this technique, W4 indicates that the model's weights are quantized to 4-bit integers (int4), and A16 indicates that the activations (or inputs/outputs) are kept as 16-bit floating-point numbers (FP16 or BF16).

This quantization method only quantizes the parameters to 4 bits, while the activation values maintain FP16 precision. The advantages of this approach include significantly reducing the model's memory usage and the number of PUs required for deployment (by approximately 75%). It also substantially reduces the incremental inference latency for small batches.

Constraints

- The llm-compressor tool supports W4A16 and per-group (group-size=128) quantization, but does not support setting the **actorder** parameter to **group**.

- Currently, W4A16 quantization is only supported for Qwen series models. For the list of supported model, see **Table 4-2**.
- Qwen series models quantized with W4A16 only support being launched in graph mode and do not support setting the Qwen series performance optimization environment variables. The specific configuration will be described in detail in the following sections.

Obtaining Quantized Model Weights

You can obtain quantized model weights in either of the following ways:

Method 1: **Download the llm-compressor quantized model from the Hugging Face community**.

Method 2: After obtaining the FP16/BF16 model weights, **use the llm-compressor tool for quantization**.

Using the llm-compressor Tool to Quantize Models

This section describes how to use the open-source quantization tool llm-compressor to quantize model weights on an NPU server and then perform quantized inference on the NPU server. For more information about the open-source quantization tool, see **llm-compressor**.

1. To use this quantization tool, switch to a specific conda environment:
   ```
   conda create --name llmcompressor --clone PyTorch-2.5.1
   conda activate llmcompressor
   ```
2. Install llm-compressor.
   ```
   pip install llmcompressor==0.6.0
   pip install transformers==4.51.3
   pip install zstandard
   ```
3. Create the quantization script. For the complete **llm_compressor_W4A16.py** script, see **llm_compressor_W4A16.py Script**.
4. Configure the calibration dataset. By default, the W4A16 quantization script uses the **mit-han-lab/pile-val-backup** dataset for calibration. If you need to specify a different dataset, modify the following fields in the **llm_compressor_W4A16.py** script:
   ```
   DATASET_ID = "mit-han-lab/pile-val-backup"
   DATASET_SPLIT = "validation"

   NUM_CALIBRATION_SAMPLES = 256
   MAX_SEQUENCE_LENGTH = 512
   ```
   If your current environment cannot access the Hugging Face website to download the dataset, you can download it manually using a web browser and upload it to the server. Then, set **DATASET_ID** to the server path. The dataset download link is: **https://huggingface.co/datasets/mit-han-lab/pile-val-backup/tree/main**.

   After downloading, upload the **val.jsonl.zst** file to a custom directory on the server, for example **<local-dir>**, and decompress it:
   ```
   cd <local-dir>
   zstd -d val.jsonl.zst
   ```
   The decompressed file will be named **val.jsonl**. After decompression, delete the original file to avoid affecting data reading during quantization:
   ```
   rm -f val.jsonl.zst
   ```


Set **DATASET_ID** to the directory containing the **val.jsonl** file:

```
DATASET_ID = "<local-dir>"
```

5. Configure the quantization algorithm. By default, the W4A16 quantization script provides the Asymmetric Weight Quantization (AWQ) algorithm:

```
# Configure the quantization algorithm to run.
recipe = [
    AWQModifier(ignore=["lm_head"], scheme="W4A16_ASYM", targets=["Linear"]),
]
```

If you want to use symmetric quantization, you can configure **scheme="W4A16"**.

6. Run the **llm_compressor_W4A16.py** script to quantize the model. The model weight path needs to be modified according to your actual situation. The quantization time depends on the model size and is expected to take 30 minutes to 3 hours.

```
# Specify the PU number for quantization based on the available PUs on the server. If not set, it defaults to PU 0.
export ASCEND_RT_VISIBLE_DEVICES=0
python llm_compressor_W4A16.py --model-path /home/ma-user/Qwen3-32B/ --quant-path /home/ma-user/Qwen3-32B-quant/
```

Parameters:

- **--model-path**: Path to the original model weights.
- **--quant-path**: Path to save the quantized weights.

7. When starting the service, add the following quantization parameters. For details, see **Starting a Real-Time Inference Service**.

```
-q compressed-tensors or --quantization compressed-tensors
```

W4A16 models do not support eager or acl-graph modes. The service must be started in ascend_turbo graph mode. The configuration is as follows:

```
--additional-config: {"ascend_turbo_graph_config": {"enabled": true}}
```

W4A16 models do not support configuring Qwen series performance optimization environment variables. The following environment variables should not be set when starting the service:

```
unset ENABLE_QWEN_HYPERDRIVE_OPT
unset ENABLE_QWEN_MICROBATCH
unset ENABLE_PHASE_AWARE_QKVO_QUANT
unset DISABLE_QWEN_DP_PROJ
```

For the Qwen2.5-72B-instruct and Qwen2-72B-instruct models, before starting the service after quantization, you need to modify the **config.json** file of the quantized W4A16 model. Change the **intermediate_size** value from **29568** to **29696**. The reason is that the FFN blocks of the Qwen2.5-72B-instruct model are **29568**, and the W4A16 quantization group size is **128**. 29568/128 = 231, which cannot be evenly divided by TP greater than 1, so the service cannot run normally in TP > 1 scenarios. Changing **intermediate_size** to **29696** (29696/128 = 232, which can be divided by TP=4 or TP=8) allows for multi-PU inference (refer to **community documentation**). The service will automatically pad the weights to the specified shape after the configuration is modified. The modification is as follows:

```
vim config.json
# Modify the following configuration item:
"intermediate_size": 29696,
```

llm_compressor_W4A16.py Script

The complete code of the **llm_compressor_W4A16.py** script is as follows:


```
import argparse
import os
from functools import partial
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

from datasets import load_dataset
from llmcompressor import oneshot
from llmcompressor.modifiers.awq import AWQModifier
from transformers import AutoModelForCausalLM, AutoTokenizer


os.environ['PYTORCH_NPU_ALLOC_CONF'] = 'expandable_segments:False'

# Select calibration dataset.
DATASET_ID = "mit-han-lab/pile-val-backup"
DATASET_SPLIT = "validation"

# Select number of samples. 256 samples is a good place to start.
# Increasing the number of samples can improve accuracy.
NUM_CALIBRATION_SAMPLES = 256
MAX_SEQUENCE_LENGTH = 512


def preprocess(example, tokenizer):
    return {
        "text": tokenizer.apply_chat_template(
            [{"role": "user", "content": example["text"]}],
            tokenize=False,
        )
    }


def quantize(model_id, quantized_path):
    # Select model and load it
    model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype="auto")
    tokenizer = AutoTokenizer.from_pretrained(model_id, trust_remote_code=True)

    ds = load_dataset(DATASET_ID, split=f"{DATASET_SPLIT}[:{NUM_CALIBRATION_SAMPLES}]")
    ds = ds.shuffle(seed=42)
    preprocess_func = partial(preprocess, tokenizer=tokenizer)
    ds = ds.map(preprocess_func)

    # Configure the quantization algorithm to run.
    recipe = [
        AWQModifier(ignore=["lm_head"], scheme="W4A16_ASYM", targets=["Linear"]),
    ]

    oneshot(
        model=model,
        dataset=ds,
        recipe=recipe,
        max_seq_length=MAX_SEQUENCE_LENGTH,
        num_calibration_samples=NUM_CALIBRATION_SAMPLES,
    )

    # Save to disk compressed.
    model.save_pretrained(quantized_path, save_compressed=True)
    tokenizer.save_pretrained(quantized_path)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--model-path", type=str, required=True, help="Path to the input model")
    parser.add_argument("--quant-path", type=str, required=True, help="Path to save the compressed
model")
    args = parser.parse_args()
    quantize(args.model_path, args.quant_path)
```


4.1.6.1.2 W8A8 Quantization

What Is W8A8 Quantization?

W8A8 quantization is a technique that quantizes both model weights and activations to 8-bit data. This technique converts high-bit floating-point numbers to 8-bit integers, typically converting weights and activations from 16-bit or 32-bit floating-point numbers to 8-bit integer (int8) format. After quantization, the model weight size is reduced, and using int8 format data for matrix multiplication (MatMul) operations can reduce the computational load, thereby improving inference performance.

W8A8 quantization can reduce the model's memory usage and the number of PUs required for deployment. It also reduces both the initial token latency and the incremental inference latency.

Constraints

- **Supported Models** lists the models supported for W8A8 quantization.
- Activation quantization supports dynamic per-token quantization and symmetric quantization.
- Weight Quantization supports per-channel quantization and symmetric quantization.

Obtaining Quantized Model Weights

You can obtain quantized model weights in either of the following ways:

Method 1: **Download the llm-compressor quantized model from the Hugging Face community**.

Method 2: After obtaining the FP16/BF16 model weights, **use the llm-compressor tool for quantization**.

Using the llm-compressor Tool to Quantize Models

This section describes how to use the open-source quantization tool llm-compressor to quantize model weights on an NPU server and then perform quantized inference on the NPU server. For more information about the open-source quantization tool, see **llm-compressor**.

1. To use this quantization tool, switch to a specific conda environment:
   ```
   conda create --name llmcompressor --clone PyTorch-2.5.1
   conda activate llmcompressor
   ```
2. Install llm-compressor.
   ```
   pip install llmcompressor==0.6.0
   pip install transformers==4.51.3
   pip install zstandard
   ```
3. By default, W8A8 quantization uses the HuggingFaceH4/ultrachat_200k dataset as the calibration dataset. To specify a dataset, modify the following fields in the **llm_compressor_W8A8.py** script:
   ```
   DATASET_ID = "HuggingFaceH4/ultrachat_200k"
   DATASET_SPLIT = "train_sft"
   ```



```
NUM_CALIBRATION_SAMPLES = 512
MAX_SEQUENCE_LENGTH = 2048
```

If your current environment cannot access the Hugging Face website to download the dataset, you can download it manually using a web browser and upload it to the server. Then, set **DATASET_ID** to the server path. The dataset download link is: **https://huggingface.co/datasets/HuggingFaceH4/ultrachat_200k/tree/main**.

The original **data** directory must be retained for the downloaded dataset. The following is an example of the directory structure after the dataset is downloaded, where **<local-data-dir>** is the custom directory for storing the dataset.

```
<local-data-dir>
  -- ultrachat_200k
     -- data
        -- train_sft-xxx.parquet
        -- train_gen-xxx.parquet
```

Set the local dataset directory above the data folder but do not include the data folder itself:

```
DATASET_ID="<local-data-dir>/ultrachat_200k"
```

For details about the complete **llm_compressor_W8A8.py** script, see **llm_compressor_W8A8.py Script**.

4. Run the **llm_compressor_W8A8.py** file to quantize the model. The quantization time depends on the model size. It takes about 30 minutes to 3 hours.

```
# Specify the PU number for quantization based on the available PUs on the server. If not set, it defaults to PU 0.
export ASCEND_RT_VISIBLE_DEVICES=0
python llm_compressor_W8A8.py --model-path /home/ma-user/Qwen2.5-72B/ --quant-path /home/ma-user/Qwen2.5-72B-quant/
```

Parameters:

- **--model-path**: Path to the original model weights.
- **--quant-path**: Path to save the quantized weights.

If the following warning logs are recorded during quantization, ignore them. The quantization function is not affected.

```
get_GPU_usage_nv | WARNING - Pynml library error:
 NVML Shared Library Not Found
```

5. Add the following command when starting the service. For details, see **Starting a Real-Time Inference Service**.

```
 -q compressed-tensors or --quantization compressed-tensors
```

llm_compressor_W8A8.py Script

The complete code of the **llm_compressor_W8A8.py** script is as follows:

```
import argparse
import os
from functools import partial
import torch
import torch_npu
from torch_npu.contrib import transfer_to_npu

from datasets import load_dataset
from llmcompressor import oneshot
from llmcompressor.modifiers.awq import AWQModifier
from llmcompressor.modifiers.quantization import GPTQModifier
from llmcompressor.modifiers.smoothquant import SmoothQuantModifier
```



```
from llmcompressor.utils import dispatch_for_generation
from transformers import AutoModelForCausalLM, AutoTokenizer


os.environ['PYTORCH_NPU_ALLOC_CONF'] = 'expandable_segments:False'

# Select calibration dataset.
DATASET_ID = "HuggingFaceH4/ultrachat_200k"
DATASET_SPLIT = "train_sft"

# Select number of samples. 512 samples is a good place to start.
# Increasing the number of samples can improve accuracy.
NUM_CALIBRATION_SAMPLES = 512
MAX_SEQUENCE_LENGTH = 2048


def preprocess(example, tokenizer):
    return {
        "text": tokenizer.apply_chat_template(
            example["messages"],
            tokenize=False,
        )
    }


# Tokenize inputs.
def tokenize(sample, tokenizer):
    return tokenizer(
        sample["text"],
        padding=False,
        max_length=MAX_SEQUENCE_LENGTH,
        truncation=True,
        add_special_tokens=False,
    )


def quantize(model_id, quantized_path):
    # Select model and load it
    model = AutoModelForCausalLM.from_pretrained(model_id, torch_dtype="auto")
    tokenizer = AutoTokenizer.from_pretrained(model_id)

    # Load dataset and preprocess.
    ds = load_dataset(DATASET_ID, split=f"{DATASET_SPLIT}[:{NUM_CALIBRATION_SAMPLES}]")
    ds = ds.shuffle(seed=42)
    preprocess_func = partial(preprocess, tokenizer=tokenizer)
    ds = ds.map(preprocess_func)
    tokenize_func = partial(tokenize, tokenizer=tokenizer)
    ds = ds.map(tokenize_func, remove_columns=ds.column_names)

    # Configure algorithms. In this case, we:
    #   * apply SmoothQuant to make the activations easier to quantize
    #   * quantize the weights to int8 with GPTQ (static per channel)
    #   * quantize the activations to int8 (dynamic per token)
    recipe = [
        SmoothQuantModifier(smoothing_strength=0.8),
        GPTQModifier(targets="Linear", scheme="W8A8", ignore=["lm_head"]),
    ]

    oneshot(
        model=model,
        dataset=ds,
        recipe=recipe,
        max_seq_length=MAX_SEQUENCE_LENGTH,
        num_calibration_samples=NUM_CALIBRATION_SAMPLES,
    )

    # Save to disk compressed.
    model.save_pretrained(quantized_path, save_compressed=True)
    tokenizer.save_pretrained(quantized_path)
```



```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--model-path", type=str, required=True, help="Path to the input model")
    parser.add_argument("--quant-path", type=str, required=True, help="Path to save the compressed
model")
    args = parser.parse_args()
    quantize(args.model_path, args.quant_path)
```

4.1.6.2 Prefix Caching

What Is Prefix Caching?

In LLM inference applications, scenarios with long system prompts and multi-turn dialogues are often encountered. In the scenario with long system prompts, the system prompt remains the same across different requests, and the KV Cache computation is also the same. In the multi-turn dialogue scenario, each round of dialogue depends on the context of all previous rounds, and the KV Cache from the historical rounds needs to be recalculated in each subsequent round. In both cases, if the system prompt and the KV Cache from the historical rounds can be saved and reused for subsequent requests, it would significantly reduce the time to first token (TTFT). If both the Prefix Cache and the Generated KV Cache can be cached, in multi-turn dialogue applications, ignoring edge cases, it can be considered that the recompute of the historical rounds' generated dialogue is essentially eliminated.

Ascend-vLLM provides the key feature of prefix caching, which can significantly reduce TTFT in scenarios with long system prompts and multi-turn dialogues, enhancing user experience. Its advantages mainly include:

- **Shorter prefill time**: Since the KV cache corresponding to repeated token sequences across requests can be reused, this can reduce the time spent on computing the KV cache for some prefix tokens, thereby reducing the prefill time.
- **More efficient memory usage**: When the requests being processed have common prefixes, the KV cache of the common prefix part can be shared, avoiding the need to occupy multiple portions of memory repeatedly.

Constraints

- This feature cannot be used together with **ascend_scheduler_config**.
- Chunked prefill is enabled by default and can be disabled only after **ascend_scheduler_config** takes effect. Therefore, when prefix caching takes effect, chunked prefill also takes effect.
- Multimodal models do not support prefix caching.
- The Qwen2.5 and Qwen3 models support this feature.
- The KV cache of the public prefix token is reused only when the number of cross-request public prefix tokens is greater than or equal to the block size in PagedAttention.

Prefix caching parameter settings

Table 4-16 describes the supplementary parameters to be set for using prefix caching when the inference service is started. Table 4-17 shows the code example.

Table 4-16 Prefix caching parameters

Service Startup Method	Configuration Item	Type	Range	Description
offline	enable_prefix_caching	bool	• True • False	• **True**: Enables prefix caching. If this parameter is not configured, it will default to **True**. • **False**: Does not enable prefix caching.
online	--no-enable-prefix-caching	-	-	• The current version defaults to enabling prefix caching. • This configuration item is used to disable prefix caching. Note: Enabling prefix caching is specified when starting the service and is an action type parameter.

Table 4-17 Code examples for enabling prefix caching

Service Startup Method	API	Service Startup Base Command
offline	-	`LLM(model="facebook/opt-125m", enable_prefix_caching=True)`

Service Startup Method	API	Service Startup Base Command
online (No configuration needed; it is enabled by default)	openai	python -m vllm.entrypoints.openai.api_server \ --model=facebook/opt-125m

Inference Execution Reference

1. Configure the service parameters. To use this feature in Ascend-vLLM, see **Table 4-16** and **Table 4-17**. For details about other parameters, see **Starting an LLM-powered Inference Service**.
2. Start the service. For details, see **Starting an LLM-powered Inference Service**.
3. Evaluate the accuracy and performance. For details, see **Inference Service Accuracy Evaluation** and **Inference Service Performance Evaluation**.

4.1.6.3 Speculative Inference

4.1.6.3.1 Usage of Speculative Inference

What Is Speculative Inference?

Traditional LLM inference primarily relies on an auto-regressive decoding method, where each decoding step can only produce one output token, and the historical output content must be concatenated and re-input into the LLM to proceed with the next decoding step. To address this issue, a speculative inference approach has been proposed. The core idea is to use a small model, which has a much lower computational cost than the LLM, to perform speculative inference. Specifically, the small model speculatively infers multiple steps at a time, and then these inference results are collected and validated by the LLM in a single step.

In speculative mode, the small model first infers tokens 1, 2, and 3 sequentially, and then these 3 tokens are input into the LLM for inference, producing tokens 1', 2', 3', and 4'. The tokens 1', 2', and 3' are then compared with 1', 2', and 3' respectively. This process allows for the generation of 1 to 4 tokens using three speculative inferences from the small model (which are much faster compared to

the large model) and one inference from the large model, significantly improving inference performance.

In this way, speculative inference can bring the following advantages:

Shorter average decode time: Taking qwen2-72b as the large model and qwen2-0.5b as the small model, the inference time for the small model is less than 1/5 of the large model. After including the validation step, the time to execute a complete speculative inference cycle is only about 1.5 times that of the large model (with the speculative step set to 3). This single speculative inference cycle can, on average, generate 3 valid tokens. Thus, with a time cost of 1.5 times, it generates 3 times the number of tokens, resulting in a 100% performance improvement.

Speculative Inference Parameter Settings

When starting the offline or online inference service, set parameters by referring to **Table 4-18** to use speculative inference.

Table 4-18 Speculative inference parameters

Configuration Item	Parameter	Type	Description
--speculative-config	num_speculative_tokens	int	The number of tokens to predict each time, must be greater than or equal to 1. It is recommended to initially set this to **1**, and then adjust based on the acceptance rate after confirming the benefits.
	method	str	Speculative method: currently only **ngram** is supported.
	prompt_lookup_min	int	Minimum match length, effective only when the method is set to **ngram**.
	prompt_lookup_max	int	Maximum match length, effective only when the method is set to **ngram**.

Example of E2E Speculative Inference

The following uses the Qwen3-32B model as the large model and the OpenAI API service as an example.

1. Start the inference service:

```
base_model=/path/to/base_model
export VLLM_PLUGINS=ascend_vllm
export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
python -m vllm.entrypoints.openai.api_server --model=${base_model} \
--max-num-seqs=256 \
--max-model-len=8192 \
--max-num-batched-tokens=8192 \
--dtype=bfloat16 \
--tensor-parallel-size=4 \
--host=0.0.0.0 \
```



```
--port=18080 \
--gpu-memory-utilization=0.8 \
--trust-remote-code \
--additional-config='{"ascend_turbo_graph_config": {"enabled": true}}' \
--speculative-config
'{"num_speculative_tokens":1,"method":"ngram","prompt_lookup_min":1,"prompt_lookup_max":8}'
```

2. Send a curl request.

```
curl --request POST \
  --url http://0.0.0.0:18080/v1/chat/completions \
  --header 'content-type: application/json' \
  --data '{
  "model": "${base_model}",
  "messages": [
    {
      "role": "user",
      "content": "Who is the current president of the United States?"
    }
  ],
  "max_tokens": 128,
  "top_k": -1,
  "top_p": 0.1,
  "temperature": 0,
  "stream": false,
  "repetition_penalty": 1.0
}
'
```

Inference Execution Reference

1. Configure the service parameters. To use this feature in Ascend-vLLM, see **Table 4-18**. For details about other parameters, see **Starting an LLM-powered Inference Service**.
2. Start the service. For details, see **Starting an LLM-powered Inference Service**.
3. Evaluate the accuracy and performance. For details, see **Inference Service Accuracy Evaluation** and **Inference Service Performance Evaluation**.

4.1.6.3.2 N-Gram Speculation

What Is N-Gram Speculation?

N-Gram speculation is a technique used to optimize inference performance, primarily for accelerating the generation process of large language models (LLMs). It predicts subsequent tokens using N-Gram matching, reducing the computational load of the model and thus improving generation speed.

Scenarios

Applicable Scenarios

- Long text generation, such as stories and code completion.
- High repetitiveness tasks, such as batch question answering and translation.

Inapplicable Scenario

- Short text generation, for which the benefits of speculation are not significant.
- High randomness tasks, such as creative writing, where the N-Gram matching rate is low.

How It Works

1. Cached historical n-grams: Record sequences of already generated tokens (for example, ["the", "quick", "brown"]).
2. Matching-based prediction: When a new input partially matches an n-gram prefix (for example, ["the", "quick"]), the subsequent token (for example, "brown") is directly predicted.
3. Speculative execution: If the predicted token is accepted, part of the computation is skipped; otherwise, it falls back to standard decoding.

Enabling N-Gram Speculation Parameters

Configuration Item	Parameter	Type	Description
--speculative-config	num_speculative_tokens	int	The number of tokens to predict each time.
	method	str	Speculative method: ngram.
	prompt_lookup_min	int	Minimum match length.
	prompt_lookup_max	int	Maximum match length.

4.1.6.4 Graph Mode

What is ASCEND-TURBO-GRAPH?

AscendTurboGraph is a Host graph based on a Capture-Replay architecture. It effectively eliminates Host bottlenecks, supports dynamic shapes for model inputs, and does not require bucketing for graph construction, making the graph construction process faster. In the default mode (when the **INFER_MODE** environment variable is not set), some models will automatically use the ACLGraph mode to enhance performance.

ASCEND-TURBO-GRAPH Constraints

Currently, AscendTurboGraph only supports large language models (LLMs) of the Qwen2, Qwen2.5, and Qwen3 series architectures, including their quantized models. Due to the lack of adaptation for some operators, other scenarios are not yet supported.

ASCEND-TURBO-GRAPH Parameter Configuration

By default, the AscendTurboGraph mode is used, and the **VLLM_PLUGINS** environment variable is set as follows:

```
export VLLM_PLUGINS=ascend_vllm,kv_connectors
```

If you want to use the eager or acl-graph mode, the **VLLM_PLUGINS** environment variable should be set as follows:


```
export VLLM_PLUGINS=ascend
```

Table 4-19 Execution mode settings

Execution Mode	Configuration Item	Description
eager	--enforce-eager	The default value is **False** and has the highest priority.
AscendTurboGraph (Recommended)	--additional-config='{"ascend_turbo_graph_config": {"enabled": true}}'	Needs to be explicitly set in **additional_config**.
ACLGraph	N/A	If not set, the default mode is ACLGraph. ACLGraph mode is an experimental feature in the current version and is not recommended for use. **It is recommended to use the AscendTurboGraph mode**.

4.1.6.5 Chunked Prefill

What Is Chunked Prefill?

Chunked Prefill (Splitfuse) is a feature designed to break down long prompt requests into smaller chunks and schedule them across multiple forward steps. Generation for the prompt request only begins after the forward pass of the last chunk is completed. Short prompt requests are combined to precisely fill the gaps in the steps, ensuring that the computational load of each step is roughly equal, which helps to stabilize the average latency across all requests.

Key operations:

1. Long prompts are divided into smaller chunks and scheduled over multiple iterations. Output token generation only occurs after the last iteration.
2. During batch construction, a prefill chunk is combined with the remaining slots filled by decode operations, reducing the cost of batching only decode operations.

Its advantages mainly include:

- **Improved efficiency**: By efficiently combining short and long prompts, the model maintains high throughput.
- **Enhanced consistency**: Standardizing the forward pass size reduces latency fluctuations and stabilizes the generation frequency.
- **Reduced latency**: By balancing the computational utilization of prefill and decode, it reduces the **P90_ttft** (time to first token) and **P90_tpot** (time per

output token) latencies, particularly in scenarios with short inputs, short outputs, and high concurrency.

Constraints

- This feature cannot be used simultaneously with automatic prefix caching (APC) and KV cache quantization.
- The Qwen series of models support this feature.
- It is enabled by default in v1. When enabling the vllm_ascend capability, additional configuration is required to ensure it remains enabled; otherwise, it will be disabled by default. See the table below for configuration details (Note: In cases 1 and 3, the chunked_prefill feature will be enabled; in case 2, it will be disabled).

Chunked Prefill Parameters

The parameters for chunked prefill are listed in the table below.

Table 4-20 Parameters

Configuration Item	Type	Range	Description

enable-chunked-prefill	bool	• true
		• false
			1. The chunked_prefill feature is enabled by default in the open-source vllm v1 scheduler on GPUs or NPUs.
			2. When using vllm_ascend, you need to set **ascend_scheduler_config** to **enabled = true**. In this case, chunked_prefill will be disabled by default.
			`--additional-config="ascend_scheduler_config": { "enabled": true }`
			3. When using vllm_ascend and you want to enable chunked_prefill, you must add additional configuration under **ascend_scheduler_config**. Otherwise, it will remain disabled (as in case 2).
			To enable chunked_prefill when using vllm_ascend, configure these settings:
			`--additional-config="ascend_scheduler_config": { "enabled": true, "chunked_prefill_enabled": true, # If **chunked_prefill_enabled** is not set to **true**, the chunked_prefill feature will remain disabled by default. }`

max-num-batched-tokens	int	≥ 256 and a multiple of 256	In chunked prefill mode, this parameter limits the maximum split length and must be greater than or equal to **--max-model-len**. Recommended values are **4096**, **8192**, or even larger.

Inference Execution Reference

1. To use chunked prefill in Ascend-vLLM, see **Table 4-20**. For details about other parameters, see **Starting an LLM-powered Inference Service**.
2. For details about how to start the inference service, see **Starting an LLM-powered Inference Service**.

4.1.6.6 Structured Outputs

What are Structured Outputs?

Structured outputs refer to the generation of text that strictly adheres to a specified format provided by the user in foundation model requests. These formats can include JSON, SQL, or other structured data formats. The guided decoding feature supports the generation of text that conforms to the user-specified format.

What is Guided Decoding?

Guided decoding is a strategy used to generate text by providing additional context or constraints to guide the model towards producing results that are more in line with the desired outcome.

For example, when starting a service with OpenAI, you can configure the **guided_json** parameter to use a JSON Schema to guide the generation process.

JSON Schema uses special keywords to describe data structures, such as title, type, properties, required, and definitions. It guides the model to generate a JSON object containing user information by specifying object attributes, types, and formats.

Its advantages mainly include:

- **Context guidance**: By providing specific format templates, the model can better understand the attributes of each field and filter out irrelevant information.
- **Constrained generation**: You can set certain constraints, such as keywords, topics, or styles, to ensure that the generated content is more consistent and relevant.
- **Improved quality**: The generated text typically adheres strictly to the user-specified format requirements, ensuring logical and rigorous content.

Principles

Guided decoding works by converting the user-provided content format templates (such as JSON Schema, CFG language, and regular expressions) into a Finite State Machine (FSM). The FSM is used to guide and filter the output tokens. The state transitions within the FSM define the content of each field in the formatted output. By using probabilistic interventions, tokens that do not meet the specified conditions are excluded, thus normalizing the generated text.

Using Guided Decoding for Offline Inference

To use guided-decoding for offline inference, configure **GuidedDecodingParams** in the **SamplingParams** class.

The following is an example of using the offline mode:

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

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

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

MODEL_NAME indicates the model path.

Startup Parameters

Table 4-21 Parameters

Parameter	Type	Range	Description
--guided-decoding-backend	str	{xgrammar}	Enables the guided generation feature. Currently, the guided generation feature only supports the xgrammar backend.

Using Guided Decoding for Online Inference

For details about how to start the inference service, see **Starting an LLM-powered Inference Service**. A **startup parameter** needs to be added.

When guided decoding is used for online inference, the request contains the guided_json architecture. For details, see the following code:

```
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\", \"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
```



```
\"}}}"
}'
```

4.1.6.7 Tool Calling

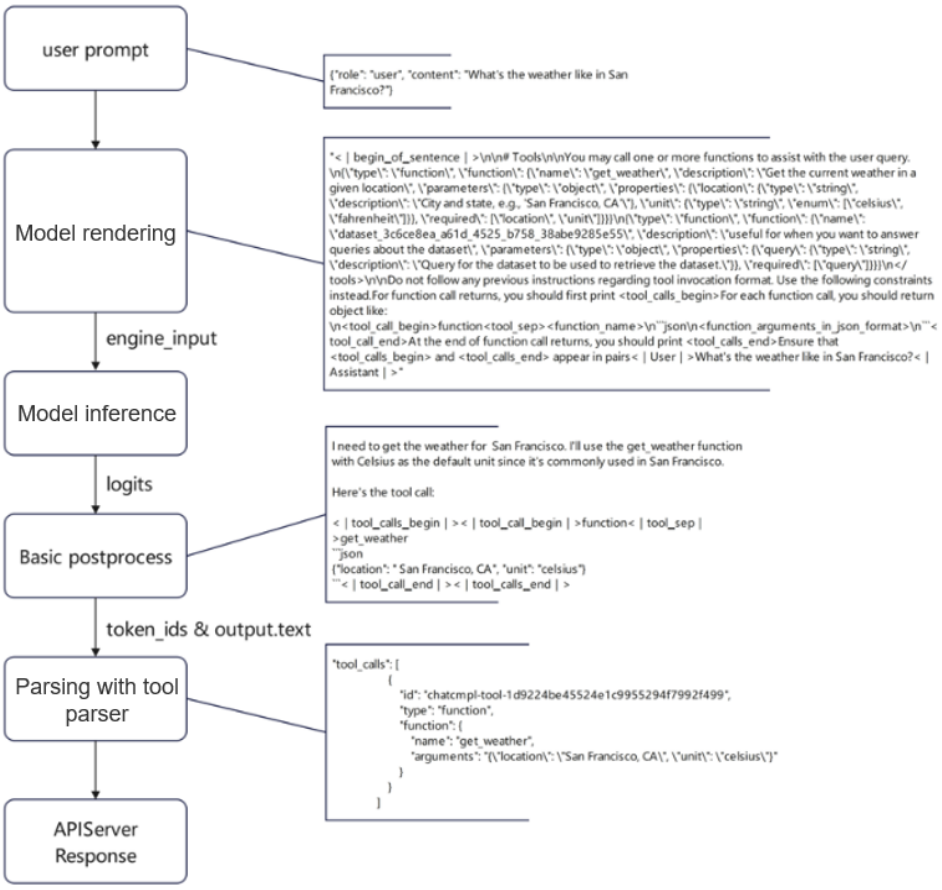
What Is Tool Calling?

Tool calling (also known as function calling or FC) links foundation models with external tools and APIs. It acts as a bridge between natural language and information systems by converting user requests into precise tool or API commands, effectively addressing their needs.

Working principle: Developers describe the functions and definitions of the tool to the model through natural language. The model determines whether to call the tool during the dialog. When the model needs to be called, the model returns the tool functions and input parameters that meet the requirements. The developer calls the tool and fills the result back to the model. The model summarizes or continues to plan subtasks based on the result.

Principles

FC uses three components: the pre-processing module (serving_chat), the post-processing module (tool parser), and chat_template. The serving_chat module formats the user input using chat_template, while the tool parser processes the model's response. The figure below illustrates this process.



1. The system detects that the tool definition is transferred in the user input and the auto tool choice option is enabled.
2. The system uses the chat template to render the user input, and constructs the tool information and user prompt in the model input.
3. The model performs inference and completes the basic post-processing process.
4. The tool parser parses the model output. If the model output contains tool call information, the tool parser parses the tool call information to the **tool_calls** field in the user response.

Specifications

Table 4-22 Model selection

Model	Advantage	Description
Qwen3-32B/ Qwen3-30B-A3B/ Qwen3-235B-A22B/ Qwen3-Coder-480B-A35B-Instruct	The performance and resource consumption are balanced, and the performance in MMLU (general knowledge) and C-Eval (Chinese evaluation) is excellent.	Qwen3-Coder-480B-A35B-Instruct tool_parser: qwen3_coder Qwen3-32B/ Qwen3-30B-A3B/ Qwen3-235B-A22B tool_parser: hermes

NOTE

1. The models in the preceding table have been verified. The capabilities supported by other released models are the same as those of the community.

2. FC does not enhance model capabilities. The model inference capability affects the FC performance and accuracy.

Tool Choice Modes

Tool choice supports two modes: **auto** for automatic identification and **named** for specific function calls. It does not support the **required** mode for mandatory function calls.

Tool choice is incompatible with speculative inference.

Mode	Supported	Description	Example
auto (recommended)	√	The model automatically identifies whether the tool needs to be called.	"tool_choice": "auto"
named	√	The system calls a specific function.	"tool_choice": {"type": "function", "function": {"name": "get_weather"}
required	×	The model must perform a function call.	"tool_choice": "required"

Supported Processing Modes

Mode	Supported	Description
Streaming output adaptation	√	Enables incremental retrieval of tool call data to enhance responsiveness.
Multi-turn tool call	√	Preserves conversational context across multiple tool calls, ensuring accurate execution and result fill-in.

Feature Compatibility

The table below lists the compatibility between function call and the existing key features of vLLM.

Feature	Compatible	Description	Compatible Error Handling
Reasoning Outputs	√	The chain-of-thought model generates detailed thought processes. For better performance during function calls, turning off this feature boosts efficiency and avoids errors when interpreting results due to unexpected outputs in **reasoning_content**. DeepSeek-R1: The chain of thought cannot be disabled. Qwen3: The chain of thought can be disabled.	-
Automatic Prefix Caching	√	-	-
Chunked Prefill	√	-	-
Speculative inference	√	-	-
Graph mode	√	-	-

NOTE

Auto mode's compatibility matches the previous section. Named mode has the same feature compatibility as guided decoding mode. Check the community feature compatibility matrix for specifics.

Basic Usage Process

1. Prepare the environment.

   Start the server with tool calling enabled. This example uses the Qwen3 model. You must use the hermes tool in the vLLM example directory to call the chat template.

   ```
   python -m vllm.entrypoints.openai.api_server \
   --model Qwen3/ \
   --chat-template=tool_chat_template_hermes.v1.jinja \
   --enable-auto-tool-choice \
   --tool-call-parser=hermes \
   --tool-parser-plugin=qwen3_tool_parser.py
   ```

2. Define tools.

   FC provides available tools to the model via the **tools** field in JSON format. The **tools** field includes information such as tool name, description, and parameter definitions. For details, see the tool parameter construction specifications.

   Define the Tool Function


```
def get_weather(location: str, unit: str):
    return f"Getting the weather for {location} in {unit}..."
```

- Assume a tool function named **get_weather** is defined in the code to retrieve weather information for a specified location.
  - **location**: The location for the weather query. This parameter is mandatory.
  - **unit**: The temperature unit for the returned result. This parameter is optional.
- Note: This example simulates a weather query scenario. In real applications, you should call an actual weather API.

Define Tools

```
"tools": [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "Get the current weather in a given location",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {"type": "string", "description": "City and state, e.g., 'San Francisco, CA'"},
                "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
            },
            "required": ["location"]
        }
    }
}]
```

- The **tools** field is a list, where each element represents a callable tool function. In this example, a tool named **get_weather** is defined.
- **type**: The type of the tool. This is a fixed value **function**, indicating a callable tool function.
- **function**: The function object that defines the tool's details, including name, description, and parameters.
  - **name**: The function name, here it is **get_weather**.
  - **description**: A description of the function, explaining its purpose.
  - **parameters**: The required parameters for the function, defined as an object containing **location** and **unit**.
    - **location**: Location information for the weather query. Type: string.
    - **unit**: Temperature unit. Type: string (enum). Options: **celsius** and **fahrenheit**.
    - **required**: Specifies required parameters. In this case, **location** is required.

3. Send a request.

   Include questions and required tools in your request. The model will detect and provide the necessary tools and their parameters based on these queries.

```
from openai import OpenAI
import json
client = OpenAI(base_url="http://localhost:8000/v1", api_key="dummy")
//User questions
messages = [{"role": "user", "content": "What's the weather like in San Francisco?"}]
tools = [
```



```
    {
        //For details, see the tools defined in Step 1.
    }
]
//Send a model request.
response = client.chat.completions.create(
    model=client.models.list().data[0].id,
    messages=messages ,
    tools=tools,
    tool_choice="auto"
)
```

4. Call external tools.

   Use the details about tools and parameters provided by the model to call the relevant external tool or API and retrieve its output.

   ```
   # Extract the tool orchestration parameters provided by the model.
   tool_call = response.choices[0].message.tool_calls[0].function
   # Tool names
   tool_name = tool_call.function.name
   # Execute the tools based on the tool names.
   if tool_name == "get_weather":
       # Extracted user parameters
       arguments = json.loads(tool_call.function.arguments)
       # Call the tools.
       tool_result = get_weather(**arguments)
   ```

   - Obtain the list of tools called by the model from **tool_calls** returned by the model.
   - Execute the tool based on the tool name. If the tool name is **get_weather**, extract user parameters and call the **get_weather** function to obtain the tool execution result.

5. Fill in execution results and generate final response.

   Return the tool execution result to the model as a message with **role=tool**. The model will then generate the final response based on this result.

   ```
   messages.append(completion.choices[0].message)
   messages.append({
       "role": "tool",
       "tool_call_id": tool_call.id,
       "content": tool_result
   })
   # Call the model to generate the final response.
   final_response = client.chat.completions.create(
       model=client.models.list().data[0].id,
       messages=messages ,
       tools=tools,
       tool_choice="auto"
   )
   print(final_response.choices[0].message.content)
   ```

Complete Code Example

```
from openai import OpenAI
import json
client = OpenAI(base_url="http://localhost:8000/v1", api_key="dummy")
def get_weather(location: str, unit: str):
    return f"Getting the weather for {location} in {unit}..."
tool_functions = {"get_weather": get_weather}
tools = [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "Get the current weather in a given location",
        "parameters": {
            "type": "object",
```



```
            "properties": {
                "location": {"type": "string", "description": "City and state, e.g., 'San Francisco, CA'"},
                "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
            },
            "required": ["location", "unit"]
        }
    }
}]
response = client.chat.completions.create(
    model=client.models.list().data[0].id,
    messages=[{"role": "user", "content": "What's the weather like in San Francisco?"}],
    tools=tools,
    tool_choice="auto"
)
tool_call = response.choices[0].message.tool_calls[0].function
print(f"Function called: {tool_call.name}")
print(f"Arguments: {tool_call.arguments}")
print(f"Result: {get_weather(**json.loads(tool_call.arguments))}")
```

Example output:

```
Function called: get_weather
Arguments: {"location": "San Francisco, CA", "unit": "fahrenheit"}
Result: Getting the weather for San Francisco, CA in fahrenheit...
```

Prompt Best Practices

The performance of Function Calling (FC) largely depends on the reasoning capabilities of the underlying model. To achieve efficient tool invocation, you should define prompts with clear and concise task descriptions, paired with well-structured tool function definitions. When using FC, follow these principles:

1. Keep task definitions simple and direct. Avoid including irrelevant or excessive information that may distract the model.
2. Avoid using the model for tasks that can be handled by code, as model reasoning is inherently probabilistic.

The table below lists common mistakes and recommended fixes.

Category	Issue	Incorrect Example	Corrected Example

Function definition	Poor function naming and vague description	{
		"type": "function",
		"function": {
		"name": "func1",
		"description": "Utility function"
		}
		}
			{
			"type": "function",
			"function": {
			"name": "CreateTask",
			"description": "Creates a task for the user and returns the task ID when a new work item is needed."
			}
			}

Parameter definition	Redundant structure and repetition	{
		"time": {
		"type": "object",
		"description": "City",
		"properties": {
		"city": {
		"description": "City"
		}
		}
		}
		}
			{
			"time": {
			"type": "string",
			"description": "City"
			}
			}

Category	Issue	Incorrect Example	Corrected Example

	Unnecessary input parameters with fixed values	{
		"time": {
		"type": "object",
		"description": "City",
		"properties": {
		"city": {
		"description": "Always pass Hangzhou"
		}
		}
		}
		}
			If an input parameter is fixed and not necessary, it should be removed and handled directly in code.

prompt	Prompt complexity leading to redundant tool calls	System prompt: You are communicating with user Marvin. You need to first query the user ID, then use it to create a task...	System prompt: You are communicating with user Marvin (ID=123). You can use the user ID to create a task...
	Ambiguous task definitions	System prompt: You can use the ID to find the user and get the task ID. The model struggles to tell the two IDs apart, which can lead to incorrect usage by the model.	System prompt: Each user has a unique user ID; each task has a task ID. You can use the user ID to retrieve user information and obtain all associated task IDs.
	Mismatch between task definition and tool function	Tool function requires location and date. Prompt: Query the weather in Beijing. The model fails to call the function if the task lacks date details or uses the default date value, leading to incorrect results.	Prompt: Query the weather in Beijing on July 30, 2025.
	Format conflicts	If the system prompt specifies a return format that conflicts with the function call, it may cause tool call failures.	Remove any formatting instructions that interfere with function execution.

Best Practices of Tool Function Definition

The model needs correct tool function definitions to work properly. These definitions must follow the **JSON Schema** standards. Build the **tools** object using these guidelines:

Structure of **tools**


```
"tools": [
  {
    "type": "function",
    "function": {
      "name": "...", // Function name (lowercase letters and underscores)
      "description": "...", // Function description
      "parameters": { ...
      } // Function parameters (in JSON Schema format)
    }
  }
]
```

- *type*: The type of the tool. This is a fixed value **function**, indicating a callable tool function.
- *function*: Function object, which is used to define the name, description, and parameters of a tool function.

Description

function

Field	Type	Mandatory	Description
name	string	Yes	Function name, which is unique. Use lowercase letters and underscores (_).
description	string	Yes	A description of the function, explaining its purpose.
parameters	object	Yes	Function parameters, which must comply with the JSON Schema format.

parameters

parameters must comply with the **JSON Schema** format.

```
"parameters": {
  {
    "type": "object",
    "properties": {
      "name": {
        "type": "string | number | boolean | object | array | integer",
        "description": "Attribute description"
      }
    },
    "required": [
      "Mandatory parameters"
    ]
  }
}
```

- *type*: The value must be **object**.
- *properties*: All supported attributes and their types.
  - *name*: Parameter name, which must be an English string and must be unique.
  - *type*: The value must comply with the **JSON Schema specifications**. The supported types include string, number, boolean, integer, object, and array.

- *required*: Parameters that are mandatory in the function.
- Other parameters vary depending on the value of *type*. For details, see the following table.

Type	Example
String, integer, number, and boolean	N/A

Object
• *description*: description
• *properties*: object attributes
• *required*: mandatory attributes
	• Example 1: Querying students' personal information
	"parameters": {
	"type": "object",
	"description": "Student information",
	"properties": {
	"age": {
	"type": "integer",
	"description": "Age"
	},
	"gender": {
	"type": "string",
	"description": "Gender"
	},
	"married": {
	"type": "boolean",
	"description": "Party member"
	}
	},
	"required": [
	"age"
	],
	}

Type	Example

List of arrays
• *description*: description
• *"items": {"type": ITEM_TYPE}*: data type of elements within an array
	• Example 1: Define a text array.
	"parameters": {
	"type": "array",
	"description": "Any number of texts",
	"items": {
	"type": "string"
	}
	}
	• Example 2: Define a two-dimensional array.
	"parameters": {
	"type": "array",
	"description": "Two-dimensional matrix",
	"items": {
	"type": "array",
	"items": {
	"type": "number"
	}
	},
	}
	• Example 3: Use **array** to implement multi-select arrays.
	"parameters": {
	"description": "Hobbies. Multiple options can be selected.",
	"type": "array",
	"items": {
	"type": "string",
	"description": """Options:
	"Swimming",
	"Running",
	"Climbing",
	"Fitness",
	"Badminton",
	"Basketball". """,
	},
	}

Complete Example

```
"tools": [
  {
    "type": "function",
    "function": {
      "name": "get_weather",
      "description": "Get weather information for a specified location",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string",
            "description": "City and state, e.g., 'San Francisco, CA'"
          }
        },
        "required": [
          "location"
        ]
      }
    }
  }
]
```


Notes

1. **Case-sensitive**: All field names and parameter names are case-sensitive (lowercase is recommended).
2.
3. **Schema compliance**: All definitions must adhere to JSON Schema standards and should be verifiable through JSON Schema validators.

Best Practices

1. **Core Guidelines for Tool Descriptions**
   - Tool descriptions should clearly articulate the tool's capabilities, parameter meanings and impact, applicable scenarios (or restricted use cases), and any constraints (like input length limits). Each tool should be described in 3 to 4 concise sentences.
   - Prioritize completeness in function and parameter definitions; examples are supplementary and should be added cautiously, especially when used with reasoning models.
2. **Function Design Principles**
   - **Naming and parameters**: Function names should be intuitive (like **parse_product_info**). Parameter descriptions should include both format (like **city: string**) and business meaning (like **Full name of the target city**). Clearly specify expected output (like **Return weather data in JSON format**).
   - **System prompt**: Use system prompts to define call conditions (like "Trigger **get_product_detail** when the user asks about product information").
   - **Engineering design**:
     - Principle of least astonishment: Use enumerated types (like *StatusEnum*) to avoid invalid inputs and ensure logical clarity.
     - Clarity principle: Prompts should be phrased clearly enough for humans to make intuitive decisions.
   - **Call optimization**:
     - Pass known parameters implicitly through code (for example, *submit_order* should not require repeated declaration of *user_id*).
     - Merge fixed-process functions (for example, combine *query_location* and *mark_location* into *query_and_mark_location*).
   - **Quantity and performance**: Limit the number of functions to 20 or fewer. Use debugging tools to iteratively refine function schemas. For complex scenarios, consider leveraging fine-tuning to improve accuracy.

Request practice

1. Tool calls

   Example request:

```
{
   "model": "deepseek",
   "messages": [
      {
```



```
            "role": "user",
            "content": "Obtain the weather in Beijing"
        }
    ],
    "tools": [
        {
            "type": "function",
            "function": {
                "name": "get_weather",
                "description": "Query weather",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "location": {
                            "type": "string",
                            "description": "City"
                        },
                        "unit": {
                            "type": "string",
                            "enum": [
                                "celsius",
                                "fahrenheit"
                            ]
                        }
                    },
                    "required": [
                        "location",
                        "unit"
                    ]
                }
            }
        },
        {
            "type": "function",
            "function": {
                "name": "send_email",
                "description": "Send an email",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "userInput": {
                            "type": "string",
                            "description": "Email content"
                        }
                    },
                    "required": [
                        "userInput"
                    ]
                }
            }
        }
    ],
    "tool_choice": "auto",
    "temperature": 0,
    "stream": false
}
```

Request result:

```
{
    "id": "1530/chat-c7277abfbc724677a570c03c7541edd7",
    "object": "chat.completion",
    "created": 1753423691,
    "model": "deepseek",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": null,
                "reasoning_content": null,
```



```
            "tool_calls": [
               {
                  "id": "chatcmpl-tool-6714630cc3fc4551a156aa48715d5139",
                  "type": "function",
                  "function": {
                     "name": "get_weather",
                     "arguments": "{\"location\":\"Beijing\",\"unit\":\"celsius\"}"
                  }
               }
            ]
         },
         "logprobs": null,
         "finish_reason": "tool_calls",
         "stop_reason": null
      }
   ],
   "usage": {
      "prompt_tokens": 309,
      "total_tokens": 359,
      "completion_tokens": 50
   },
   "prompt_logprobs": null
}
```

2. Tool summary

   Example request:

```
{
   "model": "deepseek",
   "messages": [
            {
               "role": "user",
               "content": "Obtain the weather in Beijing"
            },
            {
               "role": "assistant",
               "tool_calls": [
                  {
                     "id": "chatcmpl-tool-fc6986a3dc014e80a5d3e091c60648d9",
                     "type": "function",
                     "function": {
                        "name": "get_weather",
                        "arguments": "{\"location\": \"Beijing\", \"unit\": \"celsius\"}"
                     }
                  }
               ]},
            {
               "role": "tool",
               "tool_call_id": "chatcmpl-tool-fc6986a3dc014e80a5d3e091c60648d9",
               "content": "Beijing's temperature today ranges from 20 to 50 degrees.",
               "name": "get_weather"
             }
   ],
   "tools": [
      {
         "type": "function",
         "function": {
            "name": "get_weather",
            "description": "Query weather",
            "parameters": {
               "type": "object",
               "properties": {
                  "location": {
                     "type": "string",
                     "description": "City"
                  },
                  "unit": {
                     "type": "string",
                     "enum": [
                        "celsius",
```



```
                        "fahrenheit"
                    ]
                }
            },
            "required": [
                "location",
                "unit"
            ]
        }
    }
},
{
    "type": "function",
    "function": {
        "name": "send_email",
        "description": "Send an email",
        "parameters": {
            "type": "object",
            "properties": {
                "userInput": {
                    "type": "string",
                    "description": "Email content"
                }
            },
            "required": [
                "userInput"
            ]
        }
    }
}
],
"tool_choice": "auto",
"temperature": 0,
"stream": false
}
```

Request result:

```
{
    "id": "1530/chat-2966628beae0430b872b994f7ef0f9b4",
    "object": "chat.completion",
    "created": 1753423849,
    "model": "deepseek",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "Beijing's temperature today ranges from 20 to 50 degrees.",
                "reasoning_content": null,
                "tool_calls": []
            },
            "logprobs": null,
            "finish_reason": "stop",
            "stop_reason": null
        }
    ],
    "usage": {
        "prompt_tokens": 387,
        "total_tokens": 398,
        "completion_tokens": 11
    },
    "prompt_logprobs": null
}
```

Exception Handling

JSON format error tolerance mechanism

If the JSON format is slightly invalid, use the **json-repair** library for fault tolerance.


```
import json_repair

invalid_json = '{"location": "Beijing", "unit": "°C"}'
valid_json = json_repair.loads(invalid_json)
```

Tool Call Exception

If the model fails to call the tool function properly because of the prompt or function definition, improve both using best practices.

Model Return Error

FC relies on the model's capabilities. Because of model hallucination, its output might not match expectations, causing the tool to fail.

If the call failure rate is low in a certain scenario, you can set the retry mechanism to harden the reliability.

Q&A

Q: Why does DeepSeek-R1-0528 return the **<result>** tag in **content** in non-streaming mode?

A: This issue stems from the model's automatic tag generation. It does not impact FC, so you can safely disregard it.

Q: Why does the **</think>** tag occasionally appear in **reasoning_content** when deep thinking is enabled in chain-of-thought models such as DeepSeek-R1?

A: The DeepSeek-R1 model sometimes outputs unnecessary **</think>** tags due to a known issue. This can cut off parts of the chain-of-thought process. While there is no fix yet, adding a retry may help.

Q: Why does the returned **reasoning_content** contain the **<tool_calls_begin>** tag when deep thinking is enabled in the chain-of-thought models such as DeepSeek-R1?

A: Special characters are used in the preset prompt template of FC, which does not affect the normal use of FC. You can ignore the special characters.

Q: When FC is used, only one function is called. Why are multiple function calls returned?

A: The model calculates how many times a function is called. This often happens when prompts or tool functions are not clear. To fix this, improve your prompts and tool definitions using recommended practices.

Q: When FC is used, why cannot **tool_calls** be parsed after **max_tokens** is configured?

A: If the value of **max_tokens** is too small, the model output may be truncated and the **tool_calls** content cannot be properly parsed. Therefore, you need to set **max_tokens** to a proper value.

4.1.6.8 Reasoning Outputs

Scenarios

Reasoning outputs support DeepSeek-R1 and Qwen3 series. They produce outputs with detailed inference steps and final results. Using reasoning outputs adds a

reasoning_content field to show the thought process and logic behind each conclusion.

Supported Models

Series	Parser
DeepSeek-R1	deepseek_r1
QwQ-32B	deepseek_r1
Qwen3	qwen3

Constraints

Reasoning outputs only apply to the **/v1/chat/completions** API of OpenAI.

Enabling Reasoning Outputs

Add the following command when **starting the inference service**.

```
--enable-reasoning --reasoning-parser xxx
```

Note: **XXX** indicates the name of the reasoning parser that matches the model.

Disabling Model Chain-of-Thought Output

Currently, only the Qwen3 series models support disabling the chain-of-thought output. You can do this by adding the template parameter **"enable_thinking": false** when making an inference request. The request body example is as follows:

```
{
  "model": "Qwen3-8B",
  "chat_template_kwargs": {
    "enable_thinking": false
  },
  "messages": [
    {
      "role": "user",
      "content": "Hello"
    }
  ],
  "temperature": 0,
  "stream": false
}
```

Removing the Max Token Limit for Reasoning Content

Ascend-vLLM supports enabling or disabling the ability to remove the max token limit for reasoning content by setting an environment variable before starting the inference service. The environment variable usage example is as follows

```
export ENABLE_MAX_TOKENS_EXCLUDE_REASONING=1
```

- Not setting the environment variable or setting it to **0**: The **max token** parameter will control and truncate the length of the **reasoning content** field. This behavior is consistent with the community standard.

- Setting the environment variable to **1**: The **max token** parameter will not control or truncate the **reasoning content** field. It will only control the length of the **content** field.

4.1.7 Inference Service Accuracy Evaluation

This chapter introduces three accuracy evaluation methods: OpenCompass, Simple-evals, and MME tools.

- OpenCompass supports evaluation schemes for over 20 Hugging Face and API models, more than 70 datasets, and approximately 400,000 questions, enabling one-stop evaluation of various large language models (LLMs).
- Simple-evals is a lightweight library for evaluating language models. It can be used to evaluate datasets such as MMLU, GPQA, DROP, MGSM, and HumanEval. This tool is designed for online evaluation and uses the OpenAI API by default.
- MME is suitable for accuracy testing of multimodal models. Currently supported models include qwen2-vl-2B, qwen2-vl-7B, qwen2-vl-72B, qwen2.5-vl-7B, qwen2.5-vl-32B, qwen2.5-vl-72B, internvl2.5-26B, InternVL2-Llama3-76B-AWQ, and gemma3-27B.

Using the OpenCompass Accuracy Evaluation Tool for LLM Accuracy Evaluation

Use OpenCompass for online service accuracy evaluation.

1. Prepare the OpenCompass runtime environment using conda (recommended).
   ```
   conda create --name opencompass python=3.10 -y
   conda activate opencompass
   ```
2. Install OpenCompass.
   ```
   git clone https://github.com/open-compass/opencompass
   cd opencompass
   pip install -e .
   ```
3. Download supported datasets.
   ```
   # Download datasets to the data/ directory.
   wget https://github.com/open-compass/opencompass/releases/download/0.2.2.rc1/OpenCompassData-core-20240207.zip
   unzip OpenCompassData-core-20240207.zip
   ```

   You can also query supported models and datasets using the following command:

   ```
   python tools/list_configs.py [PATTERN1] [PATTERN2] [...]
   ```

   If no parameters are provided, it will list all model configurations in **configs/models** and **configs/dataset**.

   You can pass any number of parameters, and the script will list all configurations related to the provided strings, supporting fuzzy search and * wildcard matching. For example, the following command will list all configurations related to mmlu and llama:

   ```
   python tools/list_configs.py mmlu llama
   ```
4. Build configuration files.

   OpenCompass allows you to write complete experiment configurations in configuration files and run them directly using **run.py**. Configuration files are organized in Python format and must include the **datasets** and **models** fields.

Create a new **example.py** file in **opencompass/examples/**. This configuration file imports the required dataset and model configurations using inheritance and combines the **datasets** and **models** fields in the desired format.

```
from mmengine.config import read_base
from opencompass.models import OpenAI

with read_base():
    from opencompass.configs.datasets.gsm8k.gsm8k_gen import gsm8k_datasets

datasets = gsm8k_datasets

models  = [dict(
    abbr='Qwen3-32B-W8A8',
    type=OpenAI,
    path='Qwen3-32B-W8A8',
    tokenizer_path='/Qwen/Qwen3-32B-W8A8',
    key='EMPTY',
    openai_api_base='http://127.0.0.1:8091/v1/chat/completions',
    temperature=0.6,
    query_per_second=1,
    max_out_len=31744,
    max_seq_len=31744,
    batch_size=8
)]
```

NOTE

Some generative datasets have a default **max_out_len=512** configuration, which may truncate results before the answer is fully generated, leading to low scores. You can update the **datasets** configuration in the **example.py** configuration file.

Example:

```
gsm8k_datasets[0]["infer_cfg"]["inferencer"].pop("max_out_len")
```

Parameters:

- **abbr**: model abbreviation
- **type**: model type
- **path**: registered model name
- **tokenizer_path**: tokenizer directory (defaults to **path** if not specified)
- **key**: model access key
- **openai_api_base**: model service address
- **temperature**: generation temperature
- **query_per_second**: service request rate
- **max_out_len**: maximum output length
- **max_seq_len**: maximum input length
- **batch_size**: batch size

5. Run the accuracy test task.

```
python run.py examples/example.py -w ./outputs/demo
```

More parameters in **run.py**:

The following are some parameters related to evaluation that can help you configure more effective inference tasks based on your environment:

- **-w outputs/demo**: Directory to save evaluation logs and results. In this case, the experiment results will be saved to **outputs/demo/***{TIMESTAMP}*.
- **-r {TIMESTAMP/latest}**: Reuses existing inference results and skips completed tasks. If a timestamp is provided, it will reuse the results from

that timestamp in the workspace path; if **latest** or nothing is specified, it will reuse the latest results in the specified workspace path.

- **--mode all**: Specifies a particular stage of the task.
  - **all**: (default) Performs full evaluation, including inference and evaluation.
  - **infer**: Performs inference on each dataset.
  - **eval**: Evaluates based on inference results.
  - **viz**: Displays evaluation results only.
- **--max-num-workers**: Maximum number of parallel tasks.
- **--debug**: Runs tasks in debug mode. Tasks will be executed sequentially and output will be printed in real-time, which is useful for troubleshooting and ideal for initial task execution.

6. Evaluate the results.

   After the script runs, the test results are output to the terminal. All run outputs are directed to the **outputs/demo/** directory, with the following structure:

```
outputs/default/
├── 20250220_120000
├── 20250220_183030     # Each experiment in a separate folder
│   ├── configs         # Dumped configuration files for recording. If different experiments are rerun in the same experiment folder, multiple configurations may be retained.
│   ├── logs            # Log files for the inference and evaluation stages
│   │   ├── eval
│   │   └── infer
│   ├── predictions   # Inference results for each task
│   ├── results       # Evaluation results for each task
│   └── summary       # Summary evaluation results for a single experiment
├── ...
```

Using the Simple-evals Accuracy Evaluation Tool for LLM Accuracy Evaluation

Simple-evals is an online service accuracy evaluation tool that can be used seamlessly with any framework that supports OpenAI.

1. Install the Simple-evals evaluation tool. Create a new python environment using the conda environment in the package. The following uses accuracy as an example.

```
conda create -n accuracy --clone python-3.11.10
conda activate accuracy
cd xxx/simple_evals
bash build.sh
```

2. Run an online accuracy evaluation task.

```
# Debug mode
python simple_evals.py --model $model --dataset gpqa \
--served-model-name $served_model_name \
--url http://localhost:$port/v1 \
--max-tokens 128 \
--temperature 0.6 \
--num-threads 32 \
--debug

# start
python simple_evals.py --model $model --dataset gpqa \
```



```
--served-model-name $served_model_name \
--url http://localhost:$port/v1 \
--max-tokens 16384 \
--temperature 0.6 \
--num-threads 32
```

Parameters:

- **model**: The model to be evaluated, which affects the generated file name. For example, Qwen3-32B will generate **gpqa_Qwen3-32B_20250719_130703.json** and **gpqa_Qwen3-32B_20250719_130703.html** in the **results/** directory. The JSON file records the score, and the HTML file can be used to view detailed results.
- **dataset**: The dataset to be evaluated, supporting mmlu, gpqa, mgsm, drop, and humaneval.
- **served_model_name**: If the service to be evaluated supports OpenAI, it will have a **served_model_name**. This needs to be included in the request.
- **port**: Supports evaluation of local and online services. For local services, the URL is generally **localhost:8080**, where **8080** is the port. For online services, the service provider will provide the URL and **served_model_name** in OpenAI format. Note that the URL should end with **/v1**.
- **max-tokens**: The maximum number of tokens to generate. As the output becomes longer with the support of chain-of-thought or other features, it is recommended to set this to 16384.
- **temperature**: Affects the generated results; it is recommended to keep it unchanged.
- **num-threads**: The number of concurrent requests sent to the service. Within the supported range, a higher concurrency reduces the time required. The recommended value is **32**.
- **debug**: Due to the high resource consumption of evaluation, a debug mode is provided to verify the successful installation. It sends a small number of requests to complete the entire process. It is recommended to use debug mode for the first run to quickly complete the process.

3. Result description

   The results are generated in the current directory of **simple-evals**. Scores are saved in JSON files, and detailed results are saved in HTML files. For **gpqa_Qwen3-32B_20250719_130703.json**, **gpqa** is the dataset used for evaluation, **Qwen3-32B** is the LLM being evaluated, and **20250719_130703** is the timestamp when the evaluation was executed.

Using the MME Accuracy Evaluation Tool for Multimodal Model Accuracy Evaluation

1. Obtain the MME dataset.

   Obtain the **MME evaluation dataset** and upload it to the directory **llm_tools/llm_evaluation/mme_eval/data/eval/**.

2. Obtain the accuracy test code. The accuracy test code is located in the **llm_tools/llm_evaluation/mme_eval** directory of the code package **AscendCloud-LLM**. The directory structure is as follows:


```
mme_eval
├──metric.py       # MME accuracy test script
├──MME.sh          # Script to run MME
```

3. Run the MME accuracy test script.

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

Parameters:

a. **MODEL_PATH**: Path to the model weights. The default value is empty.

b. **MME_PATH**: Path to the MME dataset. The default value is the current path.

c. **MODEL_TYPE**: Model type. Currently supported model types include: llava, llava-next, minicpm, qwen-vl, internvl2, qwen2-vl, and llava-onevision.

d. **OUTPUT_NAME**: Name of the output result file. The default value is llava.

e. **ASCEND_RT_VISIBLE_DEVICES**: Indicates support for multiple model service instances and model parallelism, such as **0,1:2,3**. The default is device 0.

f. **QUANTIZATION**: Quantization option. If not provided, the default value is **None** (quantization is not enabled). Supported values include w4a16, which requires corresponding weights.

g. **GPU_MEMORY_UTILIZATION**: Ratio of the GPU memory used by the NPU. The input parameter name of the original vLLM is reused. The default value is **0.9**.

After the script runs, the test results are output to the terminal.

4.1.8 Inference Service Performance Evaluation

4.1.8.1 LLM Inference Performance Test

The acs-bench tool is required for the performance test. The **acs-bench prof** command is used to run an LLM performance benchmark test. You can set the data length and quantity to evaluate Ascend-vLLM service performance under various request loads. It supports both ramp-up tests and performance stress tests.

Installing the acs-bench Tool

The ModelArts 6.5.906 and newer versions include the pre-installed **acs_bench-1.0.1-py3-none-any.whl** package for the acs-bench tool. No separate installation is required.

Check if the acs-bench tool is already installed:

```
$ pip show acs-bench
```

To install the acs-bench tool, follow these steps:

1. Obtain the acs-bench tool's whl package. The whl package is located in the **llm_tools** directory within the **AscendCloud-LLM-xxx.zip** software package.

The acs-bench tool should be installed in the Python runtime environment, which can access the inference service to be tested. It is recommended to perform this operation in the container where the inference service is started.

2. (Optional) Configure the pip source according to your actual needs.

```
$ mkdir -p ~/.pip
$ vim ~/.pip/pip.conf
# Add the following content to the configuration file. The example below uses the Huawei source:
[global]
index-url=https://mirrors.tools.huawei.com/pypi/simple
trusted-host=mirrors.tools.huawei.com
timeout = 120
```

3. (Optional) Install the acs-bench tool:

```
$ pip install llm_tools/acs_bench-*-py3-none-any.whl
```

Preparations: Configuring providers.yaml

The acs-bench tool accesses the server through the **providers.yaml** configuration file, which contains information such as the server's id, name, api_key, base_url, model_name, and model_category.

Before using the acs-bench tool, create a **providers.yaml** file locally, fill in the parameter values according to your actual situation, and save it. The following is an example:

```
providers:
  -
    id: 'ascend-vllm'
    name: 'ascend-vllm'
    api_key: 'EMPTY'
    base_url: 'http://server_ip:port/v1'
    model_name: 'Qwen3-32b'
    model_category: 'Qwen3-32b'
```

Table 4-23 describes the parameters.

Table 4-23 Parameters in the **providers.yaml** file

Field	Mandatory	Description
id	No	Identifier for the service provider.
name	No	Name of the service provider.
api_key	No	Originally the api_key for OpenAI, now it can be used as the MaaS authentication code.
base_url	Yes	Base URL of the server (add a URL similar to **http://***{$IP_address}*:*{$port}*/**v1**).
model_name	Yes	Model name used when starting the inference service. If the **served-model-name** parameter is set when starting the inference service, use the value of **served-model-name**. If the **served-model-name** parameter is not set, use the default model path when starting the service.

Field	Mandatory	Description
model_category	No	Category of the model, can be omitted.

Obtaining Datasets

This section introduces how to obtain datasets.

The acs-bench tool requires datasets for testing and currently supports open-source datasets such as LongBench and ShareGPT formats. If the open-source datasets are not available locally, you can use the **acs-bench generate dataset** command to generate custom datasets. Below are examples of how to use this command. For details about the parameter descriptions, see **Dataset Generation Parameter Description**.

1. Generate a random dataset.
```
$ acs-bench generate dataset \
--tokenizer ./tokenizer/Qwen3-32b \
--dataset-type random \
--output-path ./built_in_dataset \
--input-length 128 \
--num-requests 100
```
2. Generate an embedding dataset.
```
$ acs-bench generate dataset \
--tokenizer ./tokenizer/Qwen3-32b \
--task embedding \
--output-path ./built_in_dataset \
--input-length 128 \
--num-requests 100
```
3. Generate a reranking dataset.
```
$ acs-bench generate dataset \
--tokenizer ./tokenizer/Qwen3-32b \
--task rerank \
--document-size 4 \
--output-path ./built_in_dataset \
--input-length 128 \
--num-requests 100
```
4. Filter datasets from LongBench.
```
$ acs-bench generate dataset \
--tokenizer ./tokenizer/Qwen3-32b \
--dataset-type LongBench \
--input-path ./dataset/long_bench --output-path ./built_in_dataset \
--input-length 128 \
--num-requests 100
```
5. Filter datasets from ShareGPT.
```
$ acs-bench generate dataset \
--tokenizer ./tokenizer/Qwen3-32b \
--dataset-type ShareGPT \
--input-path ./dataset/ShareGPT --output-path ./built_in_dataset \
--input-length 128 \
--num-requests 100
```

- LongBench download link: **https://huggingface.co/datasets/zai-org/LongBench/tree/main**
- ShareGPT download link: **https://huggingface.co/datasets/shibing624/sharegpt_gpt4**

NOTICE

The **--input-length** and **--num-requests** parameters in the dataset generation command only support single values.

If you need to generate datasets with different specifications, modify the **--input-length** or **--num-requests** parameters to the desired values and then execute the command.

Dataset Generation Parameter Description

The command to query the dataset generation parameters is as follows:

```
$ acs-bench generate dataset -h
```

Table 4-24 Custom dataset generation command parameters

Parameter	Type	Mandatory	Description
-dt/--dataset-type	String	No	Specifies the source of the dataset to be generated, which is the type of open-source dataset used for data filtering. The default value is **random**, which means a random token combination mode.
-i/--input-path	String	No	Specifies the path to the open-source dataset used for data filtering. This parameter is not required when **--dataset-type** is set to **random**.
-mt/--modal-type	String	No	The mode type for multimodal datasets. The default value is **text**. Current options are **text**, **image-text**, and **video-text**.
-tk/--task	String	No	The task backend for the dataset. The default value is **generate**. Current options are **generate**, **rerank**, and **embedding**.
-cfg/--config-option	String	No	Multimodal configuration options, specified in the form of "KEY:VALUE" pairs. Multiple "KEY:VALUE" pairs can be provided. Allowed key options include **image_height**, **image_width**, **duration**, and **fps**.
-o/--output-path	String	Yes	The output path for the generated JSON file containing prompts.
-il/ --input-length	Int	Yes	The length of each prompt in the custom dataset.

Parameter	Type	Mandatory	Description
-pl/--prefix-length	Int	No	The length of the common prefix prompt in the custom dataset, effective only in random mode. The default is **0**.
-n/--num-requests	Int	Yes	The number of prompts to be generated.
-ds/--document-size	Int	No	The document size for each query. The default is **4**.
-t/ --tokenizer	String	Yes	The path to the tokenizer model folder, supporting both local paths and Hugging Face model paths.
-rv/--revision	String	No	Specifies the model branch in the Hugging Face community, applicable only when the tokenizer is a Hugging Face model path. The default is **master**.
-ra/--range-ratio-above	Float	No	The ratio by which the prompt length can dynamically increase. The maximum length is input_length x (1 + **range_ratio_above**). The value range is [0, 1]. The default is **0.0**.
-rb/--range-ratio-below	Float	No	The ratio by which the prompt length can dynamically decrease. The minimum length is input_length x (1 – **range_ratio_above**). The value range is [0, 1]. The default is **0.0**.
-seed/--random-seed	Int	No	The random seed used to fix randomness.
-trc/--trust-remote-code	Bool	No	Specifies whether to trust remote code, applicable only when the tokenizer is a Hugging Face model path. The default is **False**.

Performance Stress Testing Mode Verification

An example of using the **acs-bench prof** command for performance stress testing is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Use a thread pool for concurrent testing. The default backend concurrency mode is threading-pool. You can also choose the asynchronous coroutine concurrency mode asyncio, the multi-process mode processing-pool, or the multi-thread mode threading-pool.
$ acs-bench prof \
--provider ./provider/providers.yaml \
--dataset-type custom --input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
```



```
--epochs 2 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,128,2048,2048 --output-length 128,2048,128,2048 \
--benchmark-save-path ./output_path/
```

Ramp-Up Mode Verification

An example of using the **acs-bench prof** command for ramp-up testing is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Example using the multi-thread concurrency mode, starting with a concurrency of 1 and increasing by 2 every 5,000ms.
$ acs-bench prof \
--provider ./provider/providers.yaml \
--dataset-type custom --input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--use-climb --climb-mode linear --growth-rate 2 --init-concurrency 1  --growth-interval 5000 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,128,2048,2048 --output-length 128,2048,128,2048 \
--benchmark-save-path ./output_path/
```

Concurrent Testing of Embedding Models Using an Embedding Dataset

An example of using the **acs-bench prof** command for concurrent testing of an embedding model with an embedding dataset is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Example using multi-thread concurrency mode, with the backend set to embedding
$ acs-bench prof \
--provider ./provider/providers.yaml \
--dataset-type custom --input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend embedding --warmup 1 \
--epochs 2 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,128,2048,2048 --output-length 128,2048,128,2048 \
--benchmark-save-path ./output_path/
```

Concurrent Testing of Reranking Models Using a Rerank Dataset

An example of using the **acs-bench prof** command for concurrent testing of a reranking model with an embedding dataset is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Example using multi-thread concurrency mode, with the backend set to rerank
$ acs-bench prof \
--provider ./provider/providers.yaml \
--dataset-type custom --input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend rerank --warmup 1 \
--document-size 4,4,4,4 \
--epochs 2 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,128,2048,2048 --output-length 128,2048,128,2048 \
--benchmark-save-path ./output_path/
```


Parameter Descriptions for Usage Example

NOTICE

1. The parameters **--concurrency**, **--init-concurrency**, **--num-requests**, **--input-length**, **--output-length**, and **--config-option** can be specified using either a comma-separated list (without spaces between commas) or by specifying multiple groups of parameters separately. If the **--num-requests** parameter is not specified, it defaults to the same value as the **--concurrency** parameter.

Example:

```
$ acs-bench prof \
--provider ./provider/providers.yaml \
--dataset-type custom --input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--num-requests 1 --num-requests 2 --num-requests 4 --num-requests 8 \
--concurrency 1 --concurrency 2 --concurrency 4 --concurrency 8 \
--input-length 128 --input-length 128 --input-length 2048 --input-length 2048 \
--output-length 128 --output-length 2048 --output-length 128 --output-length 2048 \
--benchmark-save-path ./output_path/
```

2. The **--input-length** parameter values in the performance stress testing and ramp-up testing examples must exist in the pre-generated dataset. If they do not exist, refer to **Obtaining Datasets** to generate a dataset with the corresponding input length.

The performance benchmark test parameters are primarily composed of four parts: **Dataset Options**, **Concurrency Options**, **Metrics Options**, and **Serving Options**. The following sections will introduce these configuration types.

Query acs-bench test parameters:

```
$ acs-bench prof -h
```

Table 4-25 Dataset Options

Parameter	Type	Mandatory	Description
-dt/--dataset-type	String	No	Specifies the type of dataset, the default value is **custom**, which means a user-defined dataset.
-cfg/--config-option	String	No	Multimodal configuration options, specified in the form of "KEY:VALUE" pairs. Multiple "KEY1:VALUE1,KEY2:VALUE2" pairs can be provided. Allowed key options include **image_height**, **image_width**, **duration**, and **fps**.
-mt/--modal-type	String	No	The mode type for multimodal datasets. The default value is **text**. Current options are **text**, **image-text**, and **video-text**.

Parameter	Type	Mandatory	Description
-i/--input-path	String	Yes	Specifies the path for the dataset.
-il/--input-length	Int	Yes	Specifies the length of the custom dataset, only effective when **dataset-type** is **custom**, supports specifying multiple input lengths, separated by ",".
-ds/--document-size	Int	No	The document size for each query. This parameter supports multiple integer inputs.
-n/--num-requests	Int	Yes	The number of requests for concurrent testing. This parameter supports specifying multiple request numbers, separated by ",". It defaults to the same as the concurrency number.
-t/--tokenizer	String	No	The path to the tokenizer model folder, supporting both local paths and Hugging Face model paths.
-rv/--revision	String	No	Specifies the model branch in the Hugging Face community, applicable only when the tokenizer is a Hugging Face model path. The default is **master**.
-seed/--random-seed	Int	No	The random seed used to fix randomness.
-trc/--trust-remote-code	Bool	No	Specifies whether to trust remote code, applicable only when the tokenizer is a Hugging Face model path. The default is **False**.

Table 4-26 Concurrency Options

Parameter	Type	Mandatory	Description
-c/--concurrency	Int	No	Maximum concurrency level. The default value is **1**. The parameter supports specifying multiple concurrency levels, separated by ",".

Parameter	Type	Mandatory	Description
-nc/--num-process	Int	No	Number of processes for parallel processing, which should be less than or equal to the number of CPUs. This parameter supports specifying multiple process numbers, separated by ",". The default value is **[1]**.
-r/--request-rate	Float	No	Request arrival rate, only effective when concurrency is **1**. The default value is infinity (**INF**).
-rm/--request-mode	String	No	Request arrival mode, which supports **normal** and **pd-adaptive**. The default value is **normal**.
-pc/--prefill-concurrency	INT	No	Maximum concurrency for all prefill operations in PD aggregation, only effective when **--request-mode** is **pd-adaptive**.
-dc/--decoder-concurrency	INT	No	Maximum concurrency for all decode operations in PD aggregation, only effective when **--request-mode** is **pd-adaptive**.
-burst/--burstiness	Float	No	Burst factor for requests, only effective when **request_rate** is not **inf**. The default value is **1.0**.
-cb/--concurrency-backend	Str	No	Concurrency backend, which defaults to **threading-pool**. Supported options: • **threading-pool**: Single-process multi-threaded concurrency backend. • **asyncio**: Asynchronous coroutine concurrency backend. • **processing-pool**: Multi-process concurrency backend. When the concurrency backend is **processing-pool**, **num-process** must be less than or equal to **min(concurrency, init_concurrency)**. If this condition is not met, the tool will automatically set the smaller of **concurrency** or **init_concurrency** to **num-process**.

Parameter	Type	Mandatory	Description
-ub/--use-climb	Bool	No	Specifies whether to enable the ramp-up mode. The default value is **False**, indicating that the ramp-up mode is not enabled.
-gr/--growth-rate	Int	No	Concurrency growth rate for each ramp-up, only effective in ramp-up mode. The default value is **0**.
-gi/--growth-interval	Float	No	Time interval for each ramp-up, only effective in ramp-up mode. The default value is 1,000 ms.
-ic/--init-concurrency	Int	No	Initial concurrency level, only effective in ramp-up mode. The default value is equal to concurrency. This parameter supports specifying multiple initial concurrency levels, separated by ",".
-cm/--climb-mode	String	No	Ramp-up mode, only effective in ramp-up mode. The default value is **linear**. Supported options: • **static**: Concurrency level remains constant, equivalent to stress testing mode. • **linear**: Concurrency level increases linearly over time intervals until it reaches the maximum concurrency level.

Table 4-27 Metrics Options

Parameter	Type	Mandatory	Description
-g, --goodput	String	No	Service SLO, indicating performance metrics that meet business requirements. The unit is milliseconds (ms). Supported metric types: ttft, tpot, and e2el. You can use **-g ttft:50 -g e2e:1000** to specify ttft and e2e metrics.

Parameter	Type	Mandatory	Description
-bi, --bucket-interval	Float	No	Indicates the sampling interval for real-time performance metrics. The unit is ms. If this parameter is specified, it can dynamically monitor the changes in performance metrics within the **bucket_interval** ms.

Table 4-28 Serving Options

Parameter	Type	Mandatory	Description
-b, --backend	String	No	Type of request service API, which can be **openai**, **openai-chat**, **embedding**, or **rerank**. The default value is **openai-chat**.
-p, --provider	String	Yes	Path to the **provider** file, which needs to be created and specified by you.
-pid, --provider-id	String	No	Specifies the provider ID to be tested, useful when the **provider** file contains multiple configurations and only a specific ID needs to be run.
-ol/--output-length	Int	Yes	Length of output tokens. You can specify multiple output lengths, separated by ",".
-ra/--range-ratio-above	Float	No	The ratio by which the output token length can dynamically increase. The maximum length is output_length x (1 + **range_ratio_above**). The value range is [0, 1]. The default is **0.0**.
-rb/--range-ratio-below	Bool	No	The ratio by which the output token length can dynamically decrease. The minimum length is output_length x (1 – **range_ratio_above**). The value range is [0, 1]. The default is **0.0**.
-w/--warmup	Int	No	Number of warmup requests. The default value is **0**, indicating that warmup is not enabled.

Parameter	Type	Mandatory	Description
-e/--epochs	Int	No	Number of times to run the same concurrency configuration. The default value is **1**, indicating that each concurrency group runs only once.
-tk/--top-k	Int	No	Top-k sampling parameter, only effective for OpenAI-compatible backends. The default value is **-1**.
-tp/--top-p	Float	No	Top-p sampling parameter, only effective for OpenAI-compatible backends. The default value is **1.0**.
-mp/--min-p	Float	No	Min-p sampling parameter, indicating the minimum probability for a token to be considered. Must be in the range [0, 1], only effective for OpenAI-compatible backends.
-temper/--temperature	Float	No	Temperature sampling parameter. The default value is **0**.
-cs/--chunk-size	Int	No	Chunk size in stream requests. The default value is **1024**.
-ef/--encoding-format	String	No	Encoding format for the backend's response. The value can be **float** or **base64**. The default value is **float**.
-usd/--use-spec-decode	Bool	No	Indicates whether speculative inference is enabled on the server. If enabled, it can be combined with the **--num-spec-tokens** parameter to calculate the MTP acceptance rate. The default value is **False**, indicating that speculative inference is disabled.
-nst/--num-spec-tokens	Int	No	Number of speculative inference tokens configured on the server. A value of **1** indicates that the server will infer one additional token each time, which can be combined with the **--use-spec-decode** parameter to calculate the MTP acceptance rate. The default value is **-1**.

Parameter	Type	Mandatory	Description
-umar/--use-mtp-accept-rate	Bool	No	Indicates whether to ignore the number of tokens generated by the model when calculating the MTP acceptance rate. The default value is **True**, indicating that the number of tokens generated by the model is ignored.
-nss/--num-scheduler-steps	Int	No	Size of **multi step** on the server, used to calculate the MTP acceptance rate. The default value is **1**.
-timeout/--timeout	Float	No	Request timeout time. The default value is 1,000s.
-ie/--ignore-eos	Bool	No	Specifies whether to ignore EOS. The default value is **True**, indicating that EOS is ignored.
-cus/--continuous-usage-stats	Bool	No	Specifies whether to include usage information in each returned chunk in stream requests. The default value is **True**, indicating that usage information is included in each returned chunk.
-sst/--skip-special-tokens	Bool	No	Specifies whether to skip special tokens. The default value is **False**.
-er/--enable-max-tokens-exclude-reasoning	Bool	No	Specifies whether to enable max-tokens excluding reasoning, which will proactively disconnect from the server when **max-tokens** is reached. The default value is **True**.
-pf/ --profile	Bool	No	Specifies whether to collect Service Profiler information from the server. The default value is False, indicating that the information is not collected. The warmup phase does not collect server service information.
-pl/--profile-level	String	No	Collection level for the server's Service Profiler. The options include **Level_none**, **Level0**, **Level1**, and **Level2**. This parameter is only effective when profiling is enabled; its default value is **none**.

Parameter	Type	Mandatory	Description
-trace/--trace	Bool	No	Specifies whether to enable the tool's trace switch, which will monitor and display the concurrency process. The default value is **False**, indicating that it is not enabled.
-s/--benchmark-save-path	String	No	Path to the folder where performance metrics are saved. The default value is **./benchmark_output**.

Artifact Description

After the script runs, a **requests** directory and a **summary** CSV file will be created in the output path specified by the **--benchmark-save-path** parameter. Inside the **requests** directory, a CSV file starting with **requests** will be output.

1.requests_{*provider*}_{*dataset_type*}_{*control_method*}_concurrency{*concurrency*}_{*concurrency_backend*}_input{*input_length*}_output{*output_length*}_{*current_time*}.csv

2.summary_{*provider*}_{*control_method*}_{*concurrency_backend*}_{*current_time*}.csv

An example is shown in the figure below.

Figure 4-7 Request details

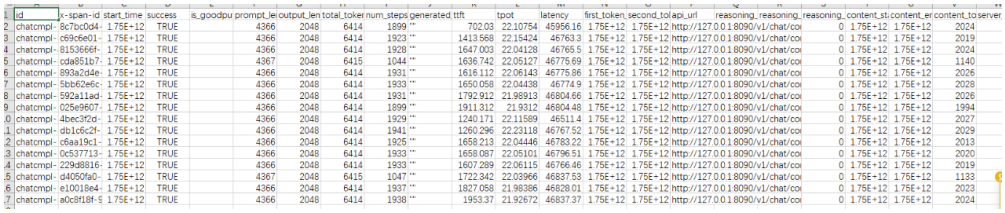
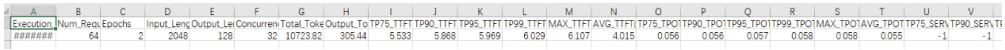


Figure 4-8 CSV file for performance metrics



4.1.8.2 Multimodal Model Inference Performance Test

Currently, performance tests of multimodal model inference supports the "language + image" and "language + video" modes. The acs-bench tool is used to implement these tests. For details about how to install the acs-bench tool, see **Installing the acs-bench Tool**.

Constraints

The current version supports "language + image" and "language + video" multimodal performance tests.

Obtaining Datasets

1. Generate an offline dataset containing images and text. For details about the parameters, see **Dataset Generation Parameter Description**.

```
# Generate a random dataset (image and text). Assume that the token length is 100 and the image length and width are (250, 250).
acs-bench generate dataset \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--output-path ./ \
--num-requests 10 \
--input-length 100 \
--modal-type "image-text" \
--config-option image_height:250,image_width:250
```

   The generated JSON dataset file **${input-length}_image_${image_height}_${image_width}.json** is stored in the **--output-path** directory.

2. Generate an offline dataset containing videos and text. For details about the parameters, see **Dataset Generation Parameter Description**.

```
# Generate a random dataset (video and text). Assume that the token length is 100, the video frame length and width are (250, 250), and the video duration and number of frames are (3, 25).
acs-bench generate dataset \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--output-path ./ \
--num-requests 10 \
--input-length 100 \
--modal-type "video-text" \
--config-option image_height:250,image_width:250,duration:3,fsp:25
```

   The generated JSON dataset file **${input-length}_video_${image_height}_${image_width}_${duration}_${fps}.json** is stored in the **--output-path** directory.

NOTICE

The **--input-length** and **--num-requests** parameters in the dataset generation command only support single values.

If you need to generate datasets with different specifications, modify the **--input-length** or **--num-requests** parameters to the desired values and then execute the command.

Performance Stress Testing Mode Verification

An example of using the **acs-bench prof** command for multimodal model performance stress testing is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Take "image + text" as an example: Use a thread pool for concurrent testing. The default backend concurrency mode is threading-pool. You can also choose the asynchronous coroutine concurrency mode asyncio, the multi-process mode processing-pool, or the multi-thread mode threading-pool.
$ acs-bench prof \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--provider ./provider/providers.yaml \
--input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,2048 --output-length 128,2048 \
--modal-type "image-text" \
```



```
--config-option image_height:250,image_width:250 \
--benchmark-save-path ./output_path/

# Take "video + text" as an example: Use a thread pool for concurrent testing. The default backend concurrency mode is threading-pool. You can also choose the asynchronous coroutine concurrency mode asyncio, the multi-process mode processing-pool, or the multi-thread mode threading-pool.
$ acs-bench prof \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--provider ./provider/providers.yaml \
--input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,2048 --output-length 128,2048 \
--modal-type "video-text" \
--config-option image_height:250,image_width:250,duration:3,fsp:25 \
--benchmark-save-path ./output_path/
```

Ramp-Up Mode Verification

An example of using the **acs-bench prof** command for multimodal model ramp-up stress testing is shown below. For details about parameter descriptions, see **Parameter Descriptions for Usage Example**. For details about output artifact descriptions, see **Artifact Description**.

```
# Take "image + text" as an example: Use a thread pool for ramp-up testing. The default backend concurrency mode is threading-pool. You can also choose the asynchronous coroutine concurrency mode asyncio, the multi-process mode processing-pool, or the multi-thread mode threading-pool.
$ acs-bench prof \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--provider ./provider/providers.yaml \
--input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--use-climb --climb-mode linear --growth-rate 2 --init-concurrency 1 --growth-interval 5000 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,2048 --output-length 128,2048 \
--modal-type "image-text" \
--config-option image_height:250,image_width:250 \
--benchmark-save-path ./output_path/

# Take "video + text" as an example: Use a thread pool for ramp-up testing. The default backend concurrency mode is threading-pool. You can also choose the asynchronous coroutine concurrency mode asyncio, the multi-process mode processing-pool, or the multi-thread mode threading-pool.
$ acs-bench prof \
--tokenizer ./model/Qwen-2.5-VL-72B/ \
--provider ./provider/providers.yaml \
--input-path ./built_in_dataset/ \
--concurrency-backend threading-pool \
--backend openai-chat --warmup 1 \
--epochs 2 \
--use-climb --climb-mode linear --growth-rate 2 --init-concurrency 1 --growth-interval 5000 \
--num-requests 1,2,4,8 --concurrency 1,2,4,8 \
--input-length 128,2048 --output-length 128,2048 \
--modal-type "video-text" \
--config-option image_height:250,image_width:250,duration:3,fsp:25 \
--benchmark-save-path ./output_path/
```


NOTICE

The **--input-length** parameter values in the performance stress testing and ramp-up testing examples must exist in the pre-generated dataset. If they do not exist, refer to **Obtaining Datasets** to generate a dataset with the corresponding input length.

4.1.8.3 Obtaining Model Inference Profiling Data

PyTorch Profiler is a performance analysis tool provided by PyTorch, used to deeply analyze performance bottlenecks during the model training/inference process, helping developers optimize computational efficiency, memory usage, and hardware utilization.

Ascend PyTorch Profiler is fully aligned with the usage in PyTorch-GPU scenarios, supporting the collection of PyTorch layer operator information, CANN layer operator information, underlying NPU operator information, and operator memory usage information, enabling a comprehensive analysis of the performance status of PyTorch AI tasks.

However, using PyTorch Profiler can result in large data volumes, longer data collection times, and performance overhead that may lead to inaccurate data and distorted results. To address these issues, a lightweight performance analysis tool called Service Profiler has been introduced, which is used to analyze performance issues at the service request level. Service Profiler currently gathers profiling data of interest to users by pre-instrumenting key points within the service framework. The current capabilities supported include observing the batch size of internal service requests, sequence length, and the execution time of a single batch iteration.

Constraints

Before using Service Profiler, ensure that the inference service can be started and handle requests normally. The Service Profiler is now included in the versioned image as a Python library.

Checking if the Service Profiler Tool is Installed

In ModelArts 6.5.906 and later, the **acs_service_profiler-1.0.1-py3-none-any.whl** package is installed by default, so there is no need for a separate installation. The package is located in the **llm_tools** directory within the **AscendCloud-LLM-xxx.zip** software package.

Check if the acs-service-profiler tool is already installed:

```
$ pip show acs-service-profiler
```

If it is not installed, refer to **Installing the acs-bench Tool** for instructions on installing the acs-bench tool. The installation command is as follows:

```
$ pip install llm_tools/acs_service_profiler-*-py3-none-any.whl
```

Note: Both Ascend PyTorch Profiler and Service Profiler are features enabled during the performance tuning phase of development and are not recommended for use in production service states. Generally, using Ascend PyTorch Profiler

involves collecting a small amount of request data (one or two requests) for analysis, while Service Profiler collects data over a period of requests (hundreds or thousands) for analysis. The following section explains how to collect data using Ascend PyTorch Profiler and Service Profiler in a real-time service scenario.

Real-Time Service Profiling Through start_profile and stop_profile

1. Before starting the inference service, set the environment variables:
   ```
   export VLLM_TORCH_PROFILER_DIR=/home/ma-user/profiler_dir # Enable Ascend PyTorch Profiler
   # export VLLM_SERVICE_PROFILER_DIR =/home/ma-user/profiler_dir # Enable Service Profiler
   ```
   VLLM_TORCH_PROFILER_DIR/VLLM_SERVICE_PROFILER_DIR is used to enable the Ascend PyTorch Profiler or Service Profiler. The collected profiler data is stored in the path specified by the environment variable. Note that both cannot be enabled simultaneously.
2. After setting the environment variables, start the inference service.

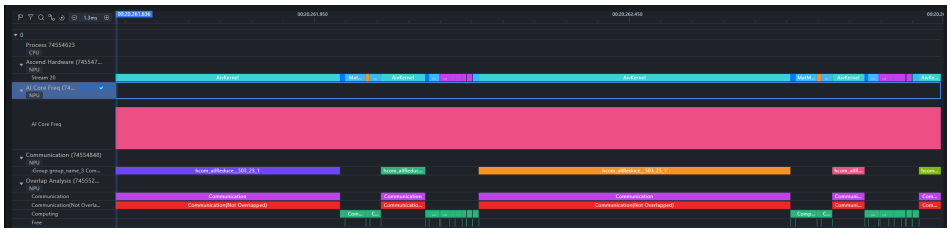
   For details about how to start the inference service, see **Starting an LLM-powered Inference Service**.
3. Send a start_profile POST request.
   ```
   curl -X POST http://${IP}:${PORT}/start_profile
   ```
   Parameters

   a. **IP**: The IP address where the service is deployed.

   b. **PORT**: The port where the service is deployed.
4. Send an actual request.

   For sending actual requests, see **LLM Inference Performance Test**.
5. Send a stop_profile POST request.
   ```
   curl -X POST http://${IP}:${PORT}/stop_profile
   ```
   The parameters are same as the start_profile POST request.
6. Perform post-processing and visualization.

   For visualizing data collected by Ascend PyTorch Profiler, it is recommended to use the MindStudio Insight tool. The visualization effect is shown in the following figure.

   For more information on MindStudio Insight, see the **MindStudio Insight tool documentation**.

   To visualize data collected by Service Profiler, use the acsprof tool for post-processing and then visualize the data in a web page that supports the Google tracing format. The specific steps are as follows:

   Post-Processing to Generate Visualization Files
   ```
   acsprof export -i ${input_path}
   ```
   The following table describes the parameters.

Parameter	Type	Description	Mandatory
-i / --input_path	String	Specifies the path to the Service Profiler collection folder, supporting both parent and subfolders.	Yes
-o / --output_path	String	Specifies the output path for the post-processed files, defaulting to the input folder path.	No
-f / --force_reparse	Bool	Specifies whether to perform forced re-parsing for already parsed folders. The default value is **False** (no forced re-parsing). In scenarios where multiple batches of data are collected, the first batch will be parsed automatically, and subsequent batches will not be parsed automatically. Set this to **True** to enable forced re-parsing.	No

Example:

```
acsprof export -i /home/ma-user/profiler_dir
```

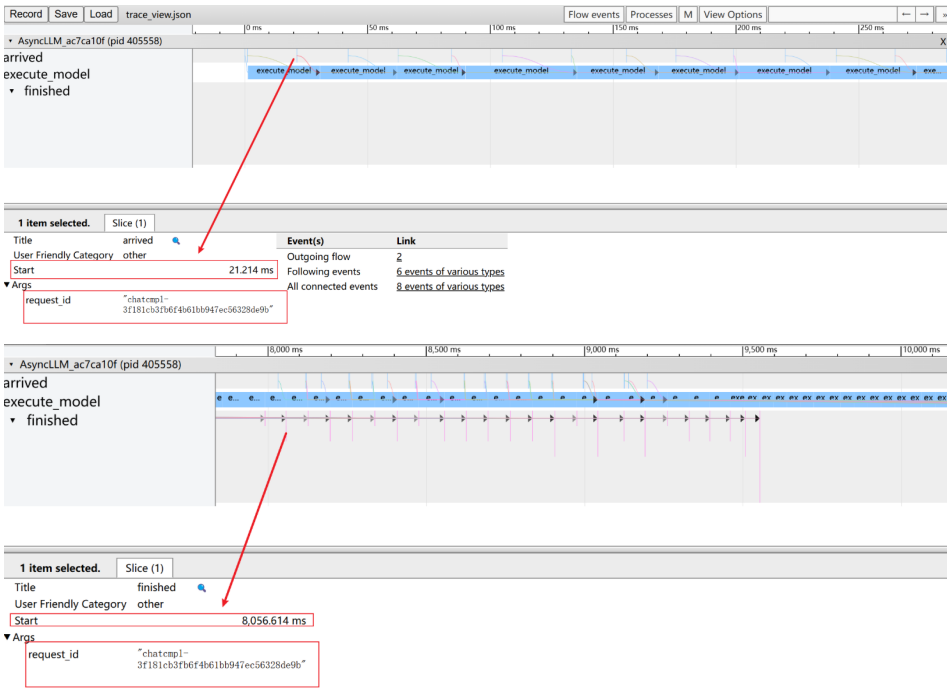
Normal log output

```
(PyTorch-2.5.1) [root@devserver-bms-        3 benchmark]# acsprof export -i /nfs/gyn/scripts/service_profiler/
============ Request Summary Board ==============
Successful requests:                    82
Duration (s):                           17.53
Total input tokens:                     44655
Total generated tokens:                 10496
Request throughput (req/s):             4.68
Output token throughput (tokens/s):     598.61
Total token throughput (tokens/s):      3145.39
---------------Time to First Token----------------
Mean TTFT (ms):                         80.64
Median TTFT (ms):                       79.37
P90 TTFT (ms):                          115.51
P99 TTFT (ms):                          148.69
-----Time per Output Token (excl. 1st token)------
Mean TPOT (ms):                         56.63
Median TPOT (ms):                       63.48
P90 TPOT (ms):                          66.40
P99 TPOT (ms):                          66.64
----------------Request Latency-----------------
Mean Latency (ms):                      7272.99
Median Latency (ms):                    8116.33
P90 Latency (ms):                       8542.51
P99 Latency (ms):                       8573.33
================================================
```

Post-processing parses the profiler data once more. It exports metrics like TTFT, TPOT, and framework throughput. It also creates a visualization file named **trace_view.json**. For multiple instances, the tool combines their timeline data into an **overview_trace_view.json** file. You can drag these files into **chrome://tracing/** or **https://ui.perfetto.dev/** for visual analysis.

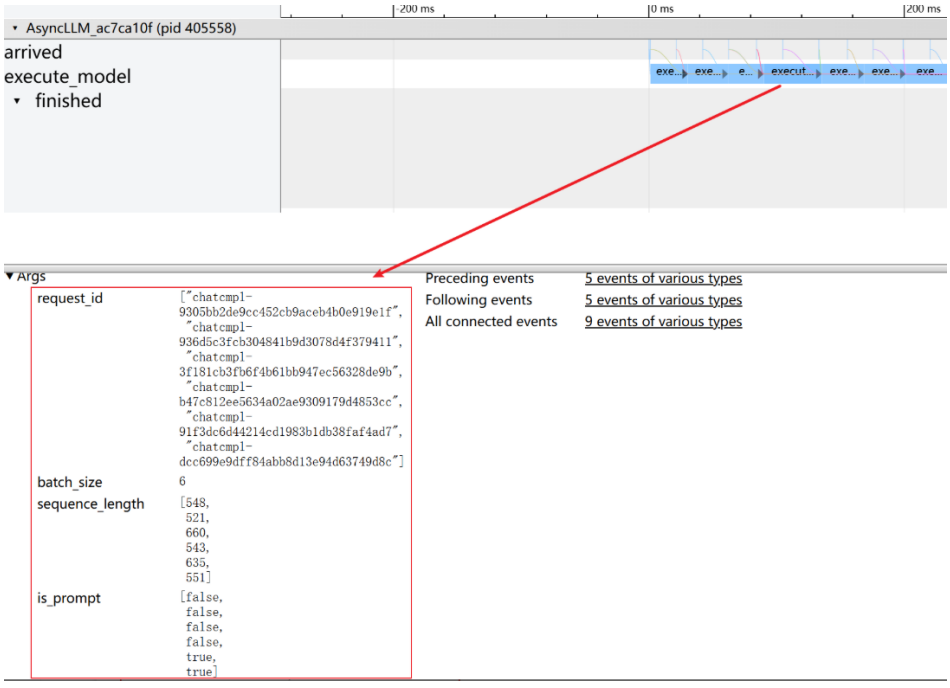
Result visualization

a. Request arrival or end time



b. Service process group batch details

The details include the execution time for a single group batch process, request ID, batch size, sequence length, and statistics on whether the request is in the prompt or decode phase.



4.1.9 Appendix

4.1.9.1 Ascend-vLLM Inference FAQs

Issue 1: NPU Out of Memory Occurred During Inference and Prediction

Solution: Adjust the GPU memory utilization when starting the inference service by reducing the value of **--gpu-memory-utilization**.

Issue 2: ValueError:User-specified max_model_len is greater than the drived max_model_len Occurred During Inference and Prediction

Solution:

```
export VLLM_ALLOW_LONG_MAX_MODEL_LEN=1
```

This allows passing a value greater than the maximum sequence length specified in the model's **config.json**.

Issue 3: Poor Performance or Accuracy Issues When Using Offline Inference

Solution: Set the **block_size** to **128**.

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


5 LLM Training

5.1 Adapting Mainstream Open-Source Models to AscendFactory NPU Training Based on Lite Server

5.1.1 Solution Overview

Description

ModelArts Lite Servers allow you to train popular open-source foundation models with the AscendFactory framework using Snt9b and Snt9b23 compute resources.

AscendFactory enables one-click training by integrating frameworks like MindSpeed-LLM (previously ModelLink), Llama-Factory, VeRL, MindSpeed-RL, and MindSpeed-MM.

Table 5-1 AscendFactory adaptation and training stages and strategies

Training Framework	Pre-Training	Reinforcement Learning	Supervised Fine-Tuning	
		GRPO	Full	LoRA
Llama-Factory	√	x	√	√
MindSpeed-LLM	√	x	√	√
VeRL	x	√	x	x
MindSpeed-RL	x	√	x	x
MindSpeed-MM	√	x	√	x

Solution Architecture

This architecture shows how to train and deploy open-source third-party foundation models.

- It outlines a solution for using these models in Lite Server scenarios, covering training, tuning, and O&M.
- Lite Server and SFS Turbo serve as the deployment infrastructure. For public network access, bind an EIP to Lite Server resources.
- Before using a third-party foundation model, create an image package using AscendFactory and its base image. This package includes the required training framework and dependencies.
- Store trained model weights and process files in SFS Turbo shared file systems, mounted across all nodes. Combined with resumable training, this ensures reliability. (Optional) If Snt9b23 lacks local disks, use OBS or EVS disks for log and file storage.
- (Optional) Consider using Cloud Eye for monitoring and configuring alarms.

Usage Process

Use this solution to deploy open-source third-party foundation models by following these steps:

1. Resource planning: After choosing a deployment solution using the overall architecture as a guide, pick the necessary compute, storage, and access-layer dependency resources according to the **supported models**, **features**, and **resource planning**.
2. **Training preparation**: After buying compute and storage resources on Huawei Cloud, download a base image and software package to build your image. Then, prepare the required open-source weights and datasets for the model.
3. **Training execution**: Adjust the suggested settings for each model using AI Compute Service and run the training jobs.
4. (Optional) **Log collection**: After finishing these steps, your basic training job is complete. For better maintenance, review the main logs created during training to identify important issues easily.
5. (Optional) **Monitoring and alarming**: The system includes built-in monitoring features. You will understand key metrics and set up alarms easily.
6. (Optional) **Configuration optimization**: To improve training efficiency, consider using suggested optimization techniques tailored to your model's parameters and data size.
7. (Optional) **Resumable training**: If the training is interrupted due to unknown reasons, you can configure resumable training and restart the training job.

5.1.2 Supported Models

Models are classified into **large language models (LLMs)** and **multimodal models**. The details are as follows.

Table 5-2 Supported LLMs and their weight download addresses

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
DeepSeek	DeepSeek-R1-671B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/deepseek-ai/DeepSeek-R1/tree/main
	DeepSeek-V3-671B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/deepseek-ai/DeepSeek-V3-Base/tree/main
	DeepSeek-V2-Lite 16B	Pre-training and full-parameter fine-tuning	MindSpeed-LLM	6.5.906 or later	https://huggingface.co/deepseek-ai/DeepSeek-V2-Lite
Qwen2	Qwen2-0.5B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
	Qwen2-1.5B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
	Qwen2-7B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/Qwen/Qwen2-7B-Instruct

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
	Qwen2-72B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2-72B-Instruct**
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
Qwen2.5	Qwen2.5-0.5B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2.5-0.5B-Instruct**
		Pre-training and fine-tuning	LLaMA-Factory		
	Qwen2.5-1.5B	Reinforcement learning	MindSpeed-RL	6.5.906 or later	**https://huggingface.co/Qwen/Qwen2.5-1.5B**
	Qwen2.5-7B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2.5-7B**
		Pre-training and fine-tuning	LLaMA-Factory		

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
		Reinforcement learning	MindSpeed-RL	6.5.906 or later	
	Qwen2.5-14B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/Qwen/Qwen2.5-14B-Instruct
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
		Reinforcement learning	LLaMA-Factory	6.5.907 or later	
	Qwen2.5-32B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/Qwen/Qwen2.5-32B
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
		Reinforcement learning	MindSpeed-RL	6.5.906 or later	
		Reinforcement learning	VeRL	6.5.907 or later	

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
	Qwen2.5-72B	Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	https://huggingface.co/Qwen/Qwen2.5-72B-Instruct
		Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	
		Reinforcement learning	LLaMA-Factory	6.5.907 or later	
Qwen3	Qwen3-0.6B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	https://huggingface.co/Qwen/Qwen3-0.6B
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-1.7B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	https://huggingface.co/Qwen/Qwen3-1.7B
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-4B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	https://huggingface.co/Qwen/Qwen3-4B

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
		Reinforcement learning	VeRL	6.5.907 or later	
	Qwen3-8B	Reinforcement learning	VeRL	6.5.906 or later	**https://huggingface.co/Qwen/Qwen3-8B**
		Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-14B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	**https://huggingface.co/Qwen/Qwen3-14B**
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-32B	Reinforcement learning	VeRL	6.5.906 or later	**https://huggingface.co/Qwen/Qwen3-32B**

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
		Pre-training and fine-tuning	MindSpeed-LLM	6.5.905 or later	
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-30B-A3B	Pre-training and full-parameter fine-tuning	MindSpeed-LLM	6.5.905 or later	https://huggingface.co/Qwen/Qwen3-30B-A3B
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	
	Qwen3-235b-A22B	Pre-training and full-parameter fine-tuning	MindSpeed-LLM	6.5.905 or later	https://huggingface.co/Qwen/Qwen3-235B-A22B
		Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
Llama	Llama3.1-8B/70B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	https://huggingface.co/meta-llama/Meta-Llama-3.1-70B-Instruct
	Llama3.2-1B/3B	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct
GLM	glm-4-9b-chat	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/THUDM/glm-4-9b-chat
		Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	
Mistral AI	Mixtral-8x7B-Instruct-v0.1	Pre-training and fine-tuning	MindSpeed-LLM	6.5.902 or later	https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Table 5-3 Supported multimodal models and their weight download addresses

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
Qwen 2 VL	Qwen2-VL-2B	Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct/tree/main**
	Qwen2-VL-7B	Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct/tree/main**
	Qwen2-VL-72B	Pre-training and fine-tuning	LLaMA-Factory	6.5.902 or later	**https://huggingface.co/Qwen/Qwen2-VL-72B-Instruct**
Qwen 2.5 VL	Qwen2.5-VL-3B	Reinforcement learning	VeRL	6.5.906 or later	**https://huggingface.co/Qwen/Qwen2.5-VL-3B-Instruct**
		Pre-training and fine-tuning	MindSpeed-MM	6.5.907 or later	
		Pre-training and fine-tuning	LLaMA-Factory	6.5.907 or later	
	Qwen2.5-VL-7B	Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	**https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct**
		Pre-training and fine-tuning	MindSpeed-MM	6.5.907 or later	

Series	Model	Training Scenario	Training Framework	Version	Open-Source Weight File Download Address
		Reinforcement learning	VeRL	6.5.906 or later	
	Qwen2.5-VL-32B	Pre-training and fine-tuning	LLaMA-Factory	6.5.906 or later	**https://huggingface.co/Qwen/Qwen2.5-VL-32B-Instruct**
		Reinforcement learning	VeRL	6.5.905 or later	
	Qwen2.5-VL-72B	Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	**https://huggingface.co/Qwen/Qwen2.5-VL-72B-Instruct**
		Reinforcement learning	VeRL	6.5.906 or later	
Gemma	Gemma3-27b	Pre-training and fine-tuning	LLaMA-Factory	6.5.905 or later	**https://huggingface.co/google/gemma-3-27b-it**

5.1.3 Training Features Supported by Each Model

The AscendFactory solution supports various training features for each model, as outlined in this section.

Type	Series	Model	Pre-Training and Fine-Tuning			Reinforcement Learning	
			MindSpeed-LLM	LlamaFactory	MindSpeed-MM	VeRL	MindSpeed-RL

			Pre-training and full-parameter fine-tuning	LoRA fine-tuning	Multi-sample pack	Flash	SPTD parallelism	Long sequence parallelism	Mixture of Expert (MoE) parallelism	Dynamic sentence length	Training methods	ZeRO parallelism	Flash	Pre-training and full-parameter fine-tuning	SPTD parallelism	Distributed optimizer	Recomputation	Training methods	sglang	vllm	Training backend	Training methods	vllm	Training backend Megatron	Long sequence parallelism

					Fine-tuning	Attention	(SP, PP, TP, DP)	(RingAttention, Ulysses, hybrid long sequence)	(Expert parallelism and communication rearrangement optimization		(PT: pre-training)	(ZeRO-1, ZeRO-2, and ZeRO-3)	Attention		(SP, PP, TP, DP)				Version	Version			Version		

									ization)																
LLM	DeepSeek	DeepSeek-R1-671B	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		DeepSeek-V3-671B	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□

		DeepSeek-V2-Lite16B	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
	Qwen2	Qwen2-0.5B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen2-1.5B	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen2-7B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□

		Qwen2-72B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
	Qwen2.5	Qwen2.5-0.5B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen2.5-1.5B	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	GRPO	0.9.1	□	□
		Qwen2.5-7B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	GRPO	0.9.1	□	□

		Qwen2.5-14B	□	□	□	□	□	□	□	□	PT,SFT,DPO	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen2.5-32B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	GRPO,DAPO,PPO	□	0.9.1	FSDP	GRPO	0.9.1	□	□
		Qwen2.5-72B	□	□	□	□	□	□	□	□	PT,SFT,DPO	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
	Qwen3	Qwen3-0.6B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□

		Qwen3-1.7B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen3-4B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen3-8B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	GRPO	□	0.9.1	FSDP	□	□	□	□
		Qwen3-14B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	GRPO,DAPO,PPO	□	0.9.1	FSDP	□	□	□	□

		Qwen3-32B	□	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	GRPO,DAPO,PPO	□	0.9.1	FSDP	□	□	□	□
		Qwen3-30B-A3B	□	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Qwen3-235b-A22B	□	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□

	Llama	Llama3.1-8B/70B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
		Llama3.2-1B/3B	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
	GLM	glm-4-9b-chat	□	□	□	□	□	□	□	□	PT,SFT	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□

	Mixtral	Mixtral-8x7B-Instruct-v0.1	□	□	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A	□	□	□	□	□	□	□	□
Multimodalmodel	Qwen2VL	Qwen2-VL-2B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A
		Qwen2-VL-7B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A

		Qwen2-VL-72B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A
	Qwen2.5VL	Qwen2.5-VL-3B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	GRPO	□	0.9.1	FSDP	N/A	N/A	N/A	N/A
		Qwen2.5-VL-7B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT,DPO	□	□	□	□	□	□	GRPO,DAPO,PPO	□	0.9.1	FSDP	N/A	N/A	N/A	N/A

		Qwen2.5-VL-32B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	GRPO,DAPO,PPO	□	0.9.1	FSDP	N/A	N/A	N/A	N/A
		Qwen2.5-VL-72B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	GRPO	□	0.9.1	FSDP	N/A	N/A	N/A	N/A
	Gemma	Gemma3-27b	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	PT,SFT	□	□	□	□	□	□	□	□	□	□	N/A	N/A	N/A	N/A

NOTE

- "N/A" means the model does not work with the framework. Multimodal models, for example, do not support the MindSpeed-LLM training framework.

5.1.4 Minimum Number of PUs and Sequence Length Supported by Each Model

Model Training Time and Cluster Scale Prediction

Training time and the number of PUs depend on the model, cluster specifications (Snt9b B3/B2/B1 or Snt9b23), and dataset size. To estimate PU count or training time, use these formulas:

- Training time (second) = Total tokens/(TPS x Number of PUs). This estimate provides a rough range and is for reference only.
- Number of training PUs = Total tokens/(Time x TPS). If this value exceeds eight, increase it to the next multiple of eight and make sure it is no less than the **model's minimum PU requirement**.

Parameters:

1. Total tokens: Depend on several factors like dataset size, epochs, sequence length, and the model used. Preprocessing steps like tokenization and padding can add unnecessary tokens.
   - Total tokens (calculated based on training steps) = Sequence length x Total number of dataset samples. Sequence length can be either dynamic or fixed. Typically, dynamic sequences result in fewer tokens than this calculation suggests. For details about how to set parameters, see **Table 5-12**. By default, MindSpeed-LLM and LlaMA-Factory use fixed sequence length.
2. TPS: Check the benchmark table for each model's throughput (token/s/p) and the number of training PUs used. The table shows baseline measurements taken with a fixed sequence length. For access to the benchmark table, contact Huawei engineers.

Minimum Number of PUs for Model Training

The table below describes the recommended training parameters and compute specifications of different models. Currently, only the number of PUs in the supervised fine-tuning and pre-training phases is provided. An Snt9b flavor typically has eight PUs per node, while an Snt9b23 flavor also uses eight PUs per node but equals 16 DIEs. One DIE matches one PU in Snt9b. For training with Snt9b23, the smallest unit is 2 DIEs when setting the parallel strategy. The configurations below are for reference only. If a setup contains fewer than eight PUs, default to using eight PUs for training. Adjust the PU count as needed.

In the table, **-** indicates that the specification is not supported. **4 x Snt** indicates four PUs in Snt9b and 4 DIEs in Snt9b23, respectively.

Table 5-4 Minimum number of PUs for model training

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
llama3.1-8b	Full-parameter	4,096/8,192	4 x Snt		8 x Snt		-	-	-	-	-	-
	LoRA		4 x Snt		1 x Snt	2 x Snt	-	-	-	-	-	-
llama3.1-70b	Full-parameter	4,096	32 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
	Full-parameter	8,192	64 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
llama3.2-1b	Full-parameter/LoRA	4,096/8,192	1 x Snt	2 x Snt	1 x Snt	1 x Snt	-	-	-	-	-	-
llama3.2-3b	Full-parameter	4,096/8,192	2 x Snt		4 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
	LoRA		1 x Snt	2 x Snt	1 x Snt	2 x Snt	-	-	-	-	-	-
qwen2-0.5b	Full-parameter/LoRA	4,096/8,192	1 x Snt	2 x Snt	1 x Snt	2 x Snt	-	-	-	-	-	-
qwen2-1.5b	Full-parameter/LoRA	4,096/8,192	1 x Snt	2 x Snt	-		-	-	-	-	-	-
qwen2-7b	Full-parameter	4,096	4 x Snt		1 x Snt	2 x Snt	-	-	-	-	-	-
	LoRA		4 x Snt		8 x Snt		-	-	-	-	-	-
	Full-parameter	8,192	8 x Snt		1 x Snt	2 x Snt	-	-	-	-	-	-
	LoRA		8 x Snt		8 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
qwen2-72b	Full-parameter	4,096	32 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
	Full-parameter	8,192	64 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
qwen2.5-0.5b	Full-parameter/LoRA	4,096/8,192	1 x Snt	2 x Snt	1 x Snt	2 x Snt	-	-	-	-	-	-
qwen2.5-1.5b	Full-parameter/LoRA	4,096/8,192	1 x Snt	2 x Snt	-		-	-	8 x Snt		-	-
qwen2.5-7b	Full-parameter	4,096	4 x Snt		8 x Snt		8 x Snt	8 x Snt	8 x Snt	8 x Snt	-	-
	LoRA		2 x Snt		1 x Snt	2 x Snt					-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
	Full-parameter	8,192	8 x Snt		8 x Snt						-	-
	LoRA		2 x Snt		1 x Snt	2 x Snt					-	-
qwen2.5-14b	Full-parameter	4,096	8 x Snt		8 x Snt		8 x Snt	8 x Snt	-	-	-	-
	LoRA		4 x Snt		4 x Snt				-	-	-	-
	Full-parameter	8,192	8 x Snt		16 x Snt				-	-	-	-
	LoRA		8 x Snt		4 x Snt				-	-	-	-
qwen2.5-32b	Full-parameter	4,096	16 x Snt		32 x Snt		16 x Snt	16 x Snt	16 x Snt	16 x Snt	-	-
	LoRA		16 x Snt		8 x Snt						-	-
	Full-parameter	8192	16 x Snt		32 x Snt						-	-
	LoRA		16 x Snt		16 x Snt						-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
qwen2.5-72b	Full-parameter	4,096	32 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
	Full-parameter	8,192	64 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA		16 x Snt		32 x Snt		-	-	-	-	-	-
qwen2vl-2b	Full-parameter	4,096/8,192	-		2 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		1 x Snt		-	-	-	-	-	-
qwen2vl-7b	Full-parameter	4,096/8,192	-		8 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		1 x Snt	2 x Snt	-	-	-	-	-	-
qwen2vl-72b	Full-parameter	1,024	-		32 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
	LoRA	1,024	-		16 x Snt		-	-	-	-	-	-
qwen2.5_vl-3b	Full-parameter	1,024	-		-		-	-	-	-	8 x Snt	
qwen2.5_vl-7b	Full-parameter	1,024/4,096/8,192	-		8 x Snt		8 x Snt	8 x Snt	-	-	8 x Snt	
	LoRA	4,096	-		1 x Snt	2 x Snt			-	-	-	-
qwen2.5_vl-32b	Full-parameter	4,096	-		32 x Snt		16 x Snt		-	-	-	-
		8,192	-		64 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		16 x Snt		-	-	-	-	-	-
qwen2.5_vl-72b	Full-parameter	4,096/8,192	-		64 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		32 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
qwen3-0.6b	Full-parameter/LoRA	4,096/8,192	8 x Snt		8 x Snt		-	-	-	-	-	-
qwen3-1.7b	Full-parameter/LoRA	4,096/8,192	8 x Snt		8 x Snt		-	-	-	-	-	-
qwen3-4b	Full-parameter/LoRA	4,096/8,192	8 x Snt		8 x Snt		-	-	-	-	-	-
qwen3-8b	Full-parameter/LoRA	4,096/8,192	8 x Snt		8 x Snt		8 x Snt		-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
qwen3-14b	Full-parameter/LoRA	4,096/8,192	8 x Snt		8 x Snt		-	-	-	-	-	-
qwen3-32b	Full-parameter	4,096	16 x Snt		32 x Snt		16 x Snt		-	-	-	-
		8,192	16 x Snt		32 x Snt		-	-	-	-	-	-
	LoRA	4,096	8 x Snt		8 x Snt		-	-	-	-	-	-
		8,192	8 x Snt		16 x Snt		-	-	-	-	-	-
qwen3_moe-30B_A3B	Full-parameter	4,096	16 x Snt		32 x Snt		-	-	-	-	-	-
		8,192	32 x Snt		64 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	16 x Snt		32 x Snt		-	-	-	-	-	-
qwen3_moe-235B_A22B	Full-parameter	4,096	256 x Snt		512 x Snt		-	-	-	-	-	-
	LoRA	4,096	128 x Snt		256 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
glm4-9b	Full-parameter	4,096/8,192	8 x Snt		8 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	2 x Snt		1 x Snt	2 x Snt	-	-	-	-	-	-
mixtral-8x7b	Full-parameter	4,096/8,192	16 x Snt		-		-	-	-	-	-	-
DeepSeek-V3/R1	Full-parameter	4,096	512 x Snt		-		-	-	-	-	-	-
	LoRA		64 x Snt		-		-	-	-	-	-	-
internvl2.5-8b	Full-parameter/LoRA	4,096/8,192	-		8 x Snt		-	-	-	-	-	-
internvl2.5-38b	Full-parameter	4,096/8,192	-		32 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		16 x Snt		-	-	-	-	-	-

Supported Model Parameters	Training Strategy	Sequence Length	MindSpeed-LLM PUs/DIEs		LlaMA-Factory PUs/DIEs		VeRL PUs/DIEs		MindSpeed-RL PUs/DIEs		MindSpeed-MM PUs/DIEs	
			Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23	Snt9b	Snt9b23
internvl2.5-78b	Full-parameter	4,096	-		32 x Snt		-	-	-	-	-	-
		8,192	-		64 x Snt		-	-	-	-	-	-
	LoRA	4,096	-		16 x Snt		-	-	-	-	-	-
		8,192	-		32 x Snt		-	-	-	-	-	-
gemma3-27b	Full-parameter	4,096	-		16 x Snt		-	-	-	-	-	-
		8,192	-		48 x Snt		-	-	-	-	-	-
	LoRA	4,096/8,192	-		16 x Snt		-	-	-	-	-	-

NOTE

- LlaMA-Factory's ZeRO parallelism splits the optimizer, gradients, and weights across multiple PUs. This means the size of the cluster impacts both the best setup and overall performance.
- Enabling distributed optimizer parallelism on MindSpeed-LLM splits and shares optimizer parameters across all nodes in the cluster. The best setup depends on the total number of PUs used.
- The benchmark balances the fewest running PUs with peak performance. Adjust settings according to your cluster size and needs.

5.1.5 Version Software Description and Requirements

Version Differences

Use this guide for AI Compute Service version 6.5.906 or newer. The latest version is 6.5.907. You are advised to use the latest software package and image.

Table 5-5 Version differences

Version	Description
6.5.907	Compared with 6.5.906, 6.5.907 supports the following new features: 1. VeRL reinforcement learning framework: The Qwen2.5VL, Qwen3, Qwen2.5 series models are added and support PPO, DAPO, and GRPO. 2. Llama-Factory framework: The Qwen2.5-14B, Qwen2.5-VL-7B, and Qwen2.5-72B models are added and support DPO reinforcement learning. 3. MindSpeed-MM framework: The Qwen2.5VL-7B and Qwen2.5VL-3B models are added and support pre-training and fine-tuning. Compared with 6.5.906, 6.5.907 does not support the following feature changes: 1. Llama-Factory framework: Internvl2.5-8B, Internvl2.5-38B, and Internvl2.5-78B do not support full-parameter and LoRA fine-tuning.
6.5.906	Compared with 6.5.905, 6.5.906 has the following feature changes: 1. The MindSpeed_RL reinforcement learning framework enables GRPO algorithm training for the Qwen2.5 model series. 2. The VeRL reinforcement learning framework works with both the Qwen3-8B LLM and the Qwen2.5-VL multimodal model series.

Base Image Versions

The following tables list the base image addresses and their versions for this tutorial.

Table 5-6 Base image addresses

Usage	Address	Version
Base Snt9b image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	6.5.907

Usage	Address	Version
Base Snt9b23 image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129	6.5.907

Table 5-7 Base image addresses (dedicated for DeepSeek)

Usage	Address	Version
Base Snt9b image (dedicated for DeepSeek)	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 CN-Hong Kong swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	6.5.907
Base Snt9b23 image (dedicated for DeepSeek)	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_2_1_ascend:pytorch_2.1.0-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129	6.5.907

Table 5-8 Model image versions

Server Model	Model	Version
Snt9b	CANN	8.2.RC1
	Driver	25.2.1

Server Model	Model	Version
	PyTorch	2.5.1
		2.1.0 (for DeepSeek)
Snt9b23	CANN	8.2.RC1
	Driver	25.2.1
	PyTorch	2.5.1
		2.1.0 (for DeepSeek)

Software Package Obtaining

The following table lists the software version and dependency package required by this solution and how to obtain them.

Table 5-9 Software versions and download URL

AI Compute Service Version	Software	Description	How to Obtain
6.5.907	AscendCloud-6.5.907-timestamp.zip	The training code used in this tutorial is included.	1. Go to **Support-E**. 2. Find ModelArts 6.5.907.1. **NOTE** If the software information does not appear when opening the download link, you lack access permissions. Contact your company's Huawei technical support for assistance with downloading.

Software Package Structure

Key training files in the **AscendCloud-LLM** code package:

```
|——AscendCloud-LLM
  |——llm_train                  # Model training code package
    |——AscendFactory
      |——examples/               # Directory of the config file
      |——data.tgz                # Compressed package of sample data
      |——third-party/            # Patch package
      |——src/acs_train_solution/  # Training package
      |——install.sh              # Environment initialization script
      |——Dockerfile
      |——scripts_install         # Installation script packages of each framework
      |——dependences.yaml        # YAML file of the open-source community code version
```


5.1.6 Training Preparations

5.1.6.1 Preparing the Lite Server Environment

Purchasing Lite Server Resources

- **Enable Lite Server resources** and obtain passwords. Verify SSH access to all servers. Confirm proper network connectivity between them.
- For distributed training on multiple servers, purchase mountable storage disks accessible by the servers. Both Scalable File Service (SFS) and Elastic Volume Service (EVS) can be mounted to Lite Servers. The section below describes the SFS solution. For details about the EVS solution, see **Configuring the Storage**.
- Ensure that the container connects to the public network for Git clone's internet access during installation. Mount an EIP to the resources. For details, see **Configuring the Network**.

NOTE

If a container is used or shared by multiple users, you should restrict the container from accessing the OpenStack management address (169.254.169.254) to prevent host machine metadata acquisition. For details, see **Forbidding Containers to Obtain Host Machine Metadata**.

(Optional) Interconnecting with SFS

If you use SFS for storage, SFS Turbo file systems are recommended. SFS Turbo provides high-performance file storage on demand. It features high reliability and availability. It can be elastically expanded and performs better as its capacity grows. The service is suitable for a wide range of scenarios.

1. Create a file system on the SFS console. For details, see **Creating an SFS Turbo File System**. File systems and ECSs in different AZs of the same region can communicate with each other. Therefore, ensure that SFS Turbo and the server are in the same region.
2. Mount the created file system to the server. For details, see **Mounting an NFS File System to ECSs (Linux)**.
3. Set automatic mounting upon restart on the server to prevent mounting loss. For details, see **Mounting a File System Automatically**.

NOTICE

Snt9b23 nodes lack local hard drives. When buying these nodes, acquire EVS disks through the ModelArts console. For storing model weight files, consider using an SFS Turbo file system.

Choose an SFS Turbo file system that offers at least 500 MB/s/TiB. Ensure its capacity is 1.2 TB or a whole number multiple of 1.2 TB.

The formula for estimating the SFS bandwidth is as follows:

Bandwidth (MB/s) ≈ (Weights for storing the optimizer state x 1,024) x Multiplication coefficient/Storage duration

The multiplication coefficient usually falls between 6 and 8. For optimal resource allocation and smooth performance, 8 is recommended.

The calculation example is as follows:

If the weights for saving the optimizer state is 200 GB and the recommended storage duration is 20 minutes, the required bandwidth is:

(200 GB x 1,024 x 8)/1,200s = 1,365 MB/s

5.1.6.2 Preparing Software Packages, Weights, and Training Datasets

Before training, upload the model weights, software packages, and training datasets to the specified directory on the Lite Server.

Table 5-10 Uploading files and directories

File	Recommended Directory	Description
Model weights	**/mnt/sfs_turbo/model/** *{Model name}*	Define **/mnt/sfs_turbo** as the variable *${work_dir}*.
AscendCloud software package (including the AscendCloud-LLM code package. For details, see **Software Package Structure**.)	**/mnt/sfs_turbo**	
Training dataset	**/mnt/sfs_turbo/training_data**	

The **/mnt/sfs_turbo** directory serves as the default working directory for mounting SFS Turbo to the host. Simply upload your model weights, software packages, and training datasets on the server.

Step 1: Uploading the Code Package and Weight File

1. Upload the **AscendCloud-LLM-xxx.zip** package to the host and decompress it. For details about how to obtain the package, see **Table 5-9**. The decompression details are as follows:


```
cd ${work_dir}
unzip AscendCloud-*.zip && unzip ./AscendCloud-LLM-*.zip
```

2. Upload the weight file to the Lite Server. The weight file must be in Hugging Face format. For details about how to obtain the open-source weight file, see **Supported Models**.
3. The weight file must be stored in the specified directory on the disk. Ensure that the model file and weight file (such as the LFS file) have been completely downloaded.

```
cd ${work_dir}
mkdir -p model/{model_name}
```

4. Modify the weight (tokenizer) file for these models based on the selected framework and model type. For details, see the **tokenizer file description**.
   - LlaMA-Factory: glm4-9b model and InternVL2_5 models

Step 2: Uploading Data to a Specified Directory

The data requirements vary according to the framework. Place the expected training data in the *${work_dir}*/**training_data** directory. The procedure is as follows:

1. Create the **training_data** directory.

```
cd ${work_dir}
mkdir training_data
```

2. Obtain the datasets and upload the specified data to the *${work_dir}*/**training_data** directory.
   - Method 1: Download data by referring to **Training Data Description**.
   - Method 2: Some datasets have been preset in the software package and can be used directly.

```
tar -zxvf ${work_dir}/llm_train/AscendFactory/data.tgz
cp  ${work_dir}/llm_train/AscendFactory/data/*  ${work_dir}/training_data
```

   - Method 3: You have processed the datasets.
3. Place the raw data or processed data by following this structure.

```
${work_dir}
 |── training_data
     |── alpaca_en_demo.json                # Original code dataset
     |── identity.json                    # Original code dataset
     ...dat
     |── alpaca_gpt4_data.json              # Custom sample dataset
```

4. [LlaMA-Factory framework] Preprocess data.
   - If data needs to be preprocessed, update the **data/dataset_info.json** file in the code directory. The command below is for reference. For details about the dataset file format and configuration, see **README_zh.md**.

```
vim dataset_info.json
```

     The new parameters are as follows:

```
"alpaca_gpt4_data": {
   "file_name": "alpaca_gpt4_data.json"
 },
```

     The following figure shows an example.


```
{
  "identity": {
    "file_name": "identity.json"
  },
  "alpaca_en_demo": {
    "file_name": "alpaca_en_demo.json"
  },
  "alpaca_zh_demo": {
    "file_name": "alpaca_zh_demo.json"
  },
  "alpaca_gpt4_data": {
    "file_name": "alpaca_gpt4_data.json"
  },
```

- If data preprocessing is not required, the data preparation is complete.

5. [VeRL framework] Preprocess data.

   The VeRL framework requires that all datasets used for training must be preprocessed.

   a. Copy the content from **VeRL Data Processing Sample Script** to the **dataset_demo.py** file locally according to the model type. Modify **datasets.load_dataset** in the script.

   ```
   dataset = datasets.load_dataset(xxx/xxx/xxx) # Replace xxx/xxx/xxx with the full or relative path to your dataset folder or file.
   ```

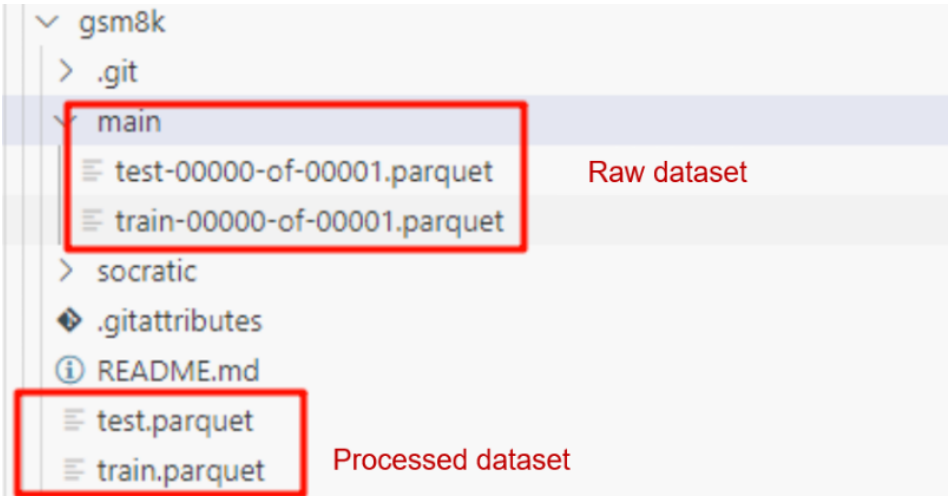
   b. Download the datasets.

   ```
   git clone https://huggingface.co/datasets/hiyouga/geometry3k # Multimodal dataset
   git clone https://huggingface.co/datasets/openai/gsm8k # LLM dataset
   ```

   c. Run the following command on the local PC to convert the dataset:

   ```
   python  dataset_demo.py --local_dir=/data/verl-workdir/data/xxx/
   ```

   --local_dir: path of the dataset generated after data processing



NOTE

The MindSpeed framework handles data conversion automatically using settings in the training YAML file, eliminating the need for manual effort.

In a multi-node setup, only the **rank_0** node handles data preprocessing and weight conversion. Ensure the original dataset, weights, and results are stored in a shared directory.

5.1.6.3 Preparing an Image

Step 1: Checking the Environment

1. Log in to the server via SSH and check the NPUs. Obtain the NPU device information:

```
npu-smi info                    # Run this command on each instance node to view the NPU status.
npu-smi info -l | grep Total    # Run this command on each instance node to view the total number of PUs and check whether these PUs have been mounted.
npu-smi info -t board -i 1 | egrep -i "software|firmware"    # Check the driver and firmware versions.
```

   If an error occurs, the NPU devices on the server may not be properly installed, or the NPU image may be mounted to another container. **Install the firmware and driver** or release the mounted NPUs.

   For details about the driver version requirements, see **Table 5-8**. If the driver version does not meet the requirements, upgrade the driver by **installing the firmware and driver**.

2. Check whether Docker is installed.

```
docker -v   # Check whether Docker is installed.
```

   If Docker is not installed, run this command:

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

3. Configure IP forwarding for intra-container network accesses. Run the following command to check the value of **net.ipv4.ip_forward**. Skip this step if the value is **1**.

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

   If the value is not **1**, configure IP forwarding:

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

   If the configuration item is not found, configure IP forwarding:

```
sed -i '$a\net.ipv4.ip_forward=1' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

4. Check whether the super pod IDs are the same.

```
for i in {0..7}; do npu-smi info -i $i -c 0 -t spod-info;done # Check the super pod IDs.
```

   Figure 5-1 Checking the super pod IDs

Step 2: Obtaining the Base Image

Use official images to deploy training. For details about the image path *{image_url}*, see **Table 5-8**.

```
docker pull {image_url}
```

Step 3: Creating a Training Image

Go to the folder (see key training files in the AscendCloud-LLM code package in **Software Package Structure**) containing the Dockerfile in the decompressed code directory and build the training image using the Dockerfile.

The installation requires cloning the Git repository online. Make sure your server has internet access.

1. Go to the code directory containing the Dockerfile. *${work_dir}* indicates the host directory storing the decompressed AscendCloud-LLM code. Change the directory based on your needs.
   ```
   cd ${work_dir}/llm_train/AscendFactory
   ```
2. Create an image.
   ```
   docker build --build-arg install_type=xxx -t <image_name>.
   ```
   Set up a proxy and include the **--build-arg** parameter with the proxy address if you cannot access the public network.
   ```
   docker build --build-arg "https_proxy=http://xxx.xxx.xxx.xxx" --build-arg "http_proxy=http://xxx.xxx.xxx.xxx" --network=host --build-arg install_type=xxx -t <image_name>.
   ```
   <*image_name*>: Custom image name. Example: **pytorch_2_3_ascend:20241106**

   install_type: installation type. The value can be **mindspeed-llm**, **llamafactory**, **verl**, **mindspeed-rl**, or **mindspeed-mm**.

CAUTION

1. When you build an image with a Dockerfile, the default working directory for the code is *${work_dir}***/llm_train/AscendFactory**. You can either **run the container image** to edit the code directly in this directory or rebuild the image.
2. Make sure the image name in the Dockerfile matches the one in **Table 5-8** of this document before building the image. Update it if needed.

```
# Modify the following content:
FROM swr.cn-southwest-2.myhuaweicloud.com/atelier/xxx
```

Step 4: Starting the Container Image

1. Before starting the container image, modify the parameters in ${} based on the parameter description. Add or modify parameters as needed. The following is sample commands to start a container.
   ```
   export work_dir="Custom working directory mounted to the container"     # Directory mounted to the container. If SFS is mounted, the mount directory can be used.
   export container_work_dir="Custom working directory mounted to the container"
   export container_name="Custom container name"
   export image_name="Image name"
   docker run -itd \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      -e ASCEND_VISIBLE_DEVICES=0-15 \
      -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 320 \
      --memory 2048g \
      --shm-size 1024g \
      --net=host \
      -v ${work_dir}:${container_work_dir} \
      --name ${container_name} \
      $image_name \
      /bin/bash
   ```

   Parameters:

- **--name** *${container_name}*: container name, which is used when you access the container. You can define a container name, for example, **ascendspeed**.
- **-v** *${work_dir}*:*${container_work_dir}*: host directory to be mounted to the container. The host and container use different file systems. **work_dir** indicates the working directory on the host. The directory stores files such as code and data required for training. *container_work_dir* indicates the directory to be mounted to the container. The two paths can be the same.

  NOTE

  - The **/home/ma-user** directory cannot be mounted to the container. This directory is the home directory of the **ma-user** user.
  - Both the driver and npu-smi must be mounted to the container.
  - Avoid assigning multiple containers to one NPU. Doing so will prevent its use in later containers.
- *${image_name}* indicates the ID of the Docker image, which can be queried by running the **docker images** command on the host.
- **--shm-size**: shared memory, which is used for communication between multiple processes. Model files with large memory need to be converted. The size must be 1,024 GB or larger.
- **--cpus**: number of CPU cores on the host. Generally, set this parameter to **192** for Snt9b servers and **320** for Snt9b23 servers.
- **--e ASCEND_VISIBLE_DEVICES=0-7**: PU IDs. Generally, keep **0-7** for Snt9b servers and change it to **0-15** for Snt9b23 servers.
- **--memory**: Generally, set this parameter to **1024g** for Snt9b servers and **2048g** for Snt9b23 servers.

2. Access the container using the container name. The default user is **ma-user** when the container is started.
   ```
   docker exec -it ${container_name} bash
   ```

5.1.7 Executing a Training Job

Step 1: Generating a YAML File for Training Configuration

1. Create the model training configuration file in YAML format. Choose between interactive mode or parameter input mode. In parameter input mode, provide all necessary parameters upfront. In interactive mode, choose required parameters after running the command. Select either of these methods based on your preference.

   a. Interactive mode:
   ```
   ascendfactory-cli config  --output_file_path=<output_file_path>
   ```

   b. Parameter input mode:
   ```
   ascendfactory-cli config --backend=<backend> --af_model_name=<af_model_name> --
   exp_name=<exp_name> --output_file_path=<output_file_path>
   ```

   - *<backend>*: framework type. The options are **mindspeed-llm**, **llamafactory**, **verl**, **mindspeed-rl**, and **mindspeed-mm**.
   - *<af_model_name>*: trained model.

- *<exp_name>*: experiment type. For MindSpeed-LLM and LlaMA-Factory fine-tuning, the options are **full-4k**, **lora-4k**, and more. For MindSpeed-LLM pre-training, the option is **full-4k**. (See **MindSpeed-LLM** for parameters.) For VeRL, the option are **ppo**, **grpo**, and **dapo**; for MindSpeed-RL, the option is **grpo**; for MindSpeed-MM, the option is **full**.
- *<output_file_path>*: output directory and file name of the YAML file, for example, **/path/to/xxx.yaml**.

2. Change the values of key parameters in the generated YAML file. For details about the parameters, see **MindSpeed-LLM**, **LlaMA-Factory**, **VeRL**, **MindSpeed-RL**, or **MindSpeed-MM**.

Step 2: Starting a Training Job

1. Run the training command in any directory, for example, the **test_benchmark** directory.

   For details about the minimum number of PUs in the pre-training and fine-tuning phases, see **Minimum Number of PUs and Sequence Length Supported by Each Model**.

   (optional) Single-node:

   ```
   # Default: 8 PUs
   ascendfactory-cli train <cfgs_yaml_file> --env.MASTER_ADDR=localhost --env.NNODES=1 --env.NODE_RANK=0
   # Specify the number of devices, for example, 2 PUs.
   ASCEND_RT_VISIBLE_DEVICES=0,1 ascendfactory-cli train <cfgs_yaml_file> --env.MASTER_ADDR=localhost --env.NNODES=1 --env.NODE_RANK=0
   # Specify the value of a parameter in the YAML file, for example, af_output_dir. The parameter is input using hyperparameter commands.
   ASCEND_RT_VISIBLE_DEVICES=0,1 ascendfactory-cli train <cfgs_yaml_file> --af_output_dir=xxx --env.MASTER_ADDR=localhost --env.NNODES=1 --env.NODE_RANK=0
   ```

   (Optional) Multiple nodes: Execute the following commands on multiple nodes at once:

   ```
   # Use the updated YAML file directly. Avoid inputting new parameters to change existing ones.
   ascendfactory-cli train <cfgs_yaml_file> --env.MASTER_ADDR=localhost --env.NNODES=1 --env.NODE_RANK=0
   # Specify the value of a parameter in the YAML file, for example, af_output_dir. The parameter is input using hyperparameter commands.
   ascendfactory-cli train <cfgs_yaml_file> --env.MASTER_ADDR=<master_addr> --env.NNODES=<nnodes> --env.NODE_RANK=<rank> --af_output_dir=xxx
   ```

   - *<cfgs_yaml_file>*: relative or absolute path of the configuration YAML file.
   - **--env.MASTER_ADDR=***<master_addr>*: IP address of the active master node. Generally, **rank 0** is selected as the active master node.
   - **--env.NNODES=***<nnodes>*: total number of training nodes.
   - **--env.NODE_RANK=***<rank>*: node ID, starting from 0. Generally, **rank 0** is selected as the active master node.
   - -- *Hyperparameter <key>*: For details about the parameter key, see **MindSpeed-LLM**, **LlaMA-Factory**, **VeRL**, **MindSpeed-RL**, or **MindSpeed-MM**.

5.1.8 Viewing Training Output Results

Viewing Logs and Weights

The MindSpeed-LLM framework prints its training loss and performance logs on the last rank node, while the MindSpeed-RL, VeRL, and Llama-Factory frameworks do so on the first rank node. The following shows the training result structure:

- MindSpeed-LLM

```
|——{af_output_dir} #{af_output_dir} Parameter value, for example, parameter configuration in the YAML file
   # Automatically generated data directory structure
   |——preprocess_data                    # Data preprocessing directory
   |——converted_weight_TP${TP}PP${PP}     # Directory of weights converted from the Hugging Face format to the Megatron format
   |——ckpt_converted_mg2hf                # Weights converted from the Megatron format to the Hugging Face format after training
   |——saved_checkpoints                   # Megatron weights after training
   |——training_loss.png                   # Loss curve
   |——exp-config.yaml                     # Training script (generated by the ascendfactory-cli config tool)
   |——logs                                # Training logs
     |——xx-xx-<Timestamp>-npu_info-R${RankID}.txt         # Training video RAM monitoring logs
     |——xx-xx-<Timestamp>-run_log-${Nodes}-${RankID}.txt   # Training run logs
   |——......
```

- Llama-Factory

```
|——{output_dir} #{output_dir} Parameter value, for example, parameter configuration in the YAML file
  # Automatically generated data directory structure
  |——model-000xx-of-000xx.safetensors      # Weight file
  |——trainer_log.jsonl                    # Loss values in the training JSONL file
  |——training_loss.png                    # Loss curve
  |——lora_merged                          # Default weight path after LoRA fine-tuning
  |——logs                                 # Training logs
    |——xx-xx-<Timestamp>-npu_info-R${RankID}_WS${world_size}.txt # Video memory monitoring logs
    |——xx-xx-<Timestamp>-run_log-R${RankID}_WS${world_size}.txt # Run logs
  |——......
```

- VeRL

```
|——{af_output_dir}                       # Log output directory
   # Automatically generated data directory structure
   |——exp-config.yaml                     # Training configuration file
   |——training_loss.png                   # Reward and response curve image
   |——plog                                # Model operator logs
     |——rank*
   |——logs                                # Training log directory
     |——verl_grpo-{af_model_name}-<Sequence length-Device type-Timestamp>-run_log-N1-WS8.txt # Loss file
   |——saved_checkpoints                   # Training log directory
     |——global_step_{number_1}
       |——actor                           # Weight file path. This directory is specified by the input parameter when weights are merged.
         |——huggingface                   # Output path for merged weights
     |——global_step_{number_N}
     |——latest_checkpointed_iteration.txt  # Iteration for saving the latest checkpoint
```

- MindSpeed-RL

```
|——{af_output_dir} #{af_output_dir} Parameter value, for example, parameter configuration in the YAML file
   # Automatically generated data directory structure
   |——preprocessed_data                   # Data preprocessing directory
   |——converted_hf2mg_weight_TP${TP}PP${PP}  # Directory of weights converted from the Hugging Face format to the Megatron format
   |——converted_mg2hf_weight              # Weights converted from the Megatron format to the Hugging Face format after training
   |——saved_checkpoints                   # Megatron weights after training
   |——logs                                # Training logs
     |——grpo_metrics-<Time>.log           # Training performance metric logs
```



```
    |——xx-xx-<Timestamp>-npu_info-R${RankID}.txt          # Training video memory monitoring logs
    |——xx-xx-<Timestamp>-run_log-${Nodes}-${RankID}.txt   # Training run logs
```

- MindSpeed-MM

```
|——{af_output_dir} #{af_output_dir} Parameter value, for example, parameter configuration in the YAML file
   # Automatically generated data directory structure
   |——converted_weight_TP${TP}PP${PP}        # Directory of weights converted from the Hugging Face format to the Megatron format
   |——ckpt_converted_mg2hf                   # Weights converted from the Megatron format to the Hugging Face format after training
   |——saved_checkpoints                      # Megatron weights after training
   |——exp-config.yaml                        # Training script (generated by the ascendfactory-cli config tool)
   |——logs                                   # Training logs
     |——xx-xx-<Timestamp>-npu_info-R${RankID}.txt          # Training video RAM monitoring logs
     |——xx-xx-<Timestamp>-run_log-${Nodes}-${RankID}.txt   # Training run logs
   |——.....
```

Checking Performance

Training performance is mainly checked based on two metrics in the training logs: throughput and convergence.

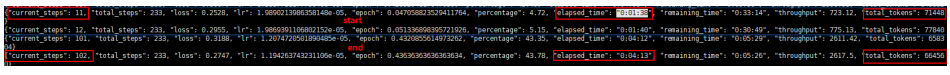
- MindSpeed-LLM
  a. Throughput (tokens/s/p): Global batch size x sequence length/(Total number of PUs x elapsed time per iteration) x 1000. The global batch size (GBS) and sequence length (SEQ_LEN) are set during training and printed in the logs.
  b. Loss convergence: The **lm loss** parameter exists in the log. Its value continuously decreases along with the training iteration period, and gradually becomes stable. You can also use the visualization tool **TrainingLogParser** to view the loss convergence.
- Llama-Factory
  a. Throughput (tokens/s/p): The throughput is calculated in **trainer_log.jsonl** under the path specified by *${output_dir}*. The value represents the average throughput across several intermediate steps. It is typically calculated and displayed in the training logs. The formula is as follows:

     delta_tokens = end_total_tokens-start_ total_tokens

     delta_time = end_elapsed_time - start_elapsed_time

     Throughput (tps) = delta_tokens/delta_time/Number of training PUs

     See the following figure.

  b. Loss convergence: The loss convergence graph is saved as **training_loss.png** in your output directory (*${output_dir}*). To visualize the results, upload the **trainer_log.jsonl** file using the **TrainingLogParser** tool.
- VeRL
  a. Throughput (tokens/s/p): The performance is calculated based on the **verl_grpo-***{af_model_name}*-*<Sequence length-Device type-Timestamp>*-**run_log-N1-WS8.txt** file in the path specified by *${output_dir}*. The **perf/**

throughput field in the training log file indicates the throughput of a single-step calculation. You can use the visualization tool **TrainingLogParser** to view the average training throughput.

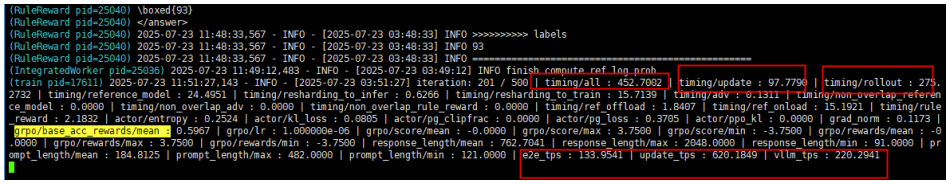
b. Score convergence: The **critic/score/mean** value in the log approaches 1 and stabilizes over time during training iterations. You can also use the visualization tool **TrainingLogParser** to view the convergence of **critic/score/mean**.

- MindSpeed-RL

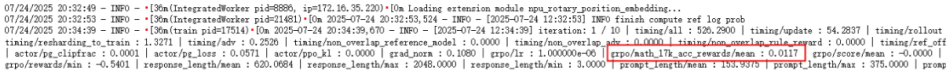
a. Throughput (tokens/s/p): The throughput is calculated in **grpo_metrics-*<Time>*.log** under the path specified by *${output_dir}*. The training log's final three metrics—**e2e_tps**, **update_tps**, and **vllm_tps**—show the end-to-end, training, and inference throughput for each step. Use the **TrainingLogParser** tool to see their overall average throughput.

b. **reward acc** convergence

- By default, Qwen2.5-7B and 32B use the DeepScaleR dataset, and their **verify_function** employs **base_acc**. This results in the presence of the **grpo/base_acc_rewards/mean** parameter in the logs. Its value steadily rises during training iterations before eventually stabilizing. You can also use the visualization tool **TrainingLogParser** to view the convergence of **grpo/base_acc_rewards/mean**.



- The Qwen2.5-1.5B model performs poorly. It defaults to using the math-17k dataset, with **verify_function** set as **math_17k_acc**. This creates the metric **grpo/math_17k_acc_rewards/mean** in the logs. Over time, as training progresses, this metric stabilizes. You can also use the visualization tool **TrainingLogParser** to view the convergence of **grpo/math_17k_acc_rewards/mean**.



- MindSpeed-MM

a. Throughput (samples/s): Global batch size/Elapsed time per iteration x 1,000. The global batch size (GBS) is set during training and printed in the logs.

b. Loss convergence: The loss value exists in the log. Its value continuously decreases along with the training iteration period, and gradually becomes stable. You can also use the visualization tool **TrainingLogParser** to view the loss convergence.

5.1.9 Collecting and Storing Logs

Viewing User Training Logs (Proc Log)

Proc logs display outputs from user training code. In single-node or multi-node Snt training jobs with multiple PUs, each Snt accelerator card creates a Python log on

the screen. These logs from different training processes on one node combine into the training log file mentioned in **Viewing Logs and Weights**. For troubleshooting, check the individual log files for each training process.

Log in to the container and run the following command to obtain the proc logs:

```
docker exec -it ${Container ID} bash
cd {container_work_dir}/{af_output_dir}/logs
```

- *{container_work_dir}* is the working directory in the container when the container image is started.
- *{af_output_dir}* is the directory for storing weights and logs configured in the YAML file of the training job.

CANN Application Logs

CANN logs record application activities generated by CANN. They consist of two types: host application logs (**plog-***{pid}*-*{time}***.log**) and device application logs (**device-***{pid}*-*{time}***.log**). These logs primarily contain:

- Logs printed by components (such as GE, FE, AI CPU, TBE, and HCCL) in the compiler and components (such as AscendCL, GE, and Runtime) in the runtime.
- Logs printed by the AI CPUs and HCCP on the device.

Log in to the container and run the following command to obtain the CANN logs:

```
docker exec -it ${Container ID} bash
cd {container_work_dir}/{af_output_dir}/plog
```

- *{container_work_dir}* is the working directory in the container when the container image is started.
- *{af_output_dir}* is the directory for storing weights and logs configured in the YAML file of the training job.
- The **run/plog** directory contains run logs, and the **debug/plog** directory contains debug logs for fault analysis and locating.

5.1.10 (Optional) Configuring Monitoring and Alarms

Monitoring Configurations

Using Cloud Eye to Monitor NPU Resources

Lite Server uses Cloud Eye for monitoring. Its required agent plug-in comes pre-installed. You can check NPU resource data directly through Cloud Eye. For details, see **Monitoring Lite Server Resources**.

Key Metrics

The table below lists the common monitoring metrics. For details about all supported metrics, see **Lite Server Monitoring Metrics**.

Table 5-11 Common monitoring metrics

No.	Category	Metric	Display Name	Description	Unit	Number System	Range
1	DDR	npu_util_rate_mem	NPU Memory Usage	NPU memory usage	%	N/A	0%–100%
2		npu_util_rate_mem_bandwidth	NPU Memory Bandwidth Usage	NPU memory bandwidth usage	%	N/A	0%–100%
3	HBM	npu_hbm_bandwidth_util	HBM Bandwidth Usage	NPU HBM bandwidth usage	%	N/A	0%–100%
4		npu_util_rate_hbm_bw	HBM Bandwidth Usage	NPU HBM bandwidth usage	%	N/A	0%–100%
5	AI Core	npu_util_rate_ai_core	NPU AI Core Usage	AI core usage of the NPU	%	N/A	0%–100%

No.	Category	Metric	Display Name	Description	Unit	Number System	Range
6	/	mem_usedPercent	Memory Usage	Memory usage of the monitored object. Linux: Obtain the metric value from the **/proc/meminfo** file: (**MemTotal** - **MemAvailable**)/ **MemTotal** If **MemAvailable** is displayed in **/proc/meminfo**, **MemUsedPercent** = (**MemTotal**- **MemAvailable**)/ **MemTotal** If **MemAvailable** is not displayed in **/proc/meminfo**, **MemUsedPercent** = (**MemTotal** – **MemFree** – **Buffers** – **Cached**)/ **MemTotal** Collection method (Windows): formula (Used memory size/ Total memory size x 100%)	%	N/A	0% – 100%

Alarm Settings

You can use Cloud Eye to collect key events and cloud resource operation events. When an event occurs, you will receive an alarm. You can check supported events in **Supported Events**.

5.1.11 Configuration Optimization and Fault Recovery

5.1.11.1 Configuration Optimization

This section explains standard optimizations and recommendations for various frameworks. Adjust settings as needed depending on your model, data, and available resources during implementation.

MindSpeed-LLM

1. Parallel policy optimization: Use model parallelism or long sequence parallelism when dealing with large model weights or lengthy sequences. This reduces video memory.

- **Key Parameters**
  - **tensor-model-parallel-size** (TP): model parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **pipeline-model-parallel-size** (PP): pipeline parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **context-parallel-size** (CP): sequence parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **expert-model-parallel-size** (EP): expert parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **data-parallel-size** (DP): The value is calculated using the following formula: Total number of training PUs/ (**tensor-model-parallel-size** x **pipeline-model-parallel-size** x **context-parallel-size**).
- **Policy adjustment**: Using model or long sequence parallelism adds extra communication costs and lowers training speed. Communication costs typically rise in this order: pipeline parallelism > data parallelism > expert parallelism > context parallelism > tensor parallelism. Adjust your parallel policy to find the best balance between video memory usage and training efficiency.

2. Video memory optimization:

- **use-distributed-optimizer**: After it is enabled, the optimizer status is split into each DP group to reduce the video memory consumption.
- **recompute-num-layers**: Specifies how many layers should fully recompute. When video memory is low, this saves only the input activations of each Transformer layer or group and recalculates the rest.
- **recompute-granularity**: Set it to **full** to perform full activation recomputation. Use this parameter together with the **recompute-num-layers** parameter.
- **recompute-method**: When set to **uniform**, it divides Transformer layers into equal-sized groups based on the **recompute-num-layers** value and stores inputs and activations per group. When set to **block**, it recalculates only the first *recompute-num-layers* Transformer layers, leaving the rest unchanged. Set this parameter to **block** if you have enough video memory to boost training speed.

- **recompute-activation-function**: Enables activation function recomputation.
- **recompute-activation-function-num-layers**: Specifies the number of layers for activation function recomputation. Note: Activation function recomputation can work with weight computation. If both are enabled, **--recompute-method** can only be set to **block**. In this case, full recomputation and activation function recomputation run independently per layer. A single layer cannot handle both types of recomputation simultaneously.
- **swap-attention**: When it is enabled, the system stores activation values in both device and CPU memory. It preloads these values from CPU during gradient backpropagation to avoid recalculating them. This approach uses H2D high bandwidth effectively, boosts storage and computation through networking, increases MFU, and speeds up training for foundation models.

3. Communication optimization:

- **overlap-grad-reduce**: When it is enabled, the system uses pipeline technology within the DP group to overlap both computations and communications.
- **overlap-param-gather**: Enabling it allows parameter gathering to overlap within the DP group. Ensure that both **--use-distributed-optimizer** and **--overlap-grad-reduce** are enabled.
- **use-ascend-coc**: When you enable it, it breaks down MatMul operations in ColumnParallelLinear and RowParallelLinear into smaller steps. It also splits nearby communication tasks—AllReduce when sequence parallelism is off, or AllGather and ReduceScatter when it is on—into finer parts within the TP group. This allows for better overlap.

LlaMA-Factory

1. ZeRO parallelism (**backend_config.training.deepspeed**): Memory usage changes based on the number of parameters in a model. Moving from ZeRO-Stage 0 to ZeRO-Stage 3 and enabling offload reduces memory usage significantly. However, this often increases communication and computation costs, potentially slowing down training.

   Configurations like **ds_config_zero0.json**, **ds_config_zero1.json**, **ds_config_zero2.json**, **ds_config_zero2_offload.json**, **ds_config_zero3.json**, and **ds_config_zero3_offload.json** are arranged from highest to lowest in terms of performance and memory usage.

2. Recomputation (**backend_config.training.recompute_layers_ratio**): Recomputation optimizes video memory by trading off performance.

   If you have enough video memory and need better performance, set **disable_gradient_checkpointing** to **true**. If video memory is low, set it to **false** instead. Use **recompute_layers_ratio** to find the right balance between memory savings and computation costs. Higher values save more memory but reduce performance.

3. Batch size of single-device training (**backend_config.training.per_device_train_batch_size**): Set **per_device_train_batch_size** to adjust training efficiency. Higher values boost performance but require more video memory. Setting it to **1** minimizes memory usage.

VeRL

1. Long sequence optimization: Set **data.max_prompt_length** and **data.max_response_length** to high values for long sequence training. If your model's max response length is 16k, set **data.max_response_length** to **16384**. Note: Too large a value can lead to low video memory. Adjust the sequence length carefully alongside other settings like sequence parallelism.
2. **actor_rollout_ref.actor.ulysses_sequence_parallel_size** splits long sequences into smaller blocks for parallel processing.
   - Adjustment rule: Set this value higher for longer sequences. For example, if **max_response_length** is **16384**, set **ulysses_sequence_parallel_size** to **8** to split the sequence into eight parts.
   - Key parameter: To avoid exceeding video memory limits, calculate **ppo_max_token_len_per_gpu** as **max_response_length** divided by **ulysses_sequence_parallel_size**. This ensures each device handles only its assigned portion of the sequence tokens during a PPO batch.
3. Batch processing parameter optimization
   - **data.train_batch_size**: Reduce it to fit within the video memory limits for longer sequences.
   - **actor_rollout_ref.rollout.n**: number of responses generated for each prompt during inference. The larger the value, the higher the video memory usage.
   - **actor_rollout_ref.actor.ppo_mini_batch_size**: The minimal configuration requires multiplying **actor_rollout_ref.rollout.n** by **ppo_mini_batch_size** to determine the number of PUs.
   - **actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu**, **actor_rollout_ref.rollout.log_prob_micro_batch_size_per_gpu**, and **actor_rollout_ref.ref.log_prob_micro_batch_size_per_gpu**: These parameters control the micro-batch size on a single device. The micro-batch size must be small enough to avoid video memory overflow. The smallest value allowed is **1**.

MindSpeed-RL

1. Inference phase parameter configuration
   - Parallelism parameters
     - **infer_tensor_parallel_size**: tensor parallel size. A larger value indicates lower video memory usage and higher communication overhead.
     - **infer_pipeline_parallel_size**: pipeline parallel size. A larger value indicates lower video memory usage and higher communication overhead.
     - **infer_expert_parallel_size**: expert parallel size. This parameter is used for MOE models. For DeepSeek models, large-scale EP is recommended.
   - VLLM engine parameters
     - **max_num_seqs**: maximum number of sequences processed each time

- **max_model_len**: maximum output sentence length
- **max_num_batched_tokens**: maximum number of batched tokens in each iteration
- **gpu_memory_utilization**: reserved video memory space for model weights and KV cache
- **enforce_eager**: Setting this parameter to **True** disables the graph mode.

2. Training phase parameter configuration
   - Parallelism parameters
     - **tensor_model_parallel_size**: tensor parallel size. A larger value indicates lower video memory usage and higher communication overhead.
     - **pipeline_model_parallel_size**: pipeline parallel size. A larger value indicates lower video memory usage and higher communication overhead.
     - **expert_model_parallel_size**: expert parallel size
     - **context_parallel_size**: sequence parallel size. A larger value indicates lower video memory usage and higher communication overhead.
   - Video memory optimization:

     swap_optimizer: Moves the optimizer state to host memory during forward and backward propagation, keeping only its logical view on the device. It reloads the state to the device for updates, lowering peak video memory consumption.
   - Batch processing parameters:
     - **global_batch_size**: amount of prompt data processed in a step
     - **mini_batch_size**: The value of **global_batch_size** divided by **mini_batch_size** indicates the number of times that the actor model is updated in each step.
     - **micro_batch_size**: micro batch size in the training phase. This parameter does not take effect when **use_dynamic_bsz** is enabled.
   - RL parameters:
     - **use_integrated_worker**: Set this parameter to **true** to enable integrated PUs.
     - **use_dynamic_bsz**: Enable it to use dynamic batching to keep sample counts consistent across DP domains and align data lengths closely.
     - **max_packing_token_size**: Specifies maximum number of tokens in a batch when dynamic batching is used.
     - **blocking**: Specifies whether to enable asynchronous processing. The default value is **true**.

  - **use_remove_padding**: Removes padding tokens to avoid invalid calculations.
  - **n_samples_per_prompt**: Specifies number of responses generated for each prompt.

MindSpeed-MM

1. Parallel policy optimization: Use model parallelism or long sequence parallelism when dealing with large model weights or lengthy sequences. This reduces video memory.

- **Key Parameters**
  - **tensor-model-parallel-size** (TP): model parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **pipeline-model-parallel-size** (PP): pipeline parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **context-parallel-size** (CP): sequence parallel size. A larger value indicates lower video memory usage and higher communication overhead.
  - **data-parallel-size** (DP): The value is calculated using the following formula: Total number of training PUs/ (**tensor-model-parallel-size** x **pipeline-model-parallel-size** x **context-parallel-size**).
- **Policy adjustment**: Using model or long sequence parallelism adds extra communication costs and lowers training speed. Communication costs typically rise in this order: pipeline parallelism > data parallelism > context parallelism > tensor parallelism. Adjust your parallel policy to find the best balance between video memory usage and training efficiency.

2. Video memory optimization:

- **use-distributed-optimizer**: After it is enabled, the optimizer status is split into each DP group to reduce the video memory consumption.
- **recompute-num-layers**: Specifies how many layers should fully recompute. When video memory is low, this saves only the input activations of each Transformer layer or group and recalculates the rest.
- **recompute-granularity**: Set it to **full** to perform full activation recomputation. Use this parameter together with the **recompute-num-layers** parameter.
- **recompute-method**: When set to **uniform**, it divides Transformer layers into equal-sized groups based on the **recompute-num-layers** value and stores inputs and activations per group. When set to **block**, it recalculates only the first *recompute-num-layers* Transformer layers, leaving the rest unchanged. Set this parameter to **block** if you have enough video memory to boost training speed.
- **recompute-activation-function**: Enables activation function recomputation.
- **recompute-activation-function-num-layers**: Specifies the number of layers for activation function recomputation. Note: Activation function recomputation can work with weight computation. If both are enabled, **--**

recompute-method can only be set to **block**. In this case, full recomputation and activation function recomputation run independently per layer. A single layer cannot handle both types of recomputation simultaneously.

- **swap-attention**: When it is enabled, the system stores activation values in both device and CPU memory. It preloads these values from CPU during gradient backpropagation to avoid recalculating them. This approach uses H2D high bandwidth effectively, boosts storage and computation through networking, increases MFU, and speeds up training for foundation models.

3. Communication optimization:

- **overlap-grad-reduce**: When it is enabled, the system uses pipeline technology within the DP group to overlap both computations and communications.
- **overlap-param-gather**: Enabling it allows parameter gathering to overlap within the DP group. Ensure that both **--use-distributed-optimizer** and **--overlap-grad-reduce** are enabled.
- **use-ascend-coc**: When you enable it, it breaks down MatMul operations in ColumnParallelLinear and RowParallelLinear into smaller steps. It also splits nearby communication tasks—AllReduce when sequence parallelism is off, or AllGather and ReduceScatter when it is on—into finer parts within the TP group. This allows for better overlap.

5.1.11.2 Resumable Training

Resumable training lets you restart training using the most recent saved weights if the process gets interrupted. This avoids redundant calculations and boosts efficiency. To resume training after an interruption, follow these steps.

MindSpeed-LLM

1. Go to the training result output directory and check the saved weights.
   ```
   cd ${af_output_dir}
   ```
   {af_output_dir} is the output folder set by the **--save** parameter in the training script. It includes several **iter**_*xx* weight folders and a **latest_checkpointed_iteration.txt** file that tracks the last checkpointed step. To resume training, edit this file to choose which weight folder to load.
2. Modify the following resumable training configurations in the YAML training configuration file.
   ```
   backend_config.training.no-load-optim: false
   backend_config.training.no-load-rng: false
   backend_config.training.finetune: false
   ```
3. Restart the training job. For details, see **Step 2: Starting a Training Job**. Add the following hyperparameter to the **ascendfactory-cli train** command.
   ```
   --load ${af_output_dir}/saved_checkpoints
   ```

LlaMA-Factory

1. Go to the training result output directory and obtain the specified checkpoint directory.
   ```
   cd ${af_output_dir}
   ```
   {af_output_dir} stores outputs as defined in the training YAML file. It includes several checkpoint folders named **checkpoint-***xx*. These folders are created

based on **backend_config.training.max_steps** and **backend_config.training.save_steps**.

2. Modify the following resumable training configurations in the YAML training configuration file.
```
backend_config.training.resume_from_checkpoint: ${af_output_dir}/checkpoint-xx
```
3. Restart the training job. For details, see **Step 2: Starting a Training Job**.

VeRL

1. Go to the training result output directory and obtain the specified checkpoint directory.
```
cd ${af_output_dir}/checkpoint_${af_model_name}
```
The YAML training configuration file sets the checkpoint location using **backend_config.trainer.default_local_dir**. This folder contains multiple weight directories named **global_step**_*XX*. Choose the latest one.
2. Edit the YAML training configuration file to resume training from a specific checkpoint or the most recent one.
```
Method 1: Training from a specified checkpoint
backend_config.trainer.resume_mode: resume_path
backend_config.trainer.resume_from_path: ${af_output_dir}/checkpoint_${af_model_name}/
global_step_xx

Method 2: Training from the latest checkpoint
backend_config.trainer.resume_mode: auto
```
3. Restart the training job. For details, see **Step 2: Starting a Training Job**.

MindSpeed-RL

1. Go to the training result output directory and obtain the specified checkpoint directory.
```
cd ${af_output_dir}/checkpoint_${af_model_name}
```
backend_config.trainer.default_local_dir shows several **global_step**_*XX* weight folders. Choose the most recent one.
2. Modify the following resumable training configurations in the YAML training configuration file.
```
actor_config:
  finetune: false # Set finetune to false for resumable training.
  load: ./ckpt-32b # Load the path to previously saved weights for resumable training.
  save: ./ckpt
  no_load_optim: false # Set no_load_optim to false for resumable training.
  no_load_rng: false # Set no_load_rng to false for resumable training.

rf_config:
  integrated_mode_config:
    ref_model_load_path: ./Qwen2.5-7B-tp4 # Set the reference model's weight path to the original
model for resumable training.
```
3. Restart the training job. For details, see **Step 2: Starting a Training Job**.

MindSpeed-MM

1. Go to the training result output directory and check the saved weights.
```
cd ${af_output_dir}
```
{af_output_dir} is the output folder set by the **--save** parameter in the training script. It includes several **iter**_*xx* weight folders and a **latest_checkpointed_iteration.txt** file that tracks the last checkpointed step. To resume training, edit this file to choose which weight folder to load.

2. Modify the following resumable training configurations in the YAML training configuration file.
```
backend_config.training.no-load-optim: false
backend_config.training.no-load-rng: false
```
3. Restart the training job. For details, see **Step 2: Starting a Training Job**. Add the following hyperparameter to the **ascendfactory-cli train** command.
```
--load ${af_output_dir}/saved_checkpoints
```

5.1.12 Training Service Configurations

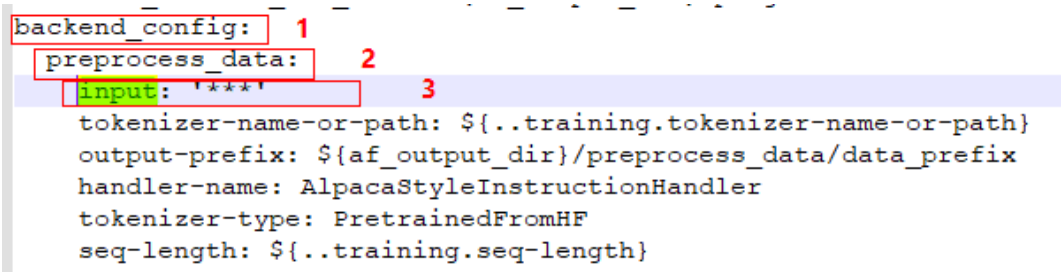
5.1.12.1 Parameters

5.1.12.1.1 MindSpeed-LLM

This section describes the YAML configuration file and parameters for training. You can choose parameters as required.

Configuring Parameters in the YAML File

Modify the YAML file.



1. Choose either of the following dataset parameters.

Parameter	Example Value	Description
backend_config.preprocess_data.input	[Pre-training: pt] Relative or absolute address of the **pre-training dataset** [Fine-tuning: sft] Relative or absolute address of the **fine-tuning dataset**	Input data path specified during training. Change it based on the actual situation. Select either of the following options.
backend_config.training.data-path	**/home/ma-user/ws/xxx**	Directory of the processed data. If the data has been processed, set this parameter.

2. Set the training scenario, weight file, output directory, and other important parameters. The details are as follows.

Parameter	Example Value	Description
backend_config.training.tokenizer-name-or-path	**/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B**	(Mandatory) Path for storing the tokenizer and Hugging Face weight files. Change it based on the actual situation.
af_output_dir	**/home/ma-user/ws/save_dir**	(Mandatory) Directory for storing logs and weight files generated after the training is complete.
backend_config.preprocess_data.handler-name	• **GeneralPretrainHandler** • **AlpacaStyleInstructionHandler** • **SharegptStyleInstructionHandler**	(Mandatory) Select a value based on the *${dataset}*. • **GeneralPretrainHandler**: Use the pre-trained Alpaca dataset. • **AlpacaStyleInstructionHandler**: Use the fine-tuned Alpaca dataset. • **SharegptStyleInstructionHandler**: Use the Sharegpt dataset.
backend_config.convert_ckpt_mg2hf	**null**	Specifies whether to convert the weights in the Megatron format to those in the Hugging Face format during training. The conversion is performed by default. If this parameter is set to **null**, the conversion is not performed.
Resumable training		
backend_config.training. no-load-optim backend_config.training. no-load-rng	**false**	Specifies whether to load the optimizer state. • **false**: The optimizer state is not loaded. • **true**: The optimizer state is loaded.
backend_config.training.finetune	**false**	Specifies whether to reset the optimizer state and iteration count to 0. • **true**: Yes • **false**: No

Parameter	Example Value	Description
backend_config.training.load	**path/to/***xxx*	Loads Megatron weights generated during training.
Pre-training		
backend_config.training.stage	**false**	Training type. Set this parameter to **false** for pre-training. • **false**: pre-training • **sft**: instruction fine-tuning
backend_config.preprocess_data.handler-name	**GeneralPretrainHandler**	Specifies a handler for processing datasets.
backend_config.training.is-instruction-dataset	**false**	Specifies whether the dataset is structured. • **true**: Yes • **false**: No
backend_config.training.finetune	**false**	Specifies whether to reset the optimizer state and iteration count to 0. • **true**: Yes • **false**: No
reset-position-ids	**true**	Resets the index after the document end marker. Main scenario: when pack mode is used in dataset processing.

3. Set other parameters.

Parameter	Example Value	Description
backend_config.training.micro-batch-size	**1**	Number of samples processed by a micro batch in pipeline parallelism. In pipeline parallelism, data of a step is divided into multiple micro batches to reduce the bubble time. The value is related to **tensor-model-parallel-size**, **pipeline-model-parallel-size**, and the model size. You can adjust the value based on the site requirements. The value is referred to as MBS.

Parameter	Example Value	Description
backend_config.training.global-batch-size	**128**	Number of samples processed by all servers in a step during training, which affects the training iteration time. Short name: GBS.
backend_config.training.tensor-model-parallel-size	**8**	Tensor parallelism, referred to as TP.
backend_config.training.pipeline-model-parallel-size	**4**	Pipeline parallelism. Generally, the value of this parameter is the number of training nodes, which is the same as the value configured during weight conversion. Short name: PP.
backend_config.training.context-parallel-size	**1**	Context parallelism. The default value is **1**. This parameter applies to training models with long sequences. If **SEQ_LEN** exceeds **32768** during training, you are advised to set this parameter to a value greater than or equal to **2**. The value is referred to as **CP**.
backend_config.training.lr	**2.5e-5**	Learning rate.
backend_config.training.min-lr	**2.5e-6**	Minimum learning rate.
backend_config.training.train-iters	**10**	Number of training iterations, with a default value. This parameter is optional.
backend_config.training.save-interval	**1000**	Model saving interval. • If the parameter value is greater than or equal to **TRAIN_ITERS**, only the last version of the model that has been trained for **TRAIN_ITERS** times is saved. • If the parameter value is less than **TRAIN_ITERS**, the model version is saved every **SAVE_INTERVAL** times. Number of saved model versions = **TRAIN_ITERS**/**SAVE_INTERVAL** + 1

Parameter	Example Value	Description
backend_config.training.save-total-limit	**-1**	Controls the number of times that the weight version is saved. • It has no impact if left unset or given a value of zero or less. • The parameter value must be less than or equal to **TRAIN_ITERS**/**SAVE_INTERVAL** + 1. • If the parameter value is greater than 1, the number of model version saving times is the same as the value of **SAVE_TOTAL_LIMIT**.
backend_config.training.load	**null**	Weight loading path. By default, the weights are loaded. If the value is **null**, the weights are not loaded.

Model Parameter Constraints

- Tensor parallelism, pipeline parallelism, and context parallelism: The value of TP x PP x CP must be exactly divisible by the number of NPUs (**word_size**).
- The value of TP x CP must be exactly divisible by **num_attention_heads** in the model parameters.
- MBS (**micro-batch-size**) and GBS (**global-batch-size**): The values of GBS and MBS must be exactly divisible by NPU/(TP x PP x CP).

5.1.12.1.2 LlaMA-Factory

This section describes the YAML configuration file and parameters for training. You can choose parameters as required.

YAML File Configuration

Modify the YAML file.

```
backend_config: 1
  training: 2
    do_train:true 3
    stage:sft
    dataset_dir:../data
    dataset:gsm8k_train_alpaca
    cutoff_len:4096
```


Table 5-12 Model training script parameters

Parameter	Example Value	Description
backend_config.training.dataset	• Instruction supervision fine-tuning: **alpaca_en_demo** • Multimodal dataset (image): **mllm_demo,identity**	(Mandatory) Dataset name registered in the **dataset_info.json** file. If you use custom data, configure the **dataset_info.json** file by referring to **README_zh.md** and store the dataset in the same directory as the **dataset_info.json** file.
backend_config.training.dataset_dir	**/home/ma-user/AscendFactory/third-party/LLaMA-Factory/data**	(Mandatory) Specify this parameter as a hyperparameter in the **ascendfactory-cli train** *XXX* command. Dataset included in the LlaMA-Factory code package: *${INSTALL_DIR}*/**third-party/LLaMA-Factory/data**, where *${INSTALL_DIR}* is related to the setting in **install.sh**. **Custom data**: **AscendFactory/data**
backend_config.training.model_name_or_path	**/home/ma-user/xxx/model/Qwen2-72B**	(Mandatory) Specify this parameter as a hyperparameter in the **ascendfactory-cli train** *XXX* command. Path for storing the tokenizer and Hugging Face weight files. Change it based on the actual situation.
backend_config.training.adapter_name_or_path	**/home/ma-user/xxx/sft_lora/**	Unmerged LoRA weights generated after LoRA training is complete. Transfer the LoRA-based fine-tuned model's weight file during incremental training.
af_output_dir	**/home/ma-user/xxx/saves/qwen2-7b/sft_lora/**	(Mandatory) Specify this parameter as a hyperparameter in the **ascendfactory-cli train** *XXX* command. Specifies the output directory. The model parameters and log files generated during training are stored in this directory.

Parameter	Example Value	Description
backend_config.training.train_from_scratch	**false**	Indicates whether the model is trained from scratch. The default value is **false**. • **true**: The model is trained from scratch, and the weights are not loaded. • **false**: The model is incrementally trained from the position where the weights are loaded.
backend_config.training.do_train	**true**	Specifies whether to execute the training step of the script. If this parameter is set to **true**, model training is performed. If this parameter is set to **false**, model training is not performed.
backend_config.training.cutoff_len	**4096**	Maximum length of text processing. Change the value as required.
backend_config.training.packing	**true**	(Optional) Using static data length fills incomplete data up to the maximum text processing size. With dynamic data length, this parameter gets removed.
backend_config.training.deepspeed	-	(Optional) ZeRO optimization strategy. The options are as follows: • ds_config_zero0.json • ds_config_zero1.json • ds_config_zero2.json • ds_config_zero3.json • ds_config_zero2_offload.json • ds_config_zero3_offload.json
backend_config.training.stage	**sft**	Specifies the current training phase. Currently, only **sft** is supported. **sft** indicates supervised fine-tuning.
backend_config.training.finetuning_type	**full**	Specifies the fine-tuning strategy. The value can be **full** or **lora**. If this parameter is set to **full**, the entire model is fine-tuned.

Parameter	Example Value	Description
backend_config.training.lora_target	**all**	Target modules that adopt LoRA. The default value is **all**.
backend_config.training.template	**qwen**	Specifies the template. If this parameter is set to **qwen**, the QWEN template is used for training.
backend_config.training.max_samples	**50000**	Specifies the maximum number of samples used during training. If this parameter is set, only the specified number of samples are used in the training process, and other samples are ignored. This can be used to control the scale and computing requirements of the training process.
backend_config.training.num_train_epochs backend_config.training.max_steps	**5** **5000**	Number of training epochs. Change it based on the actual situation. An epoch is a process where all training samples are trained once. Number of training steps. (Either one). If both parameters are set, only **max_steps** takes effect.
backend_config.training.overwrite_cache	**true**	Specifies whether to overwrite the cache. If this parameter is set to **overwrite_cache**, the cache is overwritten during training. This is usually used when the dataset changes or the cache needs to be regenerated.
backend_config.training.preprocessing_num_workers	**16**	Number of worker threads for preprocessing data. As the number of threads increases, the preprocessing speed increases, but the memory usage also increases.
backend_config.training.per_device_train_batch_size	**1**	Batch size for training on each device.
backend_config.training.gradient_accumulation_steps	**8**	Number of gradient accumulation steps. This can increase the batch size without increasing the memory consumption.

Parameter	Example Value	Description
backend_config.training.logging_steps	**2**	Number of interval steps for outputting logs during model training. Logs include information such as the training progress, learning rate, and loss value. You are advised to set this parameter.
backend_config.training.save_steps	**5000**	Number of interval steps for saving a model during model training. Saved models can be used for subsequent training. • If the parameter value is greater than or equal to **max_steps**, only the last version of the model that has been trained for **TRAIN_ITERS** times is saved. • If the parameter value is less than **max_steps**, the model version is saved every **save_steps** times. Number of saved model versions = **max_steps**/**save_steps** +1
backend_config.training.save_total_limit	**0**	Controls the number of times that the weight version is saved. • It has no impact if left unset or given a value of zero or less. • The parameter value must be less than or equal to **max_steps**/**save_steps** + 1. • If the parameter value is greater than 1, the number of model version saving times is the same as the value of **save_total_limit**.
backend_config.training.plot_loss	**true**	Specifies whether to draw the loss curve. If this parameter is set to **true**, the loss curve is saved as an image after the training is complete.

Parameter	Example Value	Description
backend_config.training.overwrite_output_dir	**true**	Specifies whether to overwrite the output directory. The default value is **true**. If this parameter is set to **true**, the output directory is cleared at the beginning of each training. If this parameter is set to **false**, the system loads the most recent training weights saved after an interruption and resumes training. Set **resume_from_checkpoint** to enable fast fault recovery.
backend_config.training.resume_from_checkpoint	*{output_dir}*/**checkpoint-***xx*	Resumable training: Resumes an unexpectedly stopped training job by loading weights from the **checkpoint-***xx* folder. This setting lets you continue training from where it left off. This parameter takes precedence over **overwrite_output_dir**.
backend_config.training.bf16/fp16	**true**	(Optional) Precision format, which defaults to **bf16**.
backend_config.training.learning_rate	**2.0e-5**	Specifies the learning rate.
backend_config.training.disable_gradient_checkpointing	**true**	Specifies whether to disable gradient checkpointing. By default, gradient checkpointing is enabled. It saves the model's state during deep learning training for easy restoration later. While this reduces memory usage, particularly with large models, it may impact performance. **True**: Gradient checkpointing is disabled.
backend_config.training.include_tokens_per_second backend_config.training.include_num_input_tokens_seen	**true**	Tokens processed per second and input tokens encountered during training. This parameter is used to measure the performance.
backend_config.lora_merge.export_dir	*${af_output_dir}*/**lora_merged**	Specifies where to save the combined weights after merging LoRA weights with the original model during fine-tuning.

Parameter	Example Value	Description
backend_config.training.recompute_layers_ratio	Float value range: [0,1]	Specifies how many layers use recomputation. If you have enough video memory, adjusting this value helps maximize unused memory for better performance. It will not work if **disable_gradient_checkpointing** is set **true**.

5.1.12.1.3 VeRL

This section describes the YAML configuration file and parameters for training. You can choose parameters as required.

YAML File Configuration

Edit the YAML file using these instructions. Parameters like *aaa.bbb* show the value of *bbb* under *aaa*. For example, **backend_config.data.train_files** refers to the **data.train_files** parameter of **backend_config**.

```
backend_config: 1
  algorithm.adv_estimator: grpo
  algorithm.use_kl_in_reward: false
  data.train_files: '***' 2
  data.val_files: '***'
  data.train_batch_size: 64
  data.max_prompt_length: 2048
  data.max_response_length: 16384
  data.filter_overlong_prompts: true
  data.truncation: error
  actor_rollout_ref.model.path: '***'
```

Table 5-13 Model training script parameters

Parameter	Example Value	Description
af_output_dir	**/home/ma-user/verl**	(Mandatory) Training output result
backend_config.data.train_files	**/data/geometry3k/train.parquet**	(Mandatory) Training set after preprocessing
backend_config.data.val_files	**/data/geometry3k/test.parquet**	(Mandatory) Validation set after preprocessing

Parameter	Example Value	Description
backend_config.actor_rollout_ref.model.path	**/model/Qwen2.5-VL-32B-Instruct**	(Mandatory) Hugging Face model path, which can be a local path or an HDFS path.
backend_config.data.train_batch_size	**32**	Batch size for one training sampling.
backend_config.actor_rollout_ref.actor.ppo_mini_batch_size	**8**	Batch size split into multiple sub-batches.
backend_config.actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu	**1**	Data volume for a single forward propagation when training the Actor on one device.
backend_config.actor_rollout_ref.rollout.log_prob_micro_batch_size_per_gpu	**1**	Data volume a single device processes in one forward propagation while calculating log probabilities during rollout.
backend_config.actor_rollout_ref.ref.log_prob_micro_batch_size_per_gpu	**1**	Data volume each devic processes during one forward propagation for calculating the reference policy's log probabilities.
backend_config.trainer.total_epochs	**5**	(Optional) Number of training epochs, which you can set depending on your needs.
backend_config.actor_rollout_ref.rollout.tensor_model_parallel_size	**4**	Model segmentation during vLLM inference.
backend_config.data.image_key	**images**	(Multimodal model) Field where images in the dataset are located. The default value is **'images'**.
engine_kwargs.vllm.disable_mm_preprocessor_cache	**True**	(Multimodal model) Specifies whether to disable the preprocessor cache of multimodal models. The default value is **False**.
backend_config.data.max_prompt_length	**1024**	Maximum prompt length. All prompts will be left-padded to this length.
backend_config.data.max_response_length	**1024**	Maximum length of responses Maximum generation length during the rollout phase in the RL algorithm.

Parameter	Example Value	Description
backend_config.actor_rollout_ref.rollout.max_num_batched_tokens	**18432**	When **max_response_length** + **max_prompt_length** is greater than 8k, set this parameter to the sum of the two parameters. The default value is **8192**.
backend_config.actor_rollout_ref.actor.ulysses_sequence_parallel_size	**1**	Sequence parallelism. This parameter is used for long sequences. The default value is **1**. When the value of **max_response_length** is greater than 8k, the value is generally rounded off (**max_response_length**/**2048**).
backend_config.data.shuffle	**True**	Specifies whether to shuffle the data in dataloader.
backend_config.data.truncation	**'error'**	Truncates **input_ids** or **prompt length** if they exceed **max_prompt_length**. The default value is **'error'**, which means they must exceed **max_prompt_length**.
backend_config.actor_rollout_ref.actor.optim.lr	**1e-6**	Actor learning rate
backend_config.actor_rollout_ref.model.use_remove_padding	**True**	Specifies whether to remove padding. • **True**: Yes • **False**: No
backend_config.actor_rollout_ref.actor.use_kl_loss	**True**	Specifies whether to use KL loss in the actor. If yes, KL is not used in the reward function. The default value is **True**.
backend_config.actor_rollout_ref.actor.kl_loss_coef	**0.01**	KL loss coefficient. The default value is **0.001**.
backend_config.actor_rollout_ref.actor.kl_loss_type	**low_var_kl**	Specifies how to calculate the KL divergence between the participant and the reference policy. The default value is **low_var_kl**.
backend_config.actor_rollout_ref.actor.entropy_coeff	**0**	Calculates the PPO loss. Generally, set this parameter to **0**.

Parameter	Example Value	Description
backend_config.actor_rollout_ref.actor.use_torch_compile	**False**	Specifies whether to enable JIT compilation acceleration. The value can be **False**.
backend_config.actor_rollout_ref.model.enable_gradient_checkpointing	**True**	Specifies whether to enable gradient checkpointing for the actor.
backend_config.actor_rollout_ref.rollout.name	**vllm**	Inference framework name vllm.
backend_config.actor_rollout_ref.rollout.gpu_memory_utilization	**0.4**	Ratio of the total device memory to the vLLM instances.
backend_config.actor_rollout_ref.rollout.n	**4**	Repeats a batch of data for *n* times (interleaving is performed during the repetition).
backend_config.trainer.logger	**['console','tensorboard']**	Log backend. The options are **wandb**, **console**, and **tensorboard**.
backend_config.trainer.val_before_train	**False**	Specifies whether to run the validation set test before training. ● **True**: Yes ● **False**: No
backend_config.trainer.resume_mode	**auto**	The default value is **auto**. Set this parameter to **auto** to load the most recent training weights from the default save location if training stops unexpectedly. For resumable training, change the parameter to **resume_path** and specify **resume_from_path**. Set it to **disable** to disable resumable training.
backend_config.trainer.resume_from_path	**null**	The default value is **null**. Set this parameter to specify the path to load weight parameters for resumable training.
backend_config.trainer.save_freq	**-1**	Frequency of saving model weight parameters (by iteration). The default value is **-1**, indicating that weight parameters are not saved.

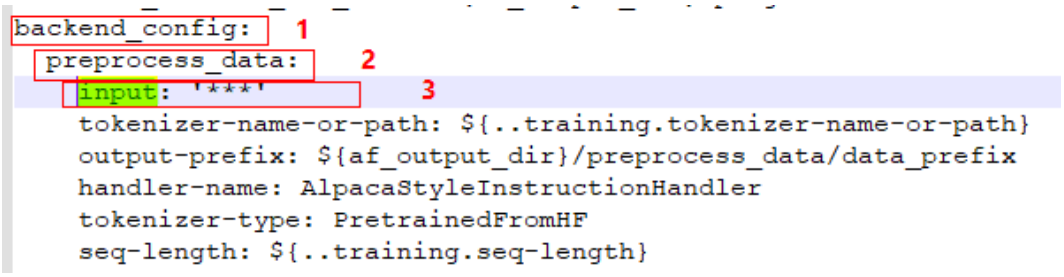
Parameter	Example Value	Description
backend_config.actor_rollout_ref.actor.ppo_max_token_len_per_gpu	**2048**	Maximum number of tokens that can be processed by a single GPU in a PPO micro batch size. Generally, set this parameter to **n** x (**data.max_prompt_length** + **data.max_response_length**).

5.1.12.1.4 MindSpeed-RL

This section describes the YAML configuration file and parameters for training. You can choose parameters as required.

Configuring Parameters in the YAML File

Modify the YAML file.



1. Choose either of the following dataset parameters.

Parameter	Example Value	Description
backend_config.preprocess_data.input	Relative or absolute address of the **dataset**	Input data path specified during training. Change it based on the actual situation. Select either of the following options.
backend_config.megatron_training.data_path	**/home/ma-user/ws/xxx**	Directory of the processed data. If the data has been processed, set this parameter.

2. Set the training scenario, weight file, output directory, and other important parameters. The details are as follows.

Parameter	Example Value	Description
backend_config.megatron_training.tokenizer_name_or_path	**/home/ma-user/ws/llm_train/AscendFactory/model/llama2-70B**	(Mandatory) Path for storing the tokenizer and Hugging Face weight files. Change it based on the actual situation.
af_output_dir	**/home/ma-user/ws/save_dir**	(Mandatory) Directory for storing logs and weight files generated after the training is complete.
backend_config.preprocess_data.handler_name	• **GeneralPretrainHandler** • **GeneralInstructionHandler** • **MOSSInstructionHandler** • **AlpacaStyleInstructionHandler** • **SharegptStyleInstructionHandler**	(Mandatory) Select a value based on the *${dataset}*. • **GeneralPretrainHandler**: Use the pre-trained Alpaca dataset. • **GeneralInstructionHandler**: Use the fine-tuned Alpaca dataset. • **MOSSInstructionHandler**: Use the fine-tuned MOSS dataset. • **AlpacaStyleInstructionHandler**: Use the fine-tuned Alpaca dataset. • **SharegptStyleInstructionHandler**: Use the Sharegpt dataset.
backend_config.actor_config.no_load_optim backend_config.actor_config.no_load_rng	**false**	Specifies whether to load the optimizer status. • **false**: The optimizer status is not loaded. • **true**: The optimizer status is loaded.

3. Set other parameters.

Parameter	Example Value	Description
backend_config.actor_config.micro_batch_size	**1**	Number of samples processed by a micro batch in pipeline parallelism. In pipeline parallelism, data of a step is divided into multiple micro batches to reduce the bubble time. The value is related to **tensor-model-parallel-size**, **pipeline-model-parallel-size**, and the model size. You can adjust the value based on the site requirements. The value is referred to as MBS.
backend_config.megatron_training.global_batch_size	**128**	Number of samples processed by all servers in a step during training, which affects the training iteration time. Short name: GBS.
backend_config.actor_config.tensor_model_parallel_size	**8**	Tensor parallelism, referred to as TP.
backend_config.actor_config.pipeline_model_parallel_size	**4**	Pipeline parallelism. Generally, the value of this parameter is the number of training nodes, which is the same as the value configured during weight conversion. Short name: PP.
backend_config.actor_config.lr	**2.5e-5**	Learning rate.
backend_config.actor_config.min_lr	**2.5e-6**	Minimum learning rate.
backend_config.megatron_training.train_iters	**10**	Number of training iterations, with a default value. This parameter is optional.
backend_config.megatron_training.save_interval	**1000**	Model saving interval. • If the parameter value is greater than or equal to **TRAIN_ITERS**, only the last version of the model that has been trained for **TRAIN_ITERS** times is saved. • If the parameter value is less than **TRAIN_ITERS**, the model version is saved every **SAVE_INTERVAL** times. Number of saved model versions = **TRAIN_ITERS**/**SAVE_INTERVAL** + 1
backend_config.actor_config.load	**null**	Weight loading path. By default, the weights are loaded. If the value is **null**, the weights are not loaded.

For other parameters not mentioned, see the **parameter configuration**.

Model parameter constraints

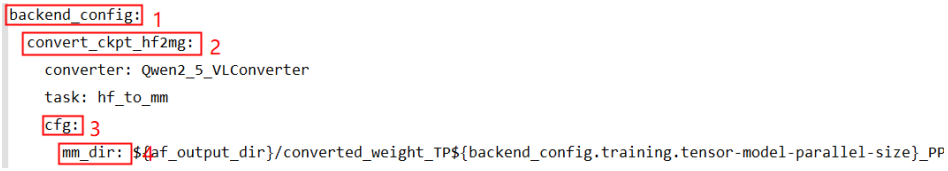
- Tensor parallelism, pipeline parallelism, and context parallelism: The value of TP x PP x CP must be exactly divisible by the number of NPUs (**word_size**).
- The value of TP x CP must be exactly divisible by **num_attention_heads** in the model parameters.
- MBS (**micro-batch-size**) and GBS (**global-batch-size**): The values of GBS and MBS must be exactly divisible by NPU/(TP x PP x CP).

5.1.12.1.5 MindSpeed-MM

This section describes the YAML configuration file and parameters for training. You can choose parameters as required.

Configuring Parameters in the YAML File

Modify the YAML file.



1. Set the dataset and model path.

Parameter	Example Value	Description
backend_config.data.dataset_param.preprocess_parameters.model_name_or_path	**/home/ma-user/AscendFactory/ckpts/hf_path/Qwen2.5-VL-7B-Instruct**	**[Mandatory]** Weight path before conversion. Change it based on the actual situation.
backend_config.data.dataset_param.basic_parameters.dataset_dir	**/home/ma-user/AscendFactory/data**	**[Mandatory]** Dataset path. Change it based on the actual situation.

2. Set the weight conversion.

Parameter	Example Value	Description
backend_config.convert_ckpt_hf2mg.cfg.mm_dir	**/home/ma-user/AscendFactory/ckpts/mm_path/converted_weight_TP${backend_config.training.tensor-model-parallel-size}_PP${backend_config.training.pipeline-model-parallel-size}**	**[Mandatory]** Directory for saving the converted file. Change it based on the actual situation.
backend_config.convert_ckpt_hf2mg.cfg.hf_config.hf_dir	**${backend_config.data.dataset_param.preprocess_parameters.model_name_or_path}**	Hugging Face weight directory.
backend_config.convert_ckpt_hf2mg.cfg.parallel_config.llm_pp_layers	**- 1** **- 10** **- 10** **- 7**	Number of llm layers assigned to each PU. It must match the value configured for **pipeline_num_layers** in **backend_config.model** during fine-tuning.
backend_config.convert_ckpt_hf2mg.cfg.parallel_config.vit_pp_layers	**- 32** **- 0** **- 0** **- 0**	Number of vit layers assigned to each PU. It must match the value configured for **pipeline_num_layers** in **backend_config.model** during fine-tuning.
backend_config.convert_ckpt_mg2hf.cfg.parallel_config.tp_size	**1**	TP parallelism count. Ensure that it matches the configuration used in training.
backend_config.convert_ckpt_mg2hf.cfg.save_hf_dir	*${af_output_dir}*/**ckpt_converted_mg2hf**	Directory for storing the converted Hugging Face model after MindSpeed-MM fine-tuning.
backend_config.convert_ckpt_mg2hf.cfg.parallel_config.llm_pp_layers	**-1** **- 10** **- 10** **- 7**	Number of llm layers assigned to each PU. It must match the value configured for **pipeline_num_layers** in **backend_config.model** during fine-tuning.

Parameter	Example Value	Description
backend_config.convert_ckpt_mg2hf.cfg.parallel_config.vit_pp_layers	**- 32** **- 0** **- 0** **- 0**	Number of vit layers assigned to each PU. It must match the value configured for **pipeline_num_layers** in **backend_config.model** during fine-tuning.
backend_config.convert_ckpt_mg2hf.cfg.parallel_config.tp_size	**1**	TP parallelism count. Ensure that it matches the configuration used in training.

3. Set the model saving, loading, and log information.

Parameter	Example Value	Description
backend_config.training.load	*${..convert_ckpt_hf2mg.cfg.mm_dir}*	Model loading path. Change it based on the actual situation.
backend_config.training.save	*${af_output_dir}*/**saved_checkpoints**	Model save path. Change it based on the actual situation.
backend_config.training.no-load-optim	**true**	Specifies whether to load the optimizer state. Set this parameter to **false** if loading is required.
backend_config.training.no-load-rng	**true**	Specifies whether to load the random number state. Set this parameter to **false** if loading is required.
backend_config.training.no-save-optim	**true**	Specifies whether to save the optimizer state. Set this parameter to **false** if loading is required.
backend_config.training.no-save-rng	**true**	Specifies whether to save the random number state. Set this parameter to **false** if loading is required.
backend_config.training.log-interval	**1**	Log interval.
backend_config.training.save-interval	**5000**	Save interval.

For other parameters not mentioned, see the **feature document**.

5.1.12.2 Modifying the Tokenizer File

Adjust the model's tokenizer file before starting the training, depending on your chosen framework.

For now, you only need to update the tokenizer for the GLM4-9B model and the InternVL2_5 series within the LlaMA-Factory framework. Edit the tokenizer file directly.

LlaMA-Factory

- **glm4-9b Model**

  Before the training starts, modify the tokenizer file **modeling_chatglm.py** in the glm4-9b model. The procedure is as follows:

  Go to the tokenizer (weight) directory and modify the content of the **modeling_chatglm.py** file.

  ```
  vim modeling_chatglm.py
  # Comment out the following content:
  # if attention_mask is not None
      # attention_mask = ~attention_mask
  ```

  Example

  ```
  class SdpaAttention(CoreAttention):
      def forward(self, query_layer, key_layer, value_layer, atte
          if attention_mask is None and query_layer.shape[2] == k
              context_layer = torch.nn.functional.scaled_dot_prod



          else:
              # if attention_mask is not None:
                  # attention_mask = ~attention_mask
  ```

- **InternVL2_5 series Models**

  Install the patch manually since the InternVL2_5 series models are not available in the Transformers repository. Follow these steps:

  - InternVL2_5-8B:

    ```
    # Download the model file using git clone. If the model file has been downloaded, skip this step.
    git clone https://huggingface.co/OpenGVLab/InternVL2_5-8B
    # Go to the weight directory and install the patch file. ${work_dir} indicates the working directory. Change it based on the site requirements.
    cd InternVL2_5-8B && git apply ${work_dir}/llm_train/AscendFactory/third-party/internvl25-8b.patch
    ```

  - InternVL2_5-38B:

    ```
    # Download the model file using git clone. If the model file has been downloaded, skip this step.
    git clone https://huggingface.co/OpenGVLab/nternVL2_5-38B
    # Go to the weight directory and install the patch file. ${work_dir} indicates the working directory. Change it based on the site requirements.
    cd InternVL2_5-38B && git apply ${work_dir}/llm_train/AscendFactory/third-party/internvl25-38b.patch
    ```

  - InternVL2_5-78B:

    ```
    # Download the model file using git clone. If the model file has been downloaded, skip this step.
    git clone https://huggingface.co/OpenGVLab/InternVL2_5-78B
    # Go to the weight directory and install the patch file. ${work_dir} indicates the working directory. Change it based on the site requirements.
    cd InternVL2_5-78B && git apply ${work_dir}/llm_train/AscendFactory/third-party/internvl25-8b.patch
    ```


5.1.12.3 Training Data Description

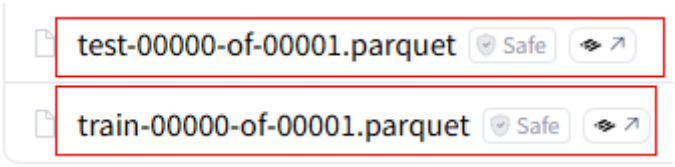
Supported Data Formats

The MindSpeed-LLM and Llama-Factory frameworks support the following common dataset formats:

- Alpaca
- ShareGPT
- MOSS (only for MindSpeed-LLM)

The download links of the sample datasets are as follows:

- Pre-training (MindSpeed-LLM): **train-00000-of-00001-a09b74b3ef9c3b56.parquet**. The data size is about 24 MB.
- Fine-tuning: **alpaca_gpt4_data.json**. The data size is 43.6 MB.
- Reinforcement learning (MindSpeed-RL): **deepscaler.json**. The data size is 21.5 MB.
- Reinforcement learning (VeRL). Datasets include training and test datasets. The following figure shows an example.



  - Large language model: **https://huggingface.co/datasets/openai/gsm8k/tree/main**
  - Multimodal model: **https://huggingface.co/datasets/hiyouga/geometry3k/tree/main**

Custom Data

1. LLaMA-Factory: dataset included in the code package. For details about how to modify the dataset, see **README_zh.md**.
2. MindSpeed-LLM: The data formats are as follows:
   - Pre-training data (MindSpeed-LLM): You can also prepare pre-training data by yourself. The data requirements are as follows:

     The data must be in standard .json format. Set **--json-key** to specify the columns for training. You can set **–json-key** to modify the dataset text field name. The default name is **text**. There are four columns in the Wikipedia dataset, including **id**, **url**, **title**, and **text**. You can specify **–json-key** to select the columns for training. For details, see **Processing a Pre-training Dataset**.

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

   - MOSS instruction fine-tuning data: This case also supports MOSS data in standard .json format, which includes multi-turn dialogues and instruction Q&A. The following is an example.


```
{
  "conversation_id": 1,
  "meta_instruction": "",
  "num_turns": 3,
  "chat": {
    "turn_1": {
      "Human": "<|Human|>: How do you guarantee adherence to proper security guidelines
during work?<eoh>\n",
      "Inner Thoughts": "<|Inner Thoughts|>: None<eot>\n",
      "Commands": "<|Commands|>: None<eoc>\n",
      "Tool Responses": "<|Results|>: None<eor>\n",
      "MOSS": "<|MOSS|>: To ensure xxx, the following suggestions are provided:\n\n1.
Understand related best practices.\n\nThese xxx environments.<eom>\n"
    },
    "turn_2": { ... },
    "turn_3": { ... },
  "category": "Brainstorming"
}
```

- Alpaca instruction fine-tuning data: For details, see the **Alpaca-style dataset**. The dataset contains the following fields:
  - **instruction**: describes the task to be performed by the model. Each instruction is unique.
  - **input**: optional context or input of the task. The **instruction** and **input** are combined into one single instruction: **instruction\ninput**.
  - **output**: answer to the generated instruction.
  - **system**: system prompt. It is used to set a scenario or provide guidelines for the entire dialogue.
  - **history**: a list that stores past conversations as pairs of user messages and model responses. This helps to maintain consistency and coherence in the dialogue.

```
[
  {
    "instruction": "Human instruction (required)",
    "input": "Human input (optional)",
    "output": "Model answer (required)",
    "system": "system prompt (optional)",
    "history": [
      ["First round instruction (optional)," "First round answer (optional)"],
      ["Second round instruction (optional)," "Second round answer (optional)"]
    ]
  }
]
```

- ShareGPT instruction fine-tuning data: The ShareGPT format comes from conversations between ChatGPT and its users. This dataset helps train dialogue systems. It gathers and organizes multiple exchanges that mimic real user-AI interactions. For details, see the **ShareGPT dataset**. The dataset contains the following fields:
  - **conversations**: Contains a series of dialog objects. Each object consists of the speaker (from) and speech content (value).
  - **from**: indicates the role of the dialogue, which can be **human** or **gpt**, indicating who said the sentence.
  - **value**: specific dialogue content.

- **system**: system prompt. It is used to set a scenario or provide guidelines for the entire dialogue.
- **tools**: describes the information about available external tools or functions. These tools may be used by the model to perform certain tasks or obtain more information.

```
[
  {
    "conversations": [
      {
        "from": "human",
        "value": "human instruction"
      },
      {
        "from": "function_call",
        "value": "tool parameter"
      },
      {
        "from": "observation",
        "value": "tool result"
      },
      {
        "from": "gpt",
        "value": "model answer"
      }
      ],
    "system": "system prompt (optional)",
    "tools": "tool description (optional)"
  }
]
```

3. MindSpeed-MM: The data formats are as follows:
   - Qwen2.5VL-3b/7b: Download the COCO2017 dataset by referring to **Preparing and Processing a Dataset**, and use scripts to convert the dataset. Alternatively, preprocess the custom dataset to the following format:

```
{
  "id": 1,
  "image": "image path",
  "conversations": [
     {"from": "human", "value": "human instructions"},
     {"from": "gpt", "value": "model answer"},
  ],
}
```

5.1.12.4 VeRL Data Processing Sample Script

The VeRL framework includes sample data processing scripts for both large language models (LLMs) and multimodal models. Choose the appropriate script according to your model type.

- **VeRL LLM GSM8k data processing**
- **Multimodal Model Geometry3K Data Processing**

LLM GSM8k Data Processing

```
import argparse
import os
import re
import datasets
from verl.utils.hdfs_io import copy, makedirs
def extract_solution(solution_str):
```



```
    solution = re.search("#### (\\-?[0-9\\.\\,]+)", solution_str)
    assert solution is not None
    final_solution = solution.group(0)
    final_solution = final_solution.split("#### ")[1].replace(",", "")
    return final_solution
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--local_dir", default="~/data/gsm8k")
    parser.add_argument("--hdfs_dir", default=None)
    args = parser.parse_args()
    data_source = "openai/gsm8k"
    dataset = datasets.load_dataset(xxx/xxx/xxx,"main")   # xxx/xxx/xxx is the dataset path.
    train_dataset = dataset["train"]
    test_dataset = dataset["test"]
    instruction_following = 'Let\'s think step by step and output the final answer after "####".'

    # add a row to each data item that represents a unique id
    def make_map_fn(split):
        def process_fn(example, idx):
            question_raw = example.pop("question")
            question = question_raw + " " + instruction_following
            answer_raw = example.pop("answer")
            solution = extract_solution(answer_raw)
            data = {
                "data_source": data_source,
                "prompt": [
                    {
                        "role": "user",
                        "content": question,
                    }
                ],
                "ability": "math",
                "reward_model": {"style": "rule", "ground_truth": solution},
                "extra_info": {
                    "split": split,
                    "index": idx,
                    "answer": answer_raw,
                    "question": question_raw,
                },
            }
            return data
        return process_fn
    train_dataset = train_dataset.map(function=make_map_fn("train"), with_indices=True)
    test_dataset = test_dataset.map(function=make_map_fn("test"), with_indices=True)

    local_dir = args.local_dir
    hdfs_dir = args.hdfs_dir

    train_dataset.to_parquet(os.path.join(local_dir, "train.parquet"))
    test_dataset.to_parquet(os.path.join(local_dir, "test.parquet"))

    if hdfs_dir is not None:
        makedirs(hdfs_dir)
        copy(src=local_dir, dst=hdfs_dir)
```

Multimodal Model Geometry3K Data Processing

```
import argparse
import os
import datasets
from verl.utils.hdfs_io import copy, makedirs

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--local_dir", default="~/data/geo3k")
    parser.add_argument("--hdfs_dir", default=None)
    args = parser.parse_args()
    data_source = "hiyouga/geometry3k"
    dataset = datasets.load_dataset(xxx/xxx/xxx) # xxx/xxx/xxx is the dataset path.
```



```
    train_dataset = dataset["train"]
    test_dataset = dataset["test"]
    instruction_following = (
        r"You FIRST think about the reasoning process as an internal monologue and then provide the final answer. "
        r"The reasoning process MUST BE enclosed within <think> </think> tags. The final answer MUST BE put in \boxed{}."
    )

    # add a row to each data item that represents a unique id
    def make_map_fn(split):
        def process_fn(example, idx):
            problem = example.pop("problem")
            prompt = problem + " " + instruction_following
            answer = example.pop("answer")
            images = example.pop("images")

            data = {
                "data_source": data_source,
                "prompt": [
                    {
                        "role": "user",
                        "content": prompt,
                    }
                ],
                "images": images,
                "ability": "math",
                "reward_model": {"style": "rule", "ground_truth": answer},
                "extra_info": {
                    "split": split,
                    "index": idx,
                    "answer": answer,
                    "question": problem,
                },
            }
            return data
        return process_fn
    train_dataset = train_dataset.map(function=make_map_fn("train"), with_indices=True, num_proc=8)
    test_dataset = test_dataset.map(function=make_map_fn("test"), with_indices=True, num_proc=8)

    local_dir = args.local_dir
    hdfs_dir = args.hdfs_dir

    train_dataset.to_parquet(os.path.join(local_dir, "train.parquet"))
    test_dataset.to_parquet(os.path.join(local_dir, "test.parquet"))

    if hdfs_dir is not None:
        makedirs(hdfs_dir)
        copy(src=local_dir, dst=hdfs_dir)
```

5.1.13 Common Error Causes and Solutions

The following links list common errors during training.

- **Out of Memory**
- **Incorrect NIC Name**
- **Timeout Error for Checkpoint Saving**
- **Installing Third-Party Dependency Packages Failed Using DockerFile or install.sh**
- **Timeout Occurs When the Llama-Factory Framework Preprocesses a Large Dataset**
- **Llama-Factory-based Training Suspended at a Certain Step**
- **Running setup.py in DockerFile or install.sh in the Llama-Factory Environment Failed**

- **MindSpeed-LLM Distilled Model Training Precision Issues**

Out of Memory

During training, the following out of memory error is reported:

```
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.
```

[Solution] The solutions for MindSpeed-LLM and Llama-Factory are different. Select a solution based on the framework.

- MindSpeed-LLM
  - Use the **npu-smi info** command to verify if any processes are using NPU resources, which could cause low memory during training. In this case, kill residual processes or wait for the resources to be released.
  - Adjust the values of **tensor-model-parallel-size** (TP) and **pipeline-model-parallel-size** (PP). Ensure that the value of TP x PP does not exceed the number of NPUs and be exactly divisible by it.
  - Adjust the values of **micro-batch-size** (MBS, minimum number of samples processed in a batch) and **global-batch-size** (GBS, number of samples processed in an iteration). Set **micro-batch-size** to **1** and ensure that the value of GBS/MBS can be exactly divided by the value of NPUs/(TP x PP).
  - Set **SEQ_LEN** to control the maximum sequence length for processing. A high value can cause out-of-memory errors.
  - Add the recomputation parameter to the **3_training.sh** file. The value of **recompute-num-layers** is the value of **num-layers** in the model network.

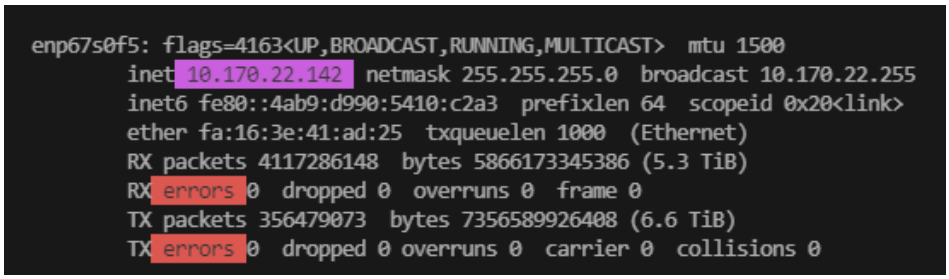
    ```
    --recompute-granularity full \
    --recompute-method block \
    --recompute-num-layers {NUM_LAYERS} \
    ```
- Llama-Factory
  - Adjust **per_device_train_batch_size** (minimum number of samples processed by a batch) to **1**.
  - Adjust DeepSpeed's zero level gradually.
    - - ZeRO-0: Data is distributed to different NPUs.
    - - ZeRO-1: Optimizer states are distributed to different NPUs.
    - - ZeRO-2: Optimizer states and gradients are distributed to different NPUs.
    - - ZeRO-2-Offload: Optimizer states are distributed to different NPUs and offload is enabled.
    - - ZeRO-3: Optimizer states, gradients, and model parameters are distributed to different NPUs.
    - - ZeRO-3-Offload: Optimizer states, gradients, and model parameters are distributed to different NPUs and offload is enabled.
  - Increase the number of PUs in ascending order.

Incorrect NIC Name

If the system displays a message indicating that the NIC name is incorrect or the communication times out when the training starts, run the **ifconfig** command to check whether the NIC name is correct.

For example, if the **ifconfig** command shows that the server's IP address uses the NIC named **enp67s0f5**, set this as the environment variable for the NIC name.

Figure 5-2 Incorrect NIC name



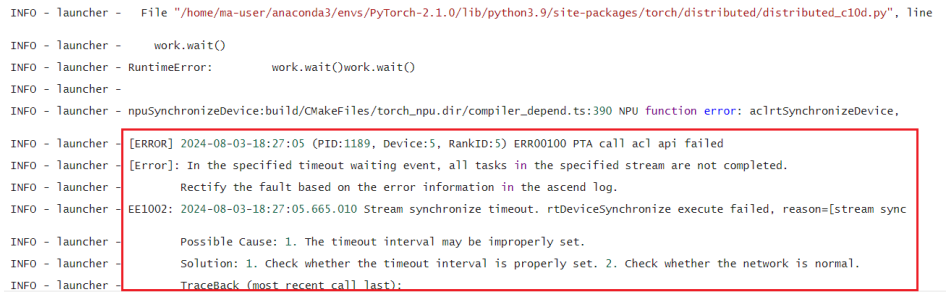

```
export GLOO_SOCKET_IFNAME=enp67s0f5  # Specify the network port name when multiple nodes communicate with each other using GLOO,
export TP_SOCKET_IFNAME=enp67s0f5  # Specify the network port name when multiple nodes communicate with each other using TP,
export HCCL_SOCKET_IFNAME=enp67s0f5  # Specify the network port name when multiple nodes communicate with each other using HCCL,
```

For details about environment variables, see **Distributed communication package - torch.distributed**.

Timeout Error for Checkpoint Saving

Once multi-node cluster training finishes, only certain nodes save the weights while others wait for communication. If this wait lasts over 36 minutes, a timeout error happens.

Figure 5-3 Error message



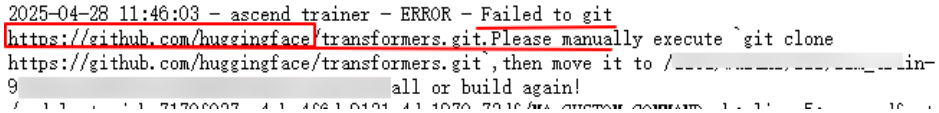
[Solution]

1. Check if the disk I/O bandwidth works properly to save the file in under 36 minutes. For one node, the largest file size allowed is 60 GB, but keep it below 40 GB for better performance. Saving the file within 36 minutes prevents timeout errors.
2. Ignore the error because the error does not affect the weights.

Installing Third-Party Dependency Packages Failed Using DockerFile or install.sh

[Symptom]

Downloading and installing third-party dependency packages such as Llama-Factory and MindSpeed-LLM in **AscendFactory/dependences.yaml** failed.



[Root Cause]

Fetching data from Git failed due to no internet connection.

[Solution]

Set up a proxy or use a server with internet access to download third-party dependencies listed in **AscendFactory/dependences.yaml**. Ensure the package names and versions match the *${save_name}* and *${version}* entries in the file. Move these packages to the **AscendFactory/third-party** folder, then rerun the Dockerfile or **install.sh** script.

Timeout Occurs When the Llama-Factory Framework Preprocesses a Large Dataset

[Root Cause]

The Llama-Factory framework initially handles data on PU 0 before moving sequentially through PUs 1 to 7. This sequential approach takes too long and often leads to timeouts.

[Solution]

- Solution A: Update the LLaMA-Factory barrier policy to process PUs 0 to 7 simultaneously instead of handling PU 0 first followed by PUs 1 to 7. Execute this command before starting the training:
  ```
  export DISABLE_MAIN_PROCESS_FIRST = True
  ```
- Solution B: Keep the default processing policy but set the training job's timeout to 2 hours. Execute this command before starting the training:
  ```
  export ACL_DEVICE_SYNC_TIMEOUT=7200
  ```
  Solution B is user-friendly but may time out after two hours with very large datasets. You can adjust the timeout setting as needed.

Llama-Factory-based Training Suspended at a Certain Step

[Symptom]

The multi-node training job suspends for two hours during a specific step, causing the job to time out.

[Root Cause]

The ascend_trace thread locks resources when capturing the call stack. The **dataloader_worker** process inherits this lock after being forked, causing it to suspend as it cannot acquire the lock.

[Solution]

Before starting a training job, load the **export ASCEND_COREDUMP_SIGNAL=none** environment variable to disable the stack trace.

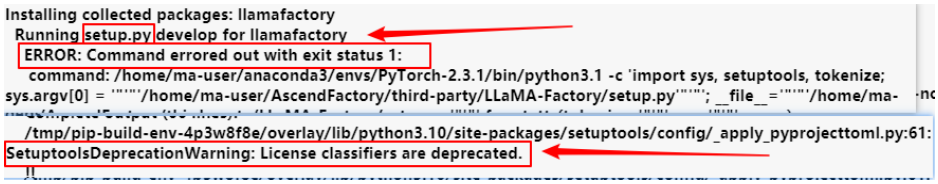
```
export ASCEND_COREDUMP_SIGNAL=none
```

Running setup.py in DockerFile or install.sh in the Llama-Factory Environment Failed

[Symptom]

Running **setup.py** in the Llama-Factory code directory failed. The error message "SetuptoolsDeprecationWarning: License classifiers are deprecated." is displayed.

Figure 5-4 Error message reported by setup.py

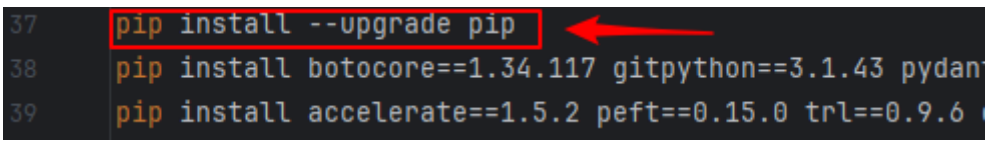


[Root Cause]

The pip dependency package version is too early, causing conflicts with other dependency packages.

[Solution]

1. Add the **pip install --upgrade pip** command to the **AscendFactory/install.sh** file.



2. Run the DockerFile or **install.sh** again.

MindSpeed-LLM Distilled Model Training Precision Issues

[Root cause]

The MindSpeed-LLM framework uses fixed parameter values that do not match the distilled model's setup, causing training precision issues.

[Solution]

Update the parameter values in **scripts_modellink/*{model}*/3_training.sh** to match those in the **config.json** file before starting training. Refer to the table below for details.

Table 5-14 Parameter values to be changed in the **3_training.sh** file

Distilled Model	Original Model	3_training.sh Parameter
DeepSeek-R1-Distill-Qwen-7B	qwen2.5-7b	--rotary-base 10000
DeepSeek-R1-Distill-Qwen-14B/32B	qwen2.5-14b/32b	--norm-epsilon 1e-5

6 Image Generation Model Training and Inference

6.1 Adapting Stable Diffusion for NPU Inference with Diffusers/ComfyUI and Lite Server (6.5.907)

This guide explains how to deploy the Stable Diffusion and HUNYUAN text-to-image models, using the Diffusers and ComfyUI frameworks on the ModelArts Lite Server. It also covers running these models with NPU-based inference.

Solution Overview

This solution describes how to use NPU compute resources to deploy the Diffusers and ComfyUI frameworks for Server-based inference. First, contact Huawei's enterprise technical support team to buy the required Server resources.

This solution is designed exclusively for enterprise users.

Resource Specifications

You are advised to use ModelArts Lite Server's Snt9B and Snt9B23 resources for inference deployment.

Table 6-1 Environment requirements

Name	Version
driver	25.2.1
PyTorch	pytorch_2.5.1

Obtaining Software Packages and Images

Table 6-2 Obtaining software packages and images

Category	Name	How to Obtain
Plug-in code package	**AscendCloud-6.5.907-*xxx*.zip** in the **AscendCloud-6.5.907** software package *xxx* in the file name indicates the timestamp. The timestamp is the actual release time of the package.	Download ModelArts 6.5.907.2 from **Support-E**. **NOTE** If the software information does not appear when opening the download link, you lack access permissions. Contact your company's Huawei technical support for assistance with downloading.
Snt9b base image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	Pull the image from SWR.
Snt9b23 base image	CN Southwest-Guiyang1: swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 CN-Hong Kong: swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129	Pull the image from SWR.

Table 6-3 Features

Kit	Model
Diffusers	SD1.5 SDXL SD3.5 HUNYUAN
ComfyUI	SD1.5 SDXL SD3.5

Step 1: Preparing the Environment

1. **Enable Lite Server resources** and obtain passwords. Verify SSH access to all servers. Confirm proper network connectivity between them.

   NOTE

   If no resource specifications are available when you purchase Server resources, contact Huawei Cloud technical support.

   If a container is used or shared by multiple users, you should restrict the container from accessing the OpenStack management address (169.254.169.254) to prevent host machine metadata acquisition. For details, see **Forbidding Containers to Obtain Host Machine Metadata**.

2. Check the environment.

   a. Log in to the server via SSH and check the NPU status. Obtain the NPU device information:

   ```
   npu-smi info
   ```

   If an error occurs, the NPU devices on the server may not be properly installed, or the NPU image may be mounted to another container. **Install the firmware and driver** or release the mounted NPUs.

   b. Check whether Docker is installed.

   ```
   docker -v   # Check whether Docker is installed.
   ```

   If Docker is not installed, run this command:

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

   c. Configure IP forwarding for intra-container network accesses. Run the following command to check the value of **net.ipv4.ip_forward**. Skip this step if the value is **1**.

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

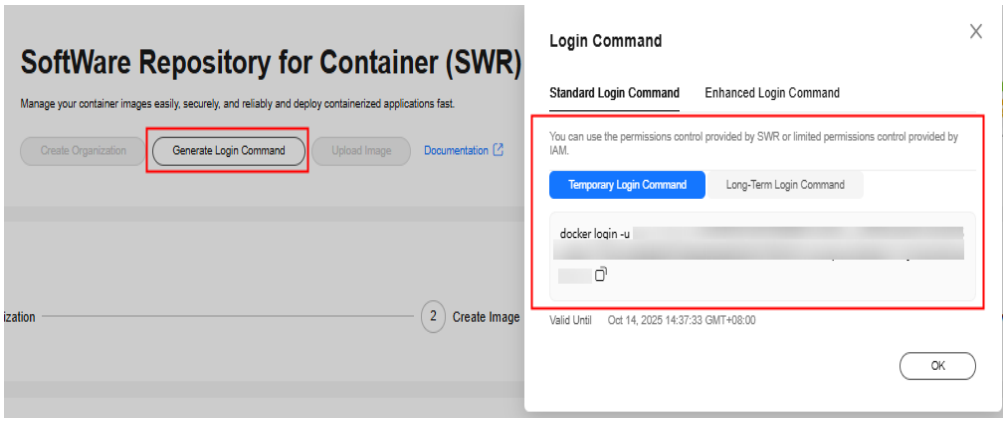
   If the value is not **1**, configure IP forwarding:

   ```
   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. Obtain the base image. Use official images to deploy inference services. For details about the image path *{image_url}*, see **Table 6-2**.

   ```
   docker pull {image_url}
   ```

   To log in to the SWR console, log in to the **SWR console** and obtain the login command by referring to the figure below.



Step 2: Starting the Container Image

Before starting the snt9b container image, modify the parameters in ${} based on the parameter description. Add or modify parameters as needed.

```
docker run -itd \
--name ${container_name} \
-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=host \
${image_name} bash
```

Parameter description:

- **--name** *${container_name}*: container name, which is used when you access the container. You can define a container name, for example, **comfyui**.
- --device=/dev/davinci3: Mounts **/dev/davinci3** of the host to **/dev/davinci3** of the container. You can run the **npu-smi info** command to view the idle PU number. After changing the davinci number, you can change the mounted PU.
- To start multi-PU inference, mount multiple PUs, for example, add **--device=/dev/davinci2**.
- *${image_name}* indicates the image name.
- **-p 8183:8183**: Enables a port. You can access the container service using **http://***host IP address*:**8183**. (If the port number is in use, change it to another one.)

1. Access the snt9b container. Replace *${container_name}* with the actual container name, for example, **comfyui**.
   ```
   docker exec -it ${container_name} bash
   ```
2. Start the snt9b23 container image. Before starting the container, modify the parameters in *${}* according to the parameter description. Add or modify parameters as needed.
   ```
   docker run -itd \
   --privileged \
   --name ${container_name} \
   -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=host \
${image_name} bash
```

Parameter description:

- **--name** *${container_name}*: container name, which is used when you access the container. You can define a container name, for example, **comfyui**.
- --device=/dev/davinci3: Mounts **/dev/davinci3** of the host to **/dev/davinci3** of the container. You can run the **npu-smi info** command to view the idle PU number. After changing the davinci number, you can change the mounted PU.
- To start multi-PU inference, mount multiple PUs, for example, add **--device=/dev/davinci2**.
- *${image_name}* indicates the image name.
- **-p 8183:8183**: Enables a port. You can access the container service using **http://***host IP address*:**8183**. (If the port number is in use, change it to another one.)

3. Access the snt9b23 container. Replace *${container_name}* with the actual container name, for example, **comfyui**.

```
docker exec -itu root ${container_name} bash
```

Step 3: Deploying Diffusers

Installing Dependencies and Model Packages

1. Run the command below to log in to Hugging Face and enter the token of your account to automatically download the model weights:

   After the login is successful, start the Diffusers inference script to automatically download the model weights.

   ```
   huggingface-cli login
   ```

   You can also manually download the model weights and upload them to the **/home/ma-user** directory of the container. The official download addresses (login required) are as follows:

   - Stable Diffusion 1.5: **https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5**
   - Stable Diffusion XL: **https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**
   - Stable Diffusion 3.5 Medium: **https://huggingface.co/stabilityai/stable-diffusion-3.5-medium/tree/main**
   - Stable Diffusion 3.5 Large: **https://huggingface.co/stabilityai/stable-diffusion-3.5-large/tree/main**
   - Hunyuan: **https://huggingface.co/Tencent-Hunyuan/HunyuanDiT-Diffusers/tree/main**

2. Install the plug-in code package.

a. Upload the **AscendCloud-AIGC-*xxx*.zip** plug-in code package to the **/home/ma-user/temp** directory of the container and decompress the package. For details about how to obtain the plug-in code package, see **Table 6-2**.

```
mkdir -p /home/ma-user/temp
cd /home/ma-user/temp
unzip AscendCloud-AIGC-*.zip # Decompress the package.
```

b. Decompress the AIGC package, go to the **/home/ma-user/temp/aigc_inference/torch_npu/utils/ascend_diffusers** directory, and install the **ascend_diffusers** package.

```
cd /home/ma-user/temp/aigc_inference/torch_npu/utils/ascend_diffusers
pip install -e .
```

c. Decompress the AIGC package, go to the **/home/ma-user/temp/aigc_inference/torch_npu/utils/AscendX-MM** directory, and install the **AscendX-MM** package.

```
cd /home/ma-user/temp/aigc_inference/torch_npu/utils/AscendX-MM
pip install -e .
```

Starting the Service

export MODEL_PATH='Path of the downloaded Hugging Face model', for example, **/home/ma-user/stable-diffusion-3.5-medium**. To let the system download the model automatically, do not add the **model_id** parameter.

```
cd /home/ma-user/temp/aigc_inference/torch_npu/diffusers/0.31.0/examples
```

The commands below start single-PU model inference. For details about the parameters, see the **Readme** file in the **/home/ma-user/temp/aigc_inference/torch_npu/diffusers** directory.

- Commands for starting Stable Diffusion 1.5 model inference:

```
pip install diffusers==0.30.2
python sd_inference_example.py --model_name sd15 --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 512 768 1024 --height 512 768 1024
```

- Commands for starting Stable Diffusion XL model inference:

```
pip install diffusers==0.30.2
python sd_inference_example.py --model_name sdxl --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 768 1024 --height 768 1024
```

- Commands for starting Stable Diffusion 3.5 model inference:

```
pip install diffusers==0.31.0
python sd_inference_example.py --model_name sd35 --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 28 --width 512 768 1024 --height 512 768 1024
```

- Commands for starting Hunyuan model inference:

```
pip install diffusers==0.30.2
export INF_NAN_MODE_FORCE_DISABLE=1
python sd_inference_example.py --model_name hunyuan --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 512 768 1024 --height 512 768 1024
```

Step 4: Deploying ComfyUI

Installing Dependencies and Model Packages

1. Download the ComfyUI software package.

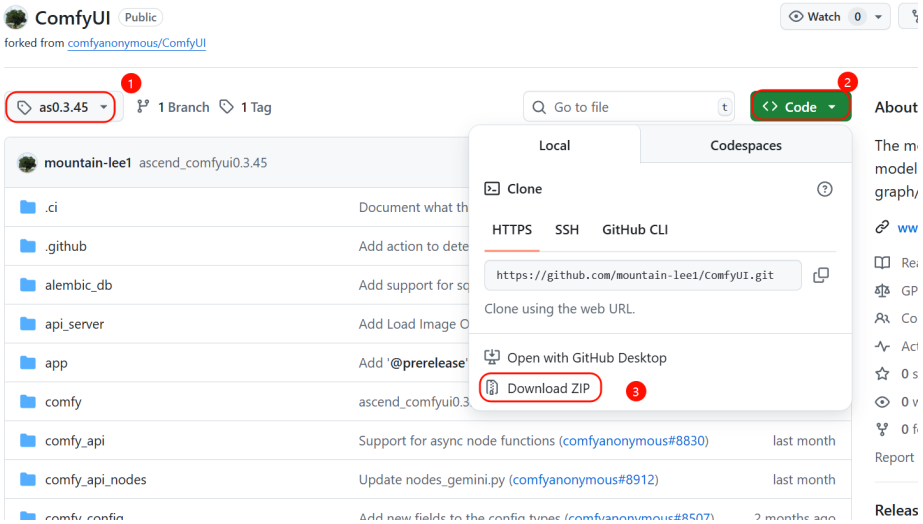
   Download the ComfyUI source code.

   ```
   git clone -b as0.3.45 https://github.com/mountain-lee1/ComfyUI.git
   cd ComfyUI
   ```

   If you cannot download the ComfyUI source code with the previous method, follow these steps: Download the source code to your PC and then upload it to the container, as shown in **Figure 6-1**.

a. Log in to https://github.com/mountain-lee1/ComfyUI, switch the tag to **as0.3.45**, click **Code**, and download the ComfyUI source code to the local PC by clicking **Download ZIP**.

Figure 6-1 Downloading the ComfyUI source code



NOTICE

Connect to the internet to access GitHub and download open-source software. Set up the network proxy if needed.

b. Upload the downloaded **ComfyUI-as0.3.45.zip** file to the **/home/ma-user/** directory of the container and decompress the package.

```
cd /home/ma-user/
unzip ComfyUI-as0.3.45.zip
cd ComfyUI-as0.3.45
```

2. Install the dependencies and change **torch** in **requirements.txt** to **torch==2.5.1**.

```
pip install -r requirements.txt
```

3. Download the model weights.

   sd1.5: Copy **v1-5-pruned-emaonly.safetensors** to the **ComfyUI/models/checkpoints** directory.

   https://huggingface.co/runwayml/stable-diffusion-v1-5/resolve/main/v1-5-pruned-emaonly.safetensors

   sdxl: Copy **sd_xl_base_1.0.safetensors** to the **ComfyUI/models/checkpoints** directory.

   https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/resolve/main/sd_xl_base_1.0.safetensors

   sd3.5: Copy **sd3.5_medium.safetensors** to the **ComfyUI/models/checkpoints** directory.

   https://www.modelscope.cn/models/cutemodel/comfyui-sd3.5-medium/file/view/master/sd3.5_medium.safetensors?status=2

   Copy **diffusion_pytorch_model.safetensors** to the **ComfyUI/models/vae** directory.

https://www.modelscope.cn/models/cutemodel/comfyui-sd3.5-medium/file/view/master/sd3.5vae.safetensors?status=2

In addition, you need to download three text_encoder-related models and copy them to the **ComfyUI/models/clip** directory.

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_l.safetensors

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/clip_g.safetensors

https://huggingface.co/Comfy-Org/stable-diffusion-3.5-fp8/blob/main/text_encoders/t5xxl_fp16.safetensors

You also need to download the workflow required for inference using the ComfyUI framework:

https://openart.ai/workflows/sneakyrobot/sd35-basic/CX6pkiT9lzJPlTpF9Cgu

4. Install the plug-in code package.
   a. Upload the **AscendCloud-AIGC-*xxx*.zip** plug-in code package to the **/home/ma-user/** directory of the container and decompress the package. For details about how to obtain the plug-in code package, see **Table 6-2**.
   ```
   cd /home/ma-user/
   unzip AscendCloud-AIGC-*.zip
   ```
   b. Go to the **ComfyUI/custom_nodes** directory and copy the **aigc_inference/torch_npu/comfyui/0.3.7/comfyui_ascend_node** folder extracted from the AIGC package to the **ComfyUI/custom_nodes** directory.
   ```
   cd ComfyUI/custom_nodes
   cp -r /home/ma-user/aigc_inference/torch_npu/comfyui/0.3.45/comfyui_ascend_node /home/ma-user/ComfyUI/custom_nodes
   ```
   c. Go to the **aigc_inference/torch_npu/utils/ascend_diffusers** directory and install the **ascend_diffusers** package.
   ```
   cd /home/ma-user/aigc_inference/torch_npu/utils/ascend_diffusers
   pip install -e .
   ```
   d. Go to the **aigc_inference/torch_npu/utils/AscendX-MM** directory and install the **AscendX-MM** package.
   ```
   cd /home/ma-user/aigc_inference/torch_npu/utils/AscendX-MM
   pip install -e .
   ```

Enabling the High-Performance Mode

Enable the high-performance mode for the Stable Diffusion model and start the service.

```
export CACHE_MODE=1
```

Starting the Service

1. Run the **ifconfig** command to obtain the container IP address. (If the **ifconfig** command is invalid, use the **ip addr** command or other methods to obtain the container IP address.)

Figure 6-2 Obtaining the snt9b container IP address

```
(PyTorch-2.1.0) [ma-user@57a5b008b312 custom_nodes]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.7  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:07  txqueuelen 0  (Ethernet)
        RX packets 6581  bytes 5071762 (4.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2651  bytes 148084 (144.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 6-3 Obtaining the snt9b23 container IP address

```
(PyTorch-2.5.1) [root@node-osmt ma-user]# ifconfig
docker0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:bcff:fea8:ea3a  prefixlen 64  scopeid 0x20<link>
        ether 02:42:bc:a8:ea:3a  txqueuelen 0  (Ethernet)
        RX packets 16760  bytes 733764 (716.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26074  bytes 140157610 (133.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp23s0f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 7.150.9.138  netmask 255.255.248.0  broadcast 7.150.15.255
        inet6 fe80::b121:b7fd:2789:b3c2  prefixlen 64  scopeid 0x20<link>
        ether fa:16:3e:ee:61:e2  txqueuelen 1000  (Ethernet)
        RX packets 811740360  bytes 1070741384739 (997.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 109827128  bytes 884500474152 (823.7 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4245147  bytes 544008187 (518.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4245147  bytes 544008187 (518.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth137729a: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::9846:c9ff:fe95:5502  prefixlen 64  scopeid 0x20<link>
        ether 9a:46:c9:95:55:02  txqueuelen 0  (Ethernet)
        RX packets 16760  bytes 968404 (945.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26117  bytes 140160624 (133.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. Go to the directory.

```
cd /home/ma-user/ComfyUI/custom_nodes
git config --global http.sslVerify false # Download nodes based on different workflows.
git clone https://github.com/ltdrdata/ComfyUI-Manager ComfyUI/custom_nodes/ComfyUI-Manager
# Download the ComfyUI manager for downloading nodes later.
cd /home/ma-user/ComfyUI
```

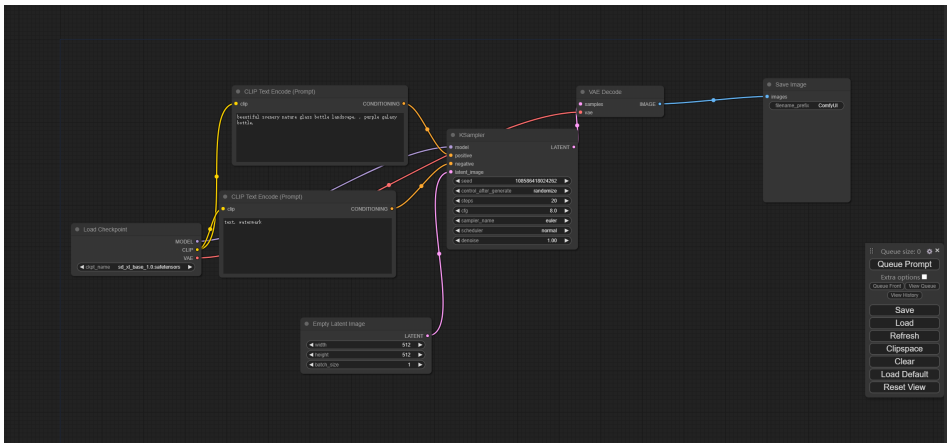
3. Start the service:

```
python main.py --port 8183 --listen 172.17.0.7 --force-fp16 --bf16-unet
```

4. Access the frontend page using **http://*{Host IP address}*:8183**.

a. Execute text-to-image.

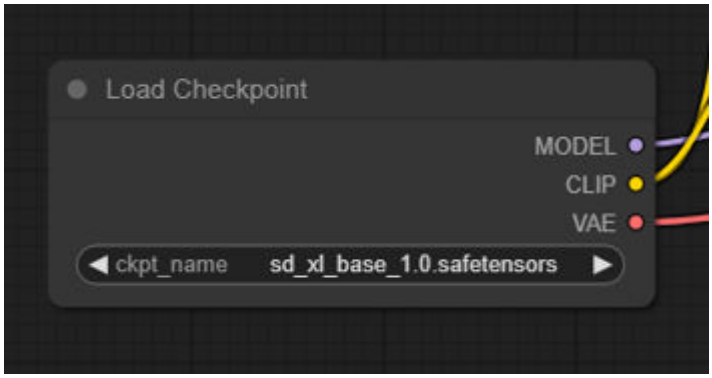
Figure 6-4 Accessing the frontend page



Besides the default workflow, you can load additional ones like the one from Stable Diffusion 3.5. If the workflow has uninstalled nodes, it might cause errors. To fix this, use the new ComfyUI manager to download and install the missing nodes. Once done, restart the ComfyUI service, preferably using the terminal start command. Then, choose the right model for each node and run the inference service.

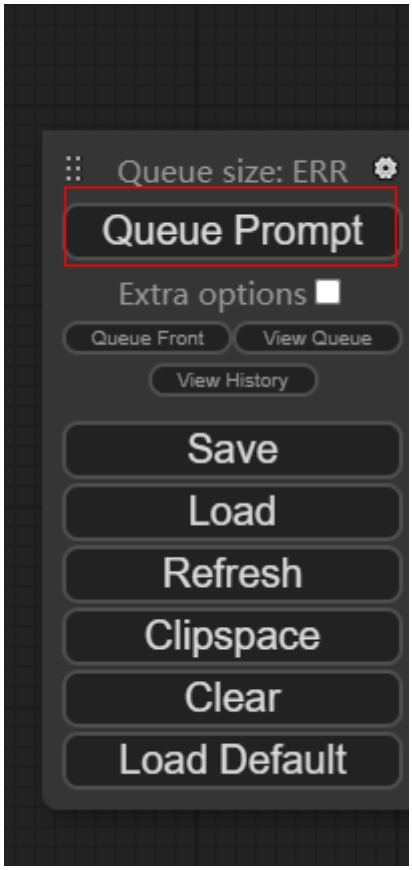
Plan the new NPU checkpoints based on the arrows in the preceding figure.

Figure 6-5 Planning checkpoints



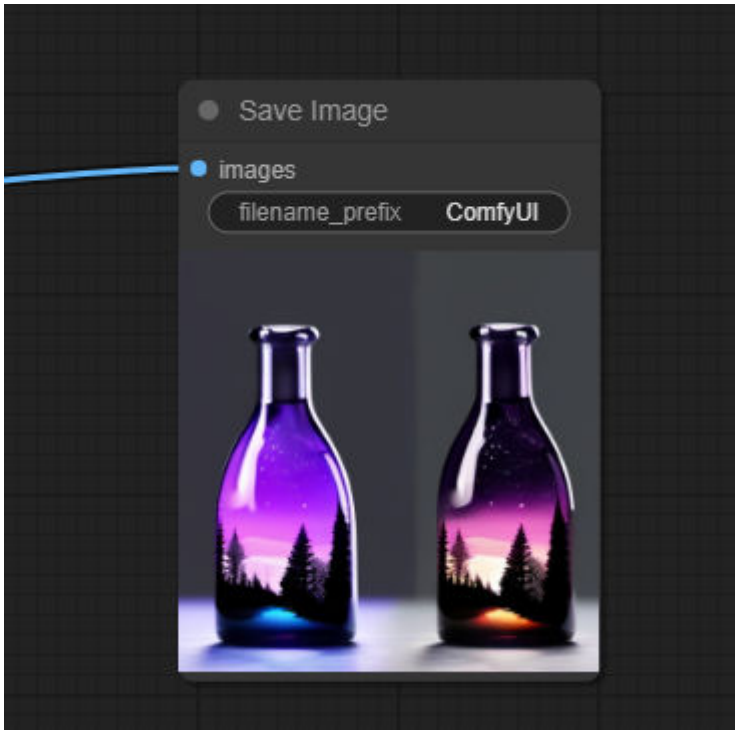
Select the weight file to be used from **ckpt_name** and click **Queue Prompt** to send it to the inference queue for inference.

Figure 6-6 Entering the inference queue



The following figure shows the success result.

Figure 6-7 Inference succeeded



6.2 Stable Diffusion XL Inference Guide Based on ModelArts Notebook (6.5.907)

Overview

This guide describes how to deploy text-to-image models like Stable Diffusion XL with the Diffusers framework on ModelArts notebook instances and run them on Huawei Cloud NPUs. Start by purchasing Server resources through Huawei's enterprise technical support team.

This solution is designed exclusively for enterprise users.

Resource Specifications

For inference deployment, it is advised to use the notebook and Snt9B resources in the CN-Hong Kong region.

Obtaining Software

Table 6-4 Software package and image

Category	Name	How to Obtain
Plugin code package	**AscendCloud-AIGC-6.5.907-*xxx*.zip** in the **AscendCloud-6.5.907** software package *xxx* in the file name indicates the timestamp. The timestamp is the actual release time of the package.	Download ModelArts 6.5.907.2 from **Support-E**. **NOTE** If the software information does not appear when opening the download link, you lack access permissions. Contact your company's Huawei technical support for assistance with downloading.
Notebook image	Snt9B: CN Southwest-Guiyang1 **swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129** Snt9B: CN-Hong Kong **swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129**	The image needs to be registered in ModelArts for selection when creating a notebook instance.

Features

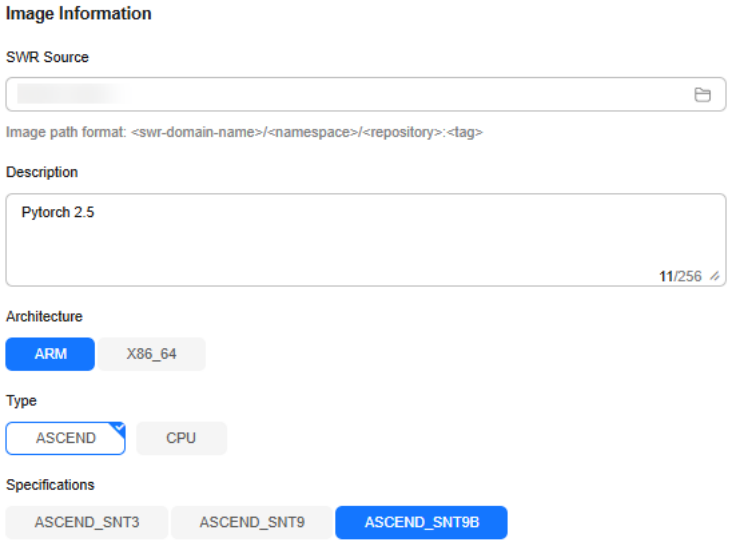
Table 6-5 Features

Kit	Model
Diffusers	SDXL

Step 1: Preparing the Environment

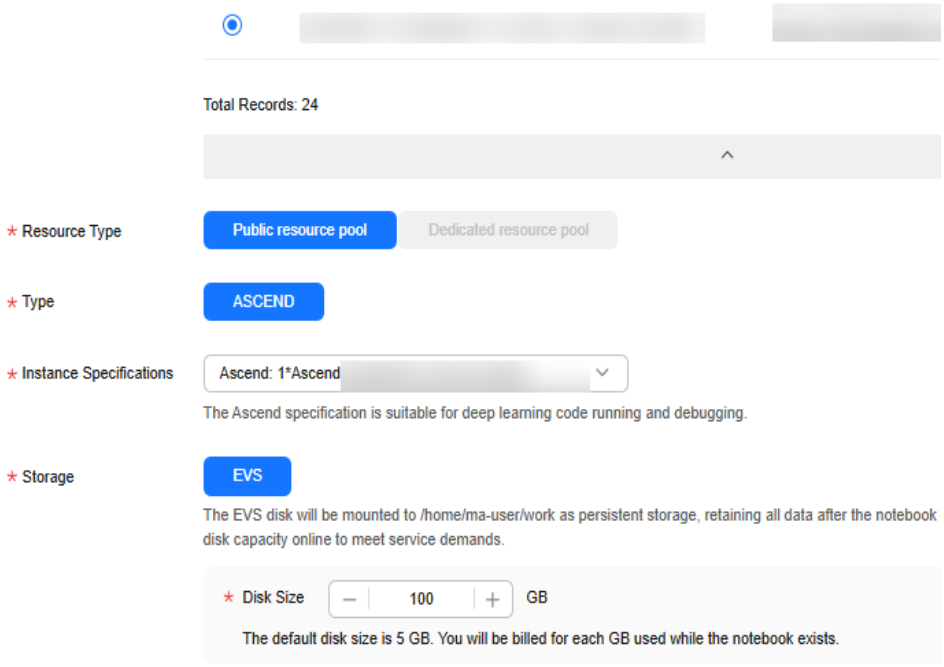
1. Log in to the **ModelArts console** and choose **Asset Management** > **Image Management** in the left navigation. In the upper right corner, click **Register** and select the notebook image specified in **Table 6-4**. Refer to the following figure for other options.

   Figure 6-8 Registering an image

2. Create a notebook instance using a custom image. For details, see **Creating a Notebook Instance (New Page)**. The default resource specification is 5 GB, but you can expand it as needed.

Figure 6-9 Creating a notebook instance



NOTE

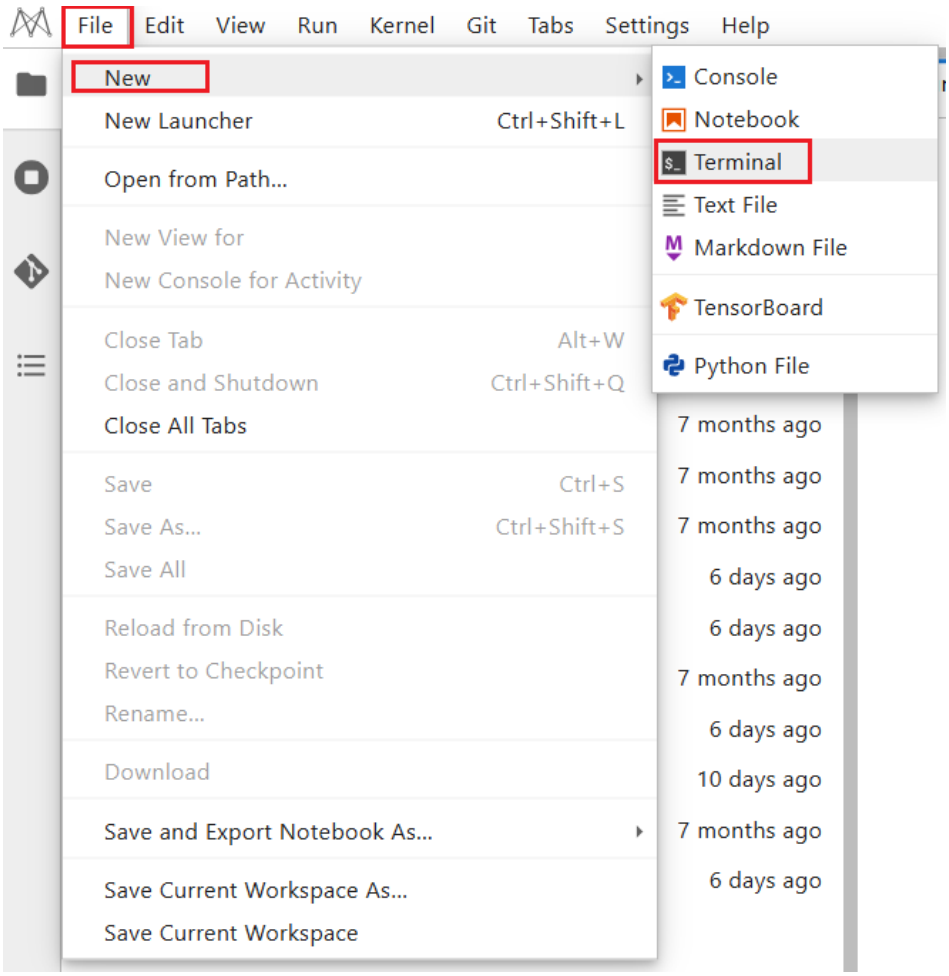
If no resource specifications are available when you purchase notebook resources, contact Huawei Cloud technical support.

3. Check the environment.

   Open the new notebook instance and start coding in the development environment.

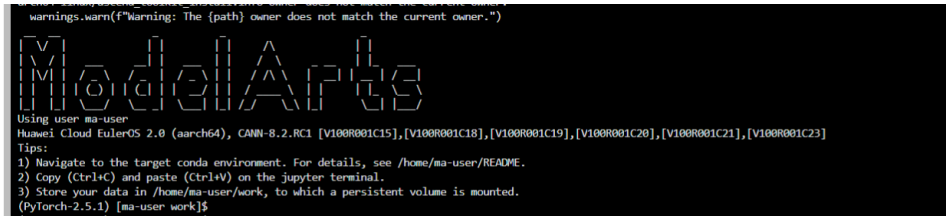
   ModelArts notebook instances are started by **ma-user** by default. After you access the instance, the default working directory is **/home/ma-user/work**.

Figure 6-10 Opening a notebook instance terminal



The figure below is an example working directory.

Figure 6-11 Working directory



Step 2: Deploying Diffusers

Installing Dependencies and Model Packages

1. Download the model weights to the **/home/ma-user/work** directory.

   Official website download links (login required):

   - Stable Diffusion XL: **https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0/tree/main**

   ModelScope download links:

- SDXL download link: **https://modelscope.cn/models/MusePublic/47_ckpt_SD_XL/summary?version=master**

2. Install the plugin code package.

   a. Upload the **AscendCloud-AIGC-*xxx*.zip** plug-in code package to the **/home/ma-user/work** directory of the container and decompress the package. For details about how to obtain the plugin code package, see **Table 6-4**. For details about how to upload a file, see **Uploading Files to JupyterLab**.

   ```
   cd /home/ma-user/work
   unzip AscendCloud-AIGC-*.zip # Decompress the package.
   ```

   b. Decompress the AIGC package, go to the **/home/ma-user/work/aigc_inference/torch_npu/utils/ascend_diffusers** directory, and install the **ascend_diffusers** package.

   ```
   cd /home/ma-user/work/aigc_inference/torch_npu/utils/ascend_diffusers
   pip install -e .
   ```

   c. Decompress the AIGC package, go to the **/home/ma-user/work/aigc_inference/torch_npu/utils/AscendX-MM** directory, and install the **AscendX-MM** package.

   ```
   cd /home/ma-user/work/aigc_inference/torch_npu/utils/AscendX-MM
   pip install -e .
   ```

Starting the Service

export MODEL_PATH='Path of the downloaded Hugging Face model', for example, **/home/ma-user/work/stable-diffusion-xl-base-1.0**. To let the system download the model automatically, do not add the **model_id** parameter.

```
export TASK_QUEUE_ENABLE=2
cd /home/ma-user/work/aigc_inference/torch_npu/diffusers/0.31.0/examples
```

The commands for starting single-PU model inference are as follows. The generated image is stored in the current directory.

- Commands for starting Stable Diffusion XL model inference:

  ```
  pip install diffusers==0.30.2
  python sd_inference_example.py --model_name sdxl --model_id ${MODEL_PATH} --prompt 'a dog' --num_inference_steps 20 --width 768 1024 --height 768 1024
  ```


7 Video Generation Model Training and Inference

7.1 Inference Guide for Wan Series Video Generation Models Adapted to NPU via Lite Server (6.5.907)

Solution Overview

This section describes how to use NPUs to perform text-to-video inference, image-to-video inference, and text-to-image inference using Wan2.1 and Wan2.2 video generation models in ModelArts Lite Server. To deploy the solution, contact Huawei technical support to purchase Server resources.

Wan series generation models are supported, including Wan2.1-T2V-14B-Diffusers, Wan2.1-T2V-1.3B-Diffusers, Wan2.1-I2V-14B-480P-Diffusers, Wan2.1-I2V-14B-720P-Diffusers, Wan2.2-T2V-A14B-Diffusers, and Wan2.2-I2V-A14B-Diffusers.

Resource Specifications

Use Snt9B or Snt9B23 single-node resources in the Lite Server environment.

Table 7-1 Snt9B23 environment requirements

Name	Version
Driver	25.2.1
PyTorch	pytorch_2.5.1

Table 7-2 Snt9B environment requirements

Name	Version
Driver	25.2.1

Name	Version
PyTorch	pytorch_2.5.1

Obtaining the Software Package and Images

Table 7-3 Software package and images to be obtained

Category	Name	How to Obtain
Plug-in code package	**AscendCloud-AIGC-6.5.907-*xxx*.zip** in the **AscendCloud-6.5.907-*xxx*.zip** software package **NOTE** *xxx* in the package name indicates the timestamp, which is subject to the actual package release time.	Download ModelArts 6.5.907.2 from **Support-E**. **NOTE** If the software information does not appear when opening the download link, you lack access permissions. Contact your company's Huawei technical support for assistance with downloading.
Base image	Snt9B23: CN North-Ulanqab1, CN East 2, and CN Southwest-Guiyang1 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 Snt9B: CN East 2 and CN Southwest-Guiyang1 swr.cn-southwest-2.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	Pull the image from SWR.

Category	Name	How to Obtain
Base image	Snt9B23: CN-Hong Kong swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b23-20250729103313-3a25129 Snt9B: CN-Hong Kong swr.ap-southeast-1.myhuaweicloud.com/atelier/pytorch_ascend:pytorch_2.5.1-cann_8.2.rc1-py_3.11-hce_2.0.2503-aarch64-snt9b-20250729103313-3a25129	Pull the image from SWR.

Constraints

- This document applies to ModelArts 6.5.907.2. Obtain the required software package and image by referring to **Table 7-3**. Follow the version mapping when using this document.
- Ensure that the container can access the Internet.

Step 1: Preparing the Environment

1. **Enable Lite Server resources** and obtain passwords. Verify SSH access to all servers. Confirm proper network connectivity between them.

   NOTE

   If a container is used or shared by multiple users, you should restrict the container from accessing the OpenStack management address (169.254.169.254) to prevent host machine metadata acquisition. For details, see **Forbidding Containers to Obtain Host Machine Metadata**.

2. Log in to the server via SSH and check the NPUs. Obtain the NPU device information:

   ```
   npu-smi info                 # Run this command on each instance node to view the NPU status.
   npu-smi info -l | grep Total    # Run this command on each instance node to view the total number of PUs.
   ```

   If an error occurs, the NPU devices on the server may not be properly installed, or the NPU image may be mounted to another container. **Install the firmware and driver** or release the mounted NPUs.

3. Check whether Docker is installed.

   ```
   docker -v   # Check whether Docker is installed.
   ```

   If Docker is not installed, run this command:

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

4. Configure IP forwarding for intra-container network accesses. Run the command below to check the value of **net.ipv4.ip_forward**. Skip this step if the value is **1**.

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


If the value is not **1**, configure IP forwarding:

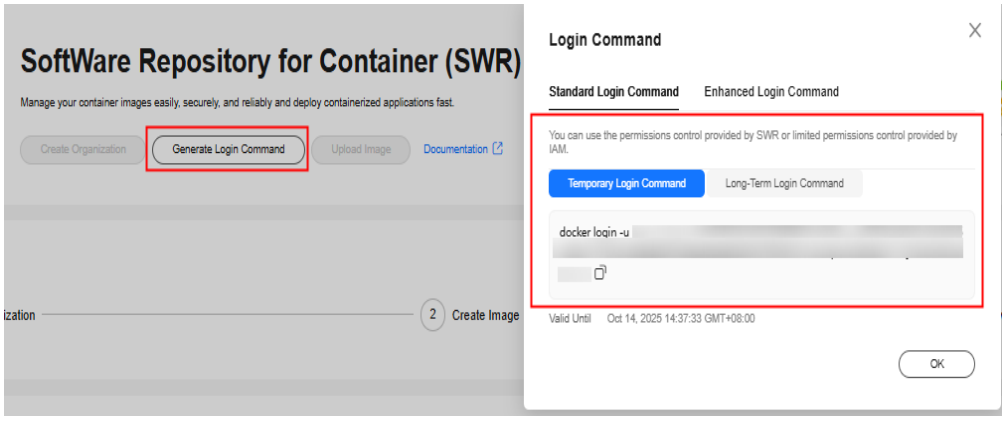
```
sed -i 's/net\.ipv4\.ip_forward=0/net\.ipv4\.ip_forward=1/g' /etc/sysctl.conf
sysctl -p | grep net.ipv4.ip_forward
```

Step 2: Obtaining the Base Image

Use official images to deploy inference services. For details about the image path *{image_url}*, see **Table 7-3**.

```
docker pull {image_url}
```

To log in to the SWR console, log in to the **SWR console** and obtain the login command by referring to the figure below.



Step 3: Starting the Container Image

1. Start the container image. Before starting the container, modify the parameters in *${}* according to the parameter description.

   Start the Snt9B23 container:

   ```
   export work_dir="Custom mounted working directory"
   export container_work_dir="Custom working directory mounted to the container"
   export container_name="Custom container name"
   export image_name="Image name or ID"
   // Start a container to run the image.
   docker run  -itd --net=host \
      --privileged \
      --device=/dev/davinci_manager \
      --device=/dev/devmm_svm \
      --device=/dev/hisi_hdc \
      --shm-size=256g \
      -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
   ```

   Start the Snt9B container:

   ```
   export work_dir="Custom mounted working directory"
   export container_work_dir="Custom working directory mounted to the container"
   export container_name="Custom container name"
   export image_name="Image name or ID"
   // Start a container to run the image.
   docker run  -itd --net=bridge \
      --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=256g \
-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
```

Parameters:

- **-v** *${work_dir}*:*${container_work_dir}*: host directory to be mounted to the container. The host and container use different file systems. *work_dir* indicates the working directory on the host. The directory stores files such as code and data required for the project. *container_work_dir* indicates the directory to be mounted to the container. The two paths can be the same.

  NOTE

  - The **/home/ma-user** directory cannot be mounted to the container. This directory is the home directory of user **ma-user**. If the container is mounted to **/home/ma-user**, the container conflicts with the base image when being started. As a result, the base image is unavailable.
  - Both the driver and npu-smi must be mounted to the container.
- **--name** *${container_name}*: container name, which is used when you access the container. You can define a container name.
- *${image_name}*: name of the base image of the corresponding model. For details, see **Table 7-3**.
- **--device=/dev/davinci0**: mount the corresponding PU to the container. If multiple PUs need to be mounted, add the configuration items one by one.

2. Access the container through the container name.

   Log in to the Snt9B23 as user **root**.

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

   For Snt9B, the **ma-user** user is used by default. All the subsequent operations are performed as user **ma-user**.

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

Step 4: Installing Dependencies and Software Packages

1. To use **git clone** and **git lfs** commands to download large models, see the following operations:

   a. Enter the URL below in the browser to download the **git-lfs** package and upload it to the **/home/ma-user** directory of the container.

   ```
   https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
   ```

   Alternatively, download **git-lfs** to the container for direct use.


```
cd /home/ma-user
wget https://github.com/git-lfs/git-lfs/releases/download/v3.2.0/git-lfs-linux-arm64-v3.2.0.tar.gz
```

b. Go to the container and run the **git-lfs** installation commands.

```
cd /home/ma-user
tar -zxvf git-lfs-linux-arm64-v3.2.0.tar.gz
cd git-lfs-3.2.0
sudo sh install.sh
```

c. Disable SSL verification for Git configuration.

```
git config --global http.sslVerify false
```

2. Install the AscendX_Video software package.

a. Upload the AscendX_Video software package **AscendCloud-AIGC-*.zip** to the **/home/ma-user** directory of the container. For details about how to obtain the package, see **Obtaining the Software Package and Images**.

b. Decompress the **AscendCloud-AIGC-*.zip** file, and run the following commands to install the Python dependencies:

```
cd /home/ma-user
unzip AscendCloud-AIGC-*.zip -d ./AscendCloud
cp -r /home/ma-user/AscendCloud/aigc_inference/torch_npu/ascendx_video ./
cd /home/ma-user/ascendx_video
pip install seal-*-linux_aarch64.whl
pip install check_device-*-linux_aarch64.whl
pip install ascendx_video-*-none-any.whl
```

c. Install the operator environment.

If the Snt9B23 machine is used, run the following command:

```
cd /home/ma-user/AscendCloud/opp/A3
```

If the Snt9B machine is used, run the following command:

```
cd /home/ma-user/AscendCloud/opp/A2
```

Install the operator:

```
unzip AscendCloud-OPP-*.zip
unzip AscendCloud-OPP-*-torch-2.5.1-py311-*.zip -d ./AscendCloud_OPP
cd AscendCloud_OPP
pip install *.whl
mkdir -p /home/ma-user/operate
bash ./ascend_cloud_ops_ascend_turbo-*_linux_aarch64.run --install-path=/home/ma-user/operate
bash ./ascend_cloud_ops_custom_opp-*_linux_aarch64_ascend910b_ascend910_93.run --install-path=/home/ma-user/operate
cd ..
unzip AscendCloud-OPS-ADV-*.zip -d ./AscendCloud_OPS-ADV
cd AscendCloud_OPS-ADV
bash ./CANN-custom_ops-*-linux.aarch64.run --install-path=/home/ma-user/operate
```

3. Initialize environment variables.

Note that the environment needs to be initialized each time you access the container.

```
source /home/ma-user/operate/AscendTurbo/set_env.bash
source /home/ma-user/operate/vendors/customize/bin/set_env.bash
source /home/ma-user/operate/vendors/customize_cloud/bin/set_env.bash
```

Step 5: Downloading Model Weights

Download the weight file to the container directory. The following lists the model addresses.

- Wan-AI/Wan2.1-T2V-14B-Diffusers: **https://huggingface.co/Wan-AI/Wan2.1-T2V-14B-Diffusers**

- Wan-AI/Wan2.1-T2V-1.3B-Diffusers: **https://huggingface.co/Wan-AI/Wan2.1-T2V-1.3B-Diffusers**
- Wan-AI/Wan2.1-I2V-14B-480P-Diffusers: **https://huggingface.co/Wan-AI/Wan2.1-I2V-14B-480P-Diffusers**
- Wan-AI/Wan2.1-I2V-14B-720P-Diffusers: **https://huggingface.co/Wan-AI/Wan2.1-I2V-14B-720P-Diffusers**
- Wan-AI/Wan2.2-T2V-A14B-Diffusers: **https://huggingface.co/Wan-AI/Wan2.2-T2V-A14B-Diffusers**
- Wan-AI/Wan2.2-I2V-A14B-Diffusers: **https://huggingface.co/Wan-AI/Wan2.2-I2V-A14B-Diffusers**

Save the weights to the **/home/ma-user/ascendx_video/weights** directory, for example:

```
weights
└──Wan-AI
   ├──Wan2.1-I2V-14B-480P-Diffusers
   ├──Wan2.1-I2V-14B-720P-Diffusers
   ├──Wan2.1-T2V-14B-Diffusers
   ├──Wan2.1-T2V-1.3B-Diffusers
   ├──Wan2.2-I2V-A14B-Diffusers
   └──Wan2.2-T2V-A14B-Diffusers
```

Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.1_14b_t2v_480p.sh**: 480P inference script of the Wan text-to-video model Wan2.1-T2V-14B.
- **infer_wan2.1_14b_t2v_720p.sh**: 720P inference script of the Wan text-to-video model Wan2.1-T2V-14B.
- **infer_wan2.1_1.3b_t2v.sh**: inference script of the Wan text-to-video model Wan2.1-T2V-1.3B.

Run the commands below to start the inference task. The following uses **infer_wan2.1_14b_t2v_480p.sh** as an example.

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_t2v_480p.sh
```

The following describes the parameters of the text-to-video inference script **infer_wan2.1_14b_t2v_480p.sh**. The parameters of scripts **infer_wan2.1_1.3b_t2v.sh** and **infer_wan2.1_14b_t2v_720p.sh** are similar to those of **infer_wan2.1_14b_t2v_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false

export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../infer.py \
        --model Wan2.1-T2V-14B \
```



```
    --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-T2V-14B-Diffusers" \
    --save_path ./output.mp4 \
    --num_inference_steps 50 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp \
    --vae_lightning \
    --turbo_mode faiz \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --seed 42 \
    --prompt "A young boy with short brown hair, dressed in a dark blue t-shirt and red pants, is seen playing a KAWAI upright piano with skill and concentration. The piano's glossy black surface reflects the room's lighting, and its white and black keys are arranged in a standard layout, indicating a scene of musical practice or learning. The boy's hands move over the keys, suggesting he is engaged in playing or practicing a piece." \
    --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting, picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete, redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

- **ASCEND_RT_VISIBLE_DEVICES**: ID of the used PU.
- **N_NPUS**: number of used PUs. You are advised to use eight PUs.
- **model**: supported inference model. Currently, Wan2.1-T2V-14B, Wan2.1-I2V-14B, Wan2.1-T2V-1.3B, Wan2.2-T2V-A14B, and Wan2.2-I2V-A14B are supported.
- **pretrained_model_name_or_path**: weight address of the corresponding model.
- **save_path**: path for storing the video generated during inference.
- **num_inference_steps**: number of inference steps.
- **frames, height, width**: dimensions of the generated video, including the number of frames, height, and width. Currently, 81 x 480 x 832, 121 x 480 x 832, 81 x 720 x 1280, and 121 x 720 x 1280 are supported.
- **prompt, negative_prompt**: positive and negative prompts for generating a video.
- **sp**: sequence parallelism parameter. It is recommended that the value be the same as the number of inference PUs.
- **fsdp**: data parallelism. **None**, **all**, **text_encoder**, and **transformer** are supported. The default value is **None**, indicating that the function is disabled. If this function is enabled, the default value is **all**, indicating that parallelism is enabled for **text_encoder** and **transformer**. If this parameter is set to **text_encoder** or **transformer**, parallelism is enabled only for the specified module.
- **vae_lightning**: VAE acceleration. This parameter is supported only in the multi-PU scenario. If this parameter is not set, VAE acceleration is disabled. Enabling this function can improve VAE performance.
- **turbo_mode**: acceleration mode. **default** and **faiz** are supported. The default value is **default**, indicating that the function is disabled. **faiz** is recommended for high performance. If this parameter is not set, the acceleration mode is disabled. Enabling this function can accelerate video inference, but slightly affects the accuracy.
- **atten_a8w8**: atten quantization acceleration. Set this parameter for high performance. If this parameter is not set, atten quantization acceleration is

disabled. Enabling this function can accelerate video inference, but slightly affects the accuracy.

- **matmul_a8w8**: matmul quantization acceleration. Set this parameter for high performance. If this parameter is not set, matmul quantization acceleration is disabled. Enabling this function can accelerate video inference, but slightly affects the accuracy.
- **rope_fused**: rotary position encoding fusion operator. Set this parameter for high performance. If this parameter is not set, the fusion operator is disabled. Enabling this function can accelerate video inference, but slightly affects the accuracy.
- **seed**: Random number seed. The default value is **42**, which affects the effect of the generated image.

After the inference task is complete, the generated video file **output.mp4** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

Step 7: Performing Inference Using the Wan2.1 Image-to-Video Model

Before starting inference using the image-to-video model, download the sample image and save it to the **/home/ma-user/ascendx_video/scripts** directory.

Figure 7-1 Example

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.1_14b_i2v_480p.sh**: 480P inference script of the Wan2.1-I2V-14B image-to-video model.
- **infer_wan2.1_14b_i2v_720p.sh**: 720P inference script of the Wan2.1-I2V-14B image-to-video model.

Run the commands below to start the inference task. The following uses **infer_wan2.1_14b_i2v_480p.sh** as an example.


```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_i2v_480p.sh
```

The following describes the parameters of the image-to-video inference script **infer_wan2.1_14b_i2v_480p.sh**. The parameters of script **infer_wan2.1_14b_i2v_720p.sh** are similar to those of **infer_wan2.1_14b_i2v_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
        --model Wan2.1-I2V-14B \
        --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-I2V-14B-480P-Diffusers" \
        --task_type i2v \
        --i2v_image_path ./astronaut.jpg \
        --save_path ./output.mp4 \
        --num_inference_steps 40 \
        --width 832 \
        --height 480 \
        --frames 81 \
        --sp $N_NPUS \
        --fsdp \
        --vae_lightning \
        --turbo_mode faiz \
        --atten_a8w8 \
        --matmul_a8w8 \
        --rope_fused \
        --seed 42 \
        --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth
of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera
shot." \
        --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting,
picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete,
redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger
fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

- **task_type**: inference task type. The value can be **t2v** (text-to-video), **i2v** (image-to-video), or **t2i** (text-to-image). The default value is **i2v**.
- **i2v_image_path**: path of the image used for video generation.
- For other parameters, use the same settings as those of **infer_wan_14b_t2v_480p.sh**. For details, see **Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model**.

After the inference task is complete, the generated video file **output.mp4** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

Step 8: Performing Inference Using the Wan2.1 Text-to-Image Model

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.1_14b_t2i_480p.sh**: 480P inference script of the Wan2.1-T2V-14B text-to-image model.

- **infer_wan2.1_14b_t2i_720p.sh**: 720P inference script of the Wan2.1-T2V-14B text-to-image model.

Run the commands below to start the inference task. The following uses **infer_wan2.1_14b_t2i_480p.sh** as an example.

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.1_14b_t2i_480p.sh
```

The following describes the parameters of the text-to-image inference script **infer_wan2.1_14b_t2i_480p.sh**. The parameters of script **infer_wan2.1_14b_t2i_720p.sh** are similar to those of **infer_wan2.1_14b_t2i_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0
N_NPUS=1
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
        --model Wan2.1-T2V-14B \
        --pretrained_model_name_or_path "../weights/Wan-AI/Wan2.1-T2V-14B-Diffusers" \
        --task_type t2i \
        --save_path ./output.png \
        --num_inference_steps 40 \
        --width 832 \
        --height 480 \
        --frames 1 \
        --atten_a8w8 \
        --matmul_a8w8 \
        --rope_fused \
        --seed 42 \
        --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth
of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera
shot." \
        --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting,
picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete,
redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger
fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

The parameters are the same as those of **infer_wan_14b_t2v.sh**. For details, see **Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model**.

After the inference task is complete, the generated image file **output.png** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

Step 9: Performing Inference Using the Wan2.2 Text-to-Video Model

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.2_14b_t2v_480p.sh**: 480P inference script of the Wan text-to-video model Wan2.2-T2V-A14B-Diffusers.
- **infer_wan2.2_14b_t2v_720p.sh**: 720P inference script of the Wan text-to-video model Wan2.2-T2V-A14B-Diffusers.

Run the commands below to start the inference task. The following uses **infer_wan2.2_14b_t2v_480p.sh** as an example.

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_t2v_480p.sh
```

The following describes the parameters of the text-to-video inference script **infer_wan2.2_14b_t2v_480p.sh**. The parameters of script **infer_wan2.2_14b_t2v_720p.sh** are similar to those of **infer_wan2.2_14b_t2v_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../infer.py \
    --model Wan2.2-T2V-A14B \
    --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-T2V-A14B-Diffusers \
    --task_type t2v \
    --save_path ./output.mp4 \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp text_encoder \
    --vae_lightning \
    --inf_vram_blocks_num 1 \
    --vae_lightning \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --guidance_scale 3.0 \
    --guidance_scale_2 4.0 \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera shot." \
    --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting, picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete, redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

- **inf_vram_blocks_num**: GPU memory optimization. Currently, only **1** is supported. If this function is enabled, the **fsdp text_encoder** parameter is required.
- **guidance_scale**: no classifier guidance for the transformer. Set this parameter based on the corresponding model.
- **guidance_scale_2**: no classifier guidance for transformer_2 of Wan2.2. Set this parameter based on the corresponding model.

The parameters are the same as those of **infer_wan_14b_t2v.sh**. For details, see **Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model**.

After the inference task is complete, the generated video file **output.mp4** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

Step 10: Performing Inference Using the Wan2.2 Image-to-Video Model

Before starting inference using the image-to-video model, download the sample image and save it to the **/home/ma-user/ascendx_video/scripts** directory.

Figure 7-2 Example

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.2_14b_i2v_480p.sh**: 480P inference script of the Wan image-to-video model Wan2.2-I2V-A14B-Diffusers.
- **infer_wan2.2_14b_i2v_720p.sh**: 720P inference script of the Wan image-to-video model Wan2.2-I2V-A14B-Diffusers.

Run the commands below to start the inference task. The following uses **infer_wan2.2_14b_i2v_480p.sh** as an example.

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_i2v_480p.sh
```

The following describes the parameters of the text-to-video inference script **infer_wan2.2_14b_i2v_480p.sh**. The parameters of script **infer_wan2.2_14b_i2v_720p.sh** are similar to those of **infer_wan2.2_14b_i2v_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1,2,3,4,5,6,7
N_NPUS=8
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../
infer.py \
        --model Wan2.2-I2V-A14B \
        --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-I2V-A14B-Diffusers \
        --task_type i2v \
```



```
    --i2v_image_path ./astronaut.jpg \
    --save_path ./output.mp4 \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 81 \
    --sp $N_NPUS \
    --fsdp \
    --vae_lightning \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --guidance_scale 3.5 \
    --guidance_scale_2 3.5 \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera shot." \
    --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting, picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete, redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

The parameters are the same as those of **infer_wan_14b_t2v.sh**. For details, see **Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model**.

After the inference task is complete, the generated video file **output.mp4** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

Step 11: Performing Inference Using the Wan2.2 Text-to-Image Model

The following scripts are stored in the **/home/ma-user/ascendx_video/scripts/** directory:

- **infer_wan2.2_14b_t2i_480p.sh**: 480P inference script of the Wan text-to-image model Wan2.2-T2V-A14B-Diffusers.
- **infer_wan2.2_14b_t2i_720p.sh**: 720P inference script of the Wan text-to-image model Wan2.2-T2V-A14B-Diffusers.

Run the commands below to start the inference task. The following uses **infer_wan2.2_14b_t2i_480p.sh** as an example.

```
cd /home/ma-user/ascendx_video/scripts/
bash infer_wan2.2_14b_t2i_480p.sh
```

The following describes the parameters of the text-to-image inference script **infer_wan2.2_14b_t2i_480p.sh**. The parameters of script **infer_wan2.2_14b_t2i_720p.sh** are similar to those of **infer_wan2.2_14b_t2i_480p.sh**.

```
export MASTER_ADDR=127.0.0.1
export MASTER_PORT=29505

export PYTORCH_NPU_ALLOC_CONF=expandable_segments:True
export MEMORY_FRAGMENTATION=1
export COMBINED_ENABLE=1
export TASK_QUEUE_ENABLE=2
export TOKENIZERS_PARALLELISM=false
export ASCEND_RT_VISIBLE_DEVICES=0,1
N_NPUS=2
torchrun --nproc_per_node=$N_NPUS --master_addr $MASTER_ADDR --master_port $MASTER_PORT ../infer.py \
    --model Wan2.2-T2V-A14B \
```



```
    --pretrained_model_name_or_path ../weights/Wan-AI/Wan2.2-T2V-A14B-Diffusers \
    --task_type t2i \
    --save_path ./output.png \
    --num_inference_steps 40 \
    --width 832 \
    --height 480 \
    --frames 1 \
    --atten_a8w8 \
    --matmul_a8w8 \
    --rope_fused \
    --guidance_scale 3.0 \
    --guidance_scale_2 4.0 \
    --seed 42 \
    --prompt "An astronaut hatching from an egg, on the surface of the moon, the darkness and depth of space realised in the background. High quality, ultrarealistic detail and breath-taking movie-like camera shot." \
    --negative_prompt "vivid tone, overexposure, static, blurred details, subtitle, style, work, painting, picture, static, overall gray, worst quality, low quality, JPEG compression residue, ugly, incomplete, redundant fingers, poorly drawn hands, poorly drawn face, deformed, disfigured, deformed limbs, finger fusion, static picture, messy background, three legs, many people in the background, walking backward"
```

The parameters are the same as those of **infer_wan_14b_t2v.sh**. For details, see **Step 6: Performing Inference Using the Wan2.1 Text-to-Video Model**.

After the inference task is complete, the generated image file **output.png** is stored in the **save_path** directory, and the script is stored in the **/home/ma-user/ascendx_video/scripts** directory by default. From there, view the inference result.

8 Permissions Management

8.1 Basic Concepts

ModelArts allows you to configure fine-grained permissions for refined management of resources and permissions. This is commonly used by large enterprises, but it is complex for individual users. It is recommended that individual users configure permissions for using ModelArts by referring to **Assigning Permissions to Individual Users for Using ModelArts**.

NOTE

If you meet any of the following conditions, read this document.

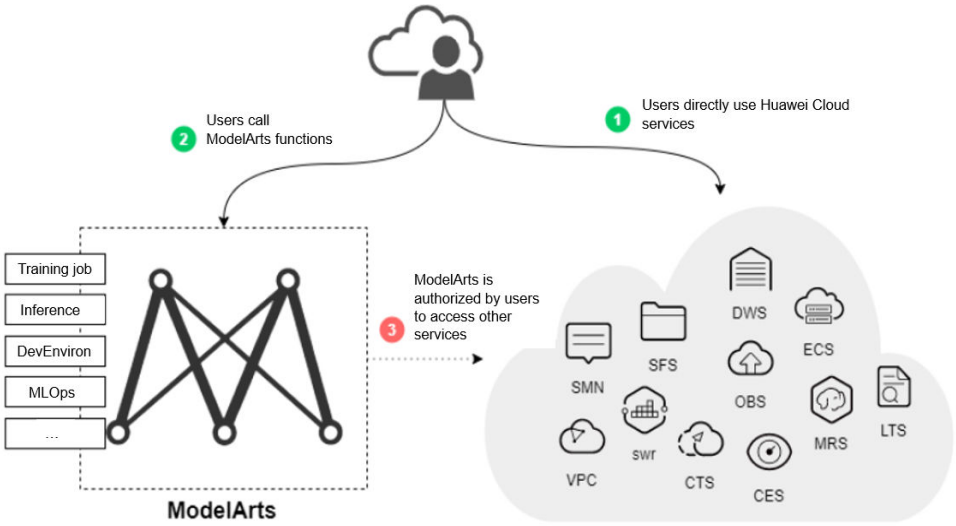
- You are an enterprise user with multiple departments. You need to specify users in different departments to access their dedicated resources. And there should be multiple roles, including administrator, algorithm developer, and application O&M engineer.
- You are an enterprise user and aim to restrict different roles to use specific functions. Logically, there should be multiple isolated environments, such as the development environment, pre-production environment, and production environment. You also want the operation permissions to vary depending on the environment. Or you need to control permissions of specific IAM user or user group.
- You are an individual user, and you have created multiple IAM users. You need to assign different ModelArts permissions to different IAM users.
- You need to understand the concepts and operations of ModelArts permissions management.

ModelArts uses Identity and Access Management (IAM) for most permissions management functions. Before reading below, learn about **Basic Concepts**. This helps you better understand this document.

To implement fine-grained permissions management, ModelArts provides permission control, agency authorization, and workspace. The following describes the details.

ModelArts Permissions and Agencies

Figure 8-1 Permissions management



Like other services, ModelArts functions are controlled by IAM permissions. For example, if you as an IAM user need to create a training job on ModelArts, you must have the **modelarts:trainJob:create** permission. For details about how to assign permissions to a user (you need to add the user to a user group and then assign permissions to the user group), see **Permissions Management**.

ModelArts must access other services for AI computing. For example, ModelArts must access OBS to read your data for training. For security purposes, ModelArts must be authorized to access other cloud services. This is agency authorization.

The following summarizes permissions management:

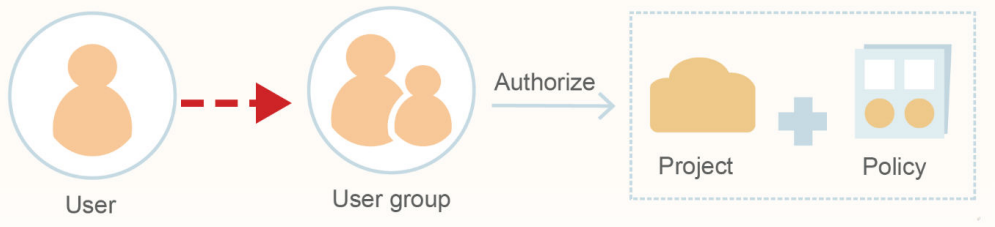
- Your access to any cloud service is controlled through IAM. You must have the permissions of the cloud service. (The required service permissions vary depending on the functions you use.)
- To use ModelArts functions, you need to grant permissions through IAM.
- ModelArts must be authorized by you to access other cloud services for AI computing.

ModelArts Permissions Management

By default, new IAM users do not have any permissions assigned. You need to add the user to a user group and grant the user group with policies, so that the users in the group can inherit the permissions. After authorization, users can perform operations on ModelArts based on permissions.

CAUTION

- ModelArts is a project-level service deployed and accessed in specific physical regions. When you authorize an agency, you can set the scope for the permissions you select to all resources, enterprises projects, or region-specific projects. If you specify region-specific projects, the selected permissions will be applied to resources in these projects.
- ModelArts supports enterprise projects. You can specify an enterprise project when selecting the authorization scope. For details, see **Creating a User Group and Assigning Permissions**.



When assigning permissions to a user group, IAM does not directly assign specific permissions to the user group. Instead, IAM needs to add the permissions to a policy and then assign the policy to the user group. To facilitate user permissions management, each cloud service provides some preset policies for you to directly use. If the preset policies cannot meet your requirements of fine-grained permissions management, you can customize policies.

Table 8-1 lists all the preset system-defined policies supported by ModelArts.

Table 8-1 System-defined policies supported by ModelArts

Policy	Description	Type
ModelArts FullAccess	Administrator permissions for ModelArts. Users granted these permissions can operate and use ModelArts.	System-defined policy
ModelArts CommonOperations	Common user permissions for ModelArts. Users granted these permissions can operate and use ModelArts, but cannot manage dedicated resource pools.	System-defined policy
ModelArts Dependency Access	Permissions on dependent services for ModelArts	System-defined policy

Generally, ModelArts FullAccess is assigned only to administrators. If fine-grained management is not required, assigning ModelArts CommonOperations to all users will meet the development requirements of most small teams. If you want to customize policies for fine-grained permissions management, see **IAM**.

NOTE

When you assign ModelArts permissions to a user, the system does not automatically assign the permissions of other services to the user. This ensures security and prevents unexpected unauthorized operations. In this case, however, you must separately assign permissions of different services to users so that they can perform some ModelArts operations.

For example, if an IAM user needs to use OBS data for training and the ModelArts training permission has been configured for the IAM user, the IAM user still needs to be assigned with the OBS read, write, and list permissions. The OBS list permission allows you to select the training data path on ModelArts. The read permission is used to preview data and read data for training. The write permission is used to save training results and logs.

- For individual users or small organizations, it is a good practice to configure the **Tenant Administrator** policy that applies to global services for IAM users. In this way, IAM users can obtain all user permissions except IAM. However, this may cause security issues. (For an individual user, its default IAM user belongs to the **admin** user group and has the **Tenant Administrator** permission.)
- If you want to restrict user operations, configure the minimum permissions of OBS for ModelArts users. For details, see **OBS Permissions Management**. For details about fine-grained permissions management of other cloud services, see the corresponding cloud service documents.

ModelArts Agency Authorization

ModelArts must be authorized by users to access other cloud services for AI computing. In the IAM permission system, such authorization is performed through agencies.

For details about the basic concepts and operations of agencies, see **Cloud Service Delegation**.

To simplify agency authorization, ModelArts supports automatic agency authorization configuration. You only need to configure an agency for yourself or specified users on the **Permission Management** page of the ModelArts console.

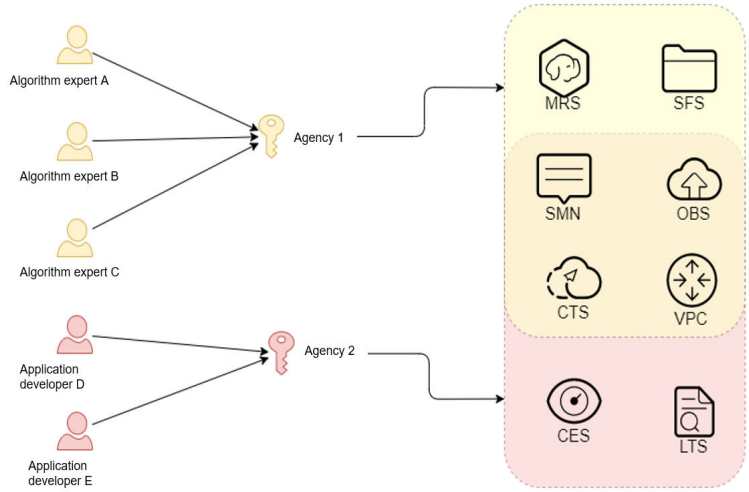
NOTE

- Only users with the IAM agency management permission can perform this operation. Generally, members in the IAM admin user group have this permission.
- ModelArts agency authorization is region-specific, which means that you must perform agency authorization in each region you use.

On the **Permission Management** page of the ModelArts console, after you click **Add Authorization**, you can configure an agency for a specific user or all users. Generally, an agency named **modelarts_agency_**<*Username*>*_Random ID* is created by default. In the **Permissions** area, you can select the preset permission configuration or select the required policies. If both options cannot meet your requirements, you can create an agency on the IAM management page (you need to delegate ModelArts to access your resources), and then use an existing agency instead of adding an agency on the **Add Authorization** page.

ModelArts associates multiple users with one agency. This means that if two users need to configure the same agency, you do not need to create an agency for each user. Instead, you only need to configure the same agency for the two users.

Figure 8-2 Mapping between users and agencies



NOTE

Each user can use ModelArts only after being associated with an agency. However, even if the permissions assigned to the agency are insufficient, no error is reported when the API is called. An error occurs only when the system uses unauthorized functions. For example, you enable message notification when creating a training job. Message notification requires SMN authorization. However, an error occurs only when messages need to be sent for the training job. The system ignores some errors, and other errors may cause job failures. When you implement permission minimization, ensure that you will still have sufficient permissions for the required operations on ModelArts.

Strict Authorization

In strict authorization mode, explicit authorization by the account administrator is required for IAM users to access ModelArts. The administrator can add the required ModelArts permissions to common users through authorization policies.

In non-strict authorization mode, IAM users can use ModelArts without explicit authorization. The administrator needs to configure the deny policy for IAM users to prevent them from using some ModelArts functions.

The administrator can change the authorization mode on the **Permission Management** page.

NOTICE

The strict authorization mode is recommended. In this mode, IAM users must be authorized to use ModelArts functions. In this way, the permission scope of IAM users can be accurately controlled, minimizing permissions granted to IAM users.

Managing Resource Access Using Workspaces

Workspace enables enterprise customers to split their resources into multiple spaces that are logically isolated and to manage access to different spaces. As an enterprise user, you can submit the request for enabling the workspace function to your technical support manager.

After workspace is enabled, a default workspace is created. All resources you have created are in this workspace. A workspace is like a ModelArts twin. You can switch between workspaces in the upper left corner of the ModelArts console. Jobs in different workspaces do not affect each other.

When creating a workspace, you must bind it to an enterprise project. Multiple workspaces can be bound to the same enterprise project, but one workspace cannot be bound to multiple enterprise projects. You can use workspaces for refined restrictions on resource access and permissions of different users. The restrictions are as follows:

- Users must be authorized to access specific workspaces (this must be configured on the pages for creating and managing workspaces). This means that access to AI assets such as datasets and algorithms can be managed using workspaces.
- In the preceding permission authorization operations, if you set the scope to enterprise projects, the authorization takes effect only for workspaces bound to the selected projects.

NOTE

- Restrictions on workspaces and permission authorization take effect at the same time. That is, a user must have both the permission to access the workspace and the permission to create training jobs (the permission applies to this workspace) so that the user can submit training jobs in this workspace.
- If you have enabled an enterprise project but have not enabled a workspace, all operations are performed in the default enterprise project. Ensure that the permissions on the required operations apply to the default enterprise project.
- The preceding restrictions do not apply to users who have not enabled any enterprise project.

Summary

Key features of ModelArts permissions management:

- If you are an individual user, you do not need to consider fine-grained permissions management. Your account has all permissions to use ModelArts by default.
- All functions of ModelArts are controlled by IAM. You can use IAM authorization to implement fine-grained permissions management for specific users.
- All users (including individual users) can use specific functions only after agency authorization on ModelArts (**Settings** > **Add Authorization**). Otherwise, unexpected errors may occur.
- If you have enabled the enterprise project function, you can also enable ModelArts workspace and use both basic authorization and workspace for refined permissions management.

8.2 Permission Management Mechanisms

8.2.1 IAM

This section describes the IAM permission configurations for all ModelArts functions.

IAM Permissions

If you need to assign different permissions to employees in your enterprise to access your purchased ModelArts resources, Identity and Access Management (IAM) is a good choice for fine-grained permissions management. IAM provides identity authentication, fine-grained permissions management, and access control. IAM helps you secure access to your cloud resources. If your Huawei account can meet your requirements and you do not need an IAM account to manage user permissions, skip this chapter.

IAM is a free service. You only pay for the resources in your account.

With IAM, you can assign permissions to control users' access to specific resources. For example, if the software developers in your enterprise need to own permissions to use ModelArts, yet you do not want them to own high-risk operation permissions such as deleting ModelArts, you can grant permissions using IAM to limit their permission on ModelArts.

For details about IAM, see **What is IAM?**.

Role/Policy-based Authorization

ModelArts supports role/policy-based authorization. By default, new IAM users do not have any permissions. You need to add a user to one or more groups, and assign permissions policies or roles to these groups. Users inherit permissions of the groups to which they are added. This process is called authorization. The users then inherit permissions from the groups and can perform specified operations on cloud services.

ModelArts is a project-level service deployed for specific regions. When you set **Scope** to **Region-specific projects** and select the specified projects (for example, **ap-southeast-2**) in the specified regions (for example, **AP-Bangkok**), the users only have permissions for ModelArts resources in the selected projects. If you set **Scope** to **All resources**, the users have permissions for ModelArts resources in all region-specific projects. When accessing ModelArts, the users need to switch to a region where they have been authorized to use cloud services.

Table 8-2 lists all system-defined policies supported by ModelArts. If preset ModelArts permissions cannot meet your requirements, create a custom policy by referring to **Policy Fields in JSON Format**.

Table 8-2 System-defined policies supported by ModelArts

Policy	Description	Type
ModelArts FullAccess	All permissions for ModelArts administrators	System-defined policy

Policy	Description	Type
ModelArts CommonOperations	All operation permissions for ModelArts common users, which do not include managing dedicated resource pools.	System-defined policy
ModelArts Dependency Access	Permissions on dependent services for ModelArts	System-defined policy

ModelArts depends on other cloud services. To check or view the cloud services, configure the corresponding permissions on the ModelArts console, as shown in the following table.

Table 8-3 Roles or policies that are required for performing operations on the ModelArts console

Console Function	Dependency	Role/Policy Required
Data management (dataset/data labeling/data processing)	Object Storage Service (OBS)	OBS Administrator
	Data Lake Insight (DLI)	DLI FullAccess
	MapReduce Service (MRS)	MRS Administrator
	GaussDB(DWS)	DWS Administrator
	Cloud Trace Service (CTS)	CTS Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Development environment notebook/ Image management/ Elastic node server	OBS	OBS Administrator
	Cloud Secret Management Service (CSMS)	CSMS ReadOnlyAccess
	CTS	CTS Administrator
	Elastic Cloud Server (ECS)	ECS FullAccess
	Software Repository for Container (SWR)	SWR Admin
	Scalable File Service (SFS)	SFS Turbo FullAccess

Console Function	Dependency	Role/Policy Required
	Application Operations Management (AOM)	AOM FullAccess
	Key Management Service (KMS)	KMS CMKFullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Algorithm management/ Training management/ Workflow/ ExeML	OBS	OBS Administrator
	Simple Message Notification (SMN)	SMN Administrator
	CTS	CTS Administrator
	Enterprise Project Management Service (EPS)	EPS FullAccess
	SFS Turbo	SFS ReadOnlyAccess
	SWR	SWR Admin
	AOM	AOM FullAccess
	KMS	KMS CMKFullAccess
	Virtual Private Cloud (VPC)	VPC FullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Model management/ Real-time service/Batch service/Edge service/Edge deployment dedicated resource pool	OBS	OBS Administrator
	Cloud Eye	CES ReadOnlyAccess
	SMN	SMN Administrator
	EPS	EPS FullAccess
	CTS	CTS Administrator
	Log Tank Service (LTS)	LTS FullAccess
	Virtual Private Cloud (VPC)	VPC FullAccess
	SWR	SWR Admin
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access

Console Function	Dependency	Role/Policy Required
AI Gallery	OBS	OBS Administrator
	CTS	CTS Administrator
	SWR	SWR Admin
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Elastic cluster (including standard and lite resource pools)	CTS	CTS Administrator
	Cloud Container Engine (CCE)	CCE Administrator
	Bare Metal Server (BMS)	BMS FullAccess
	Image Management Service (IMS)	IMS FullAccess
	Data Encryption Workshop (DEW)	DEW KeypairReadOnlyAccess
	VPC	VPC FullAccess
	ECS	ECS FullAccess
	SFS	SFS Turbo FullAccess
	OBS	OBS Administrator
	AOM	AOM FullAccess
	Tag Management Service (TMS)	TMS FullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
	Billing Center	BSS Administrator
	Elastic Volume Service (EVS)	EVS FullAccess

If system-defined policies cannot meet your requirements, you can create a custom policy. For details about the actions supported by custom policies, see **ModelArts Resource Permissions**.

To create a custom policy, choose either visual editor or JSON.

- Visual editor: Select cloud services, actions, resources, and request conditions without the need to know policy syntax.

- JSON: Create a JSON policy or edit an existing one.

For details, see **Creating a Custom Policy**. This section provides examples of common custom ModelArts policies.

- Example 1: Grant permission to manage images.

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:image:register",
                "modelarts:image:listGroup"
            ]
        }
    ]
}
```

- Example 2: Grant permission to deny creating, updating, and deleting a dedicated resource pool.

  A policy with only "Deny" permissions must be used together with other policies. If the permissions granted to an IAM user contain both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

```
{
    "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"
        }
    ]
}
```

- Example 3: Create a custom policy containing multiple actions.

  A custom policy can contain actions of multiple services that are of the global or project-level type. The following is an example policy containing actions of multiple services:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```



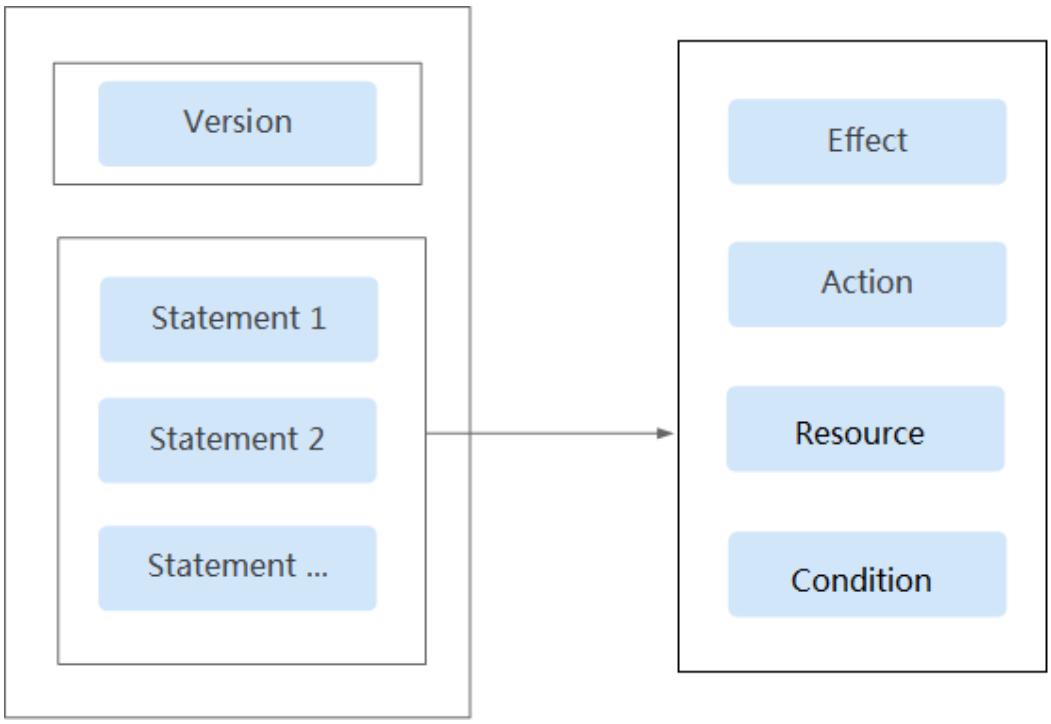
```
            "modelarts:service:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "lts:logs:list"
        ]
    }
  ]
}
```

Policy Fields in JSON Format

Policy Structure

A policy consists of a version and one or more statements (indicating different actions).

Figure 8-3 Policy structure



Policy Parameters

The following describes policy parameters. You can create custom policies by specifying the parameters. For details, see **Custom Policy Use Cases**.

Table 8-4 Policy parameters

Parameter	Description	Value
Version	Policy version	**1.1**: indicates policy-based access control.

Parameter		Description	Value
Statement: authorization statement of a policy	Effect	Whether to allow or deny the operations defined in the action	• **Allow**: indicates the operation is allowed. • **Deny**: indicates the operation is not allowed. **NOTE** If the policy used to grant user permissions contains both **Allow** and **Deny** for the same action, **Deny** takes precedence.
	Action	Operation to be performed on the service	Format: "*Service name*:*Resource type*:*Action*". Wildcard characters (*) are supported, indicating all options. Example: **modelarts:notebook:list** indicates the permission to view a notebook instance list. **modelarts** indicates the service name, **notebook** indicates the resource type, and **list** indicates the operation. View all actions of a service in its *API Reference*.
	Condition	Condition for a policy to take effect, including **condition keys and operators**	Format: "*Condition operator*:{*Condition key*:[*Value 1*,*Value 2*]}" If you set multiple conditions, the policy takes effect only when all the conditions are met. Example: **StringEndWithIfExists": {"g:UserName":["specialCharacter"]}**: The statement is valid for users whose names end with **specialCharacter**.
	Resource	Resources on which a policy takes effect	Format: *Service name*:<*Region*>:<*Account ID*>:*Resource type*:*Resource path*. Asterisks (*) are supported for resource type, indicating all resources. **NOTE** ModelArts authorization does not allow you to specify a resource path.

ModelArts Resource Types

Administrators can specify the scope based on ModelArts resource types. The following table lists the resource types supported by ModelArts:

Table 8-5 Resource types supported by ModelArts role/policy-based authorization

Resource Type	Description
notebook	Notebook instances in DevEnviron
exemlProject	ExeML projects
exemlProjectInf	ExeML-powered real-time inference service
exemlProjectTrain	ExeML-powered training jobs
exemlProjectVersion	ExeML project version
workflow	Workflow
pool	Dedicated resource pool
network	Networking of a dedicated resource pool
trainJob	Training job
trainJobLog	Runtime logs of a training job
trainJobInnerModel	Preset model
model	Models
service	Real-time service
nodeservice	Edge service
workspace	Workspace
dataset	Dataset
dataAnnotation	Dataset labels
aiAlgorithm	Algorithm for training jobs
image	Image
devserver	Elastic BMS

ModelArts Resource Permissions

For details, see "Permissions Policies and Supported Actions" in *ModelArts API Reference*.

- **Data Management Permissions**
- **DevEnviron Permissions**
- **Training Job Permissions**
- **Model Management Permissions**
- **Service Management Permissions**

8.2.2 Dependencies and Agencies

Function Dependency

Function Dependency Policies

When using ModelArts, you may be required to use other cloud services. For example, before submitting a training job, you must select OBS paths for storing the dataset and logs, respectively. Therefore, when configuring fine-grained authorization policies for a user, the administrator must configure dependent permissions so that the user can use required functions.

NOTE

- If you use ModelArts as the root user (default IAM user with the same name as the account), the root user has all permissions by default.
- Ensure that the current user has the dependent policy permissions for agency authorization. For example, if you want to grant the SWR Admin permission to a ModelArts agency, ensure that you have the SWR Admin permission.

Table 8-6 Basic configuration

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Global configuration	IAM	iam:users:listUsers	Obtain a user list. This action is required by the administrator only.
Basic function	IAM	iam:tokens:assume	(Mandatory) Use an agency to obtain temporary authentication credentials.
Basic function	BSS	bss:balance:view	Show the balance of the current account on the page after resources are created on the ModelArts console.

Table 8-7 Managing workspaces

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Workspace	IAM	iam:users:listUsers	Authorize an IAM user to use a workspace.
	ModelArts	modelarts:*:delete*	Clear resources in a workspace when deleting it.

Table 8-8 Managing notebook instances

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Lifecycle management of development environment instances	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	Start, stop, create, delete, and update an instance.
	AOM	aom:metric:get aom:metric:list aom:alarm:list	
	VPC	vpc:securityGroups:get vpc:vpcs:list vpc:securityGroups:get vpc:vpcs:list	

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Dynamically mounting storage	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	Dynamically mount storage.
	OBS	obs:bucket:ListAllMyBuckets obs:bucket:ListBucket	
Image management	ModelArts	modelarts:image:register modelarts:image:listGroup	Register and view an image on the **Image Management** page.
Saving an image	SWR	SWR Admin	The **SWR Admin** policy contains the maximum scope of SWR permissions, which can be used to: • Save a running development environment instance as an image. • Create a notebook instance using a custom image.
Using the SSH function	ECS	ecs:serverKeypairs:list ecs:serverKeypairs:get ecs:serverKeypairs:delete ecs:serverKeypairs:create	Configure a login key for a notebook instance.
	DEW	kps:domainKeypairs:get kps:domainKeypairs:list kps:domainKeypairs:createkmskey	
	KMS	kms:cmk:list	

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Mounting an SFS Turbo file system	SFS Turbo	SFS Turbo FullAccess	Read and write an SFS directory as an IAM user. Mount an SFS file system that is not created by you to a notebook instance using a dedicated resource pool.
Viewing all Instances	ModelArts	modelarts:notebook:listAllNotebooks	View development environment instances of all users on the ModelArts management console. This action is required by the development environment instance administrator.
	IAM	iam:users:listUsers	
Local VS Code plug-in or PyCharm Toolkit	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 (This policy is required if the **workspace** function is enabled.)	Access a notebook instance from local VS Code and submit training jobs.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	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	Obtain an IAM project list through local PyCharm for access configurations.

Table 8-9 Elastic node server

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Elastic node server lifecycle management	ModelArts	modelarts:devserver:create modelarts:devserver:listByUser modelarts:devserver:list modelarts:devserver:get modelarts:devserver:delete modelarts:devserver:start modelarts:devserver:stop modelarts:devserver:sync	Create, start, and stop an instance, obtain the instance list, obtain all instances of a tenant, obtain instance details, and synchronize instance status.
	ECS	ecs:serverKeypairs:createecs:*:get	
	IAM	iam:users:getUser iam:users:listUsers iam:projects:listProjects	
	VPC	vpc.*.list	
	EPS	eps.*.list	
	EVS	evs.*.list	
	IMS	ims.*.list ims.*.get	

Table 8-10 Managing training jobs

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Training management	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list modelarts:network:get modelarts:workspace:get	Create a training job and view training logs.
		modelarts:workspace:getQuota	Obtain a workspace quota. This policy is required if **workspace** function is enabled.
		modelarts:tag:list	Use Tag Management Service (TMS) in a training job.
	IAM	iam:credentials:listCredentials iam:agencies:listAgencies	Use the configured agency authorization.
	SFS Turbo	sfsturbo:shares:getShare sfsturbo:shares:getAllShares	Use SFS Turbo in a training job.
	SWR	SWR Admin	Use a custom image to create a training job.
	SMN	smn:topic:publish smn:topic:list	Notify training job status changes through SMN.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	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	Run a training job using a dataset in an OBS bucket.

Table 8-11 Using workflows

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Using a dataset	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	Use ModelArts datasets in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Model management	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	Manage ModelArts models in a workflow.
Deploying a service	ModelArts	modelarts:service:get modelarts:service:create modelarts:service:update modelarts:service:delete modelarts:service:getLogs	Manage ModelArts real-time services in a workflow.
Training jobs	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	Manage ModelArts training jobs in a workflow.
Workspace	ModelArts	modelarts:workspace:get modelarts:workspace:getQuotas	Use ModelArts workspaces in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing data	OBS	obs:bucket:ListAllMybuckets (Obtaining a bucket list) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:GetBucketLocation (Obtaining the bucket location) obs:object:GetObject (Obtaining object content and metadata) obs:object:GetObjectVersion (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:object:DeleteObject (Deleting an object or batch deleting objects) obs:object:DeleteObjectVersion (Deleting an object or batch deleting objects) obs:object:ListMultipartUpload-Parts (Listing uploaded parts) obs:object:AbortMultipartUpload (Aborting multipart uploads) obs:object:GetObjectAcl (Obtaining an object ACL) obs:object:GetObjectVersionAcl (Obtaining an object ACL) obs:bucket:PutBucketAcl (Configuring a bucket ACL) obs:object:PutObjectAcl (Configuring an object ACL)	Use OBS data in a workflow.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Executing a workflow	IAM	iam:users:listUsers (Obtaining users) iam:agencies:getAgency (Obtaining details about a specified agency) iam:tokens:assume (Obtaining an agency token)	Call other ModelArts services when a workflow is running.
Integrating DLI	DLI	dli:jobs:get (Obtaining job details) dli:jobs:list_all (Viewing a job list) dli:jobs:create (Creating a job)	Integrate DLI into a workflow.
Integrating MRS	MRS	mrs:job:get (Obtaining job details) mrs:job:submit (Creating and executing a job) mrs:job:list (Viewing a job list) mrs:job:stop (Stopping a job) mrs:job:batchDelete (Batch deleting jobs) mrs:file:list (Viewing a file list)	Integrate MRS into a workflow.

Table 8-12 Model management

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Model management	SWR	SWR Admin	Use a custom engine when you import a model from a custom image or OBS. SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	OBS	obs:bucket:ListAllMybuckets (Obtaining a bucket list) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:GetBucketLocation (Obtaining the bucket location) obs:object:GetObject (Obtaining object content and metadata) obs:object:GetObjectVersion (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:object:DeleteObject (Deleting an object or batch deleting objects) obs:object:DeleteObjectVersion (Deleting an object or batch deleting objects) obs:object:ListMultipartUpload-Parts (Listing uploaded parts) obs:object:AbortMultipartUpload (Aborting multipart uploads) obs:object:GetObjectAcl (Obtaining an object ACL) obs:object:GetObjectVersionAcl (Obtaining an object ACL) obs:bucket:PutBucketAcl (Configuring a bucket ACL) obs:object:PutObjectAcl (Configuring an object ACL)	Import a model from OBS. Specify an OBS path for model conversion.

Table 8-13 Managing service deployment

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Real-time services	LTS	lts:logs:list (Obtaining the log list)	Show LTS logs.
	OBS	obs:bucket:GetBucketPolicy (Obtaining a bucket policy) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:ListAllMyBuckets (Obtaining a bucket list) obs:bucket:PutBucketPolicy (Configuring a bucket policy) obs:bucket:DeleteBucketPolicy (Deleting a bucket policy)	Mount external volumes to a container when services are running.
Batch services	OBS	obs:object:GetObject (Obtaining object content and metadata) obs:object:PutObject (Uploading objects using PUT method, uploading objects using POST method, copying objects, appending an object, initializing a multipart task, uploading parts, and merging parts) obs:bucket:CreateBucket (Creating a bucket) obs:bucket:ListBucket (Listing objects in a bucket) obs:bucket:ListAllMyBuckets (Obtaining a bucket list)	Create batch services and perform batch inference.
Edge services	CES	ces:metricData:list: (Obtaining metric data)	View monitoring metrics.
	IEF	ief:deployment:delete (Deleting a deployment)	Manage edge services.
AOM metric alarm events	AOM	aom:alarm:list	View AOM monitoring information.

Table 8-14 Managing datasets

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing datasets and labels	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUpload-Parts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	Manage datasets in OBS. Label OBS data. Create a data management job.
Managing table datasets	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describeTable	Manage DLI data in a dataset.
Managing table datasets	GaussDB(DWS)	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	Manage DWS data in a dataset.
Managing table datasets	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:cluster:get	Manage MRS data in a dataset.
Auto labeling	ModelArts	modelarts:service:list modelarts:model:list modelarts:model:get modelarts:model:create modelarts:trainJobInnerModel:list modelarts:workspace:get modelarts:workspace:list	Enable auto labeling.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Team labeling	IAM	iam:projects:listProjects (Obtaining tenant projects) iam:users:listUsers (Obtaining users) iam:agencies:createAgency (Creating an agency) iam:quotas:listQuotasForProject (Obtaining the quotas of a project)	Manage labeling teams.

Table 8-15 Managing resources

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing resource pools	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	Create, renew, and unsubscribe from a resource pool.
	CCE	cce:cluster:list cce:cluster:get	Obtain the CCE cluster list, cluster details, and cluster certificates.
	KMS	kms:cmk:list kms:cmk:getMaterial	Obtain the key pairs created by the user.
	AOM	aom:metric:get	Obtain the monitoring data of a resource pool.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	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	Obtain AI diagnostic logs.
	ECS	ecs:availabilityZones:list ecs:cloudServerFlavors:get ecs:cloudServerQuotas:get ecs:quotas:get ecs:serverKeypairs:list	Obtain the AZs, specifications, and quotas, and configure keys.
	EVS	evs:types:get evs:quotas:get	Query EVS disk types and quotas.
	BMS	bms:serverFlavors:get	Query BMS specifications. Dependent permissions must be configured in the IAM project view.
	DEW	kps:domainKeypairs:list	Configure a key pair. Dependent permissions must be configured in the IAM project view.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Network management	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 vpc:securityGroups:get	Create and delete ModelArts networks, and interconnect VPCs.
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	Interconnect your network with SFS Turbo.

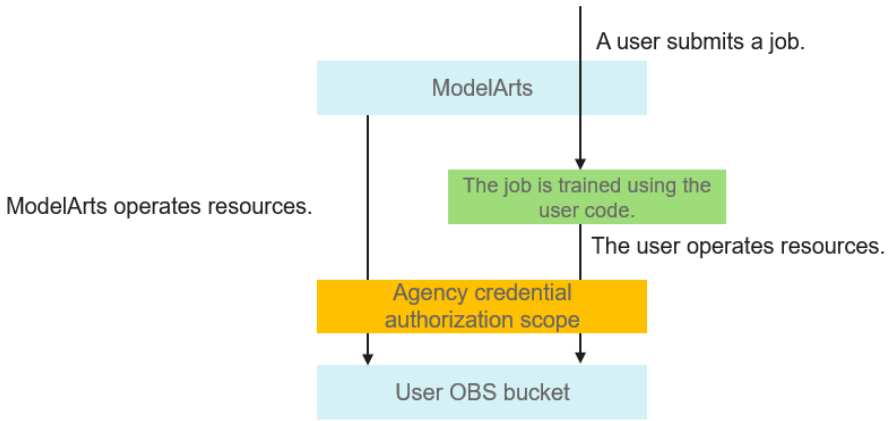
Application Scenario	Dependent Service	Dependent Policy	Supported Function
Edge resource pool	IEF	ief:node:list ief:group:get ief:application:list ief:application:get ief:node:listNodeCert ief:node:get ief:IEFInstance:get ief:deployment:list ief:group:listGroupInstanceState ief:IEFInstance:list ief:deployment:get ief:group:list	Add, delete, modify, and search for edge pools.

Agency authorization

To simplify operations when you use ModelArts, certain operations are automatically performed on the ModelArts backend, for example, downloading the datasets in an OBS bucket to a workspace before a training job is started and dumping training job logs to the OBS bucket.

ModelArts does not save your token authentication credentials. Before performing operations on your resources (such as OBS buckets) in a backend asynchronous job, you are required to explicitly authorize ModelArts through an IAM agency. ModelArts will use the agency to obtain a temporary authentication credential for performing operations on your resources. For details, see **Adding Authorization**.

Figure 8-4 Agency authorization



As shown in **Figure 8-4**, after authorization is configured on ModelArts, ModelArts uses the temporary credential to access and operate your resources, relieving you

from some complex and time-consuming operations. The agency credential will also be synchronized to your jobs (including notebook instances and training jobs). You can use the agency credential to access your resources in the jobs.

You can use either of the following methods to authorize ModelArts using an agency:

One-click authorization

ModelArts provides one-click automatic authorization. You can quickly configure agency authorization on the **Permission Management** page of ModelArts. Then, ModelArts will automatically create an agency for you and configure it in ModelArts.

In this mode, the authorization scope is specified based on the preset system policies of dependent services to ensure sufficient permissions for using services. The created agency has almost all permissions of dependent services. If you want to precisely control the scope of permissions granted to an agency, use the second method.

Custom authorization

The administrator creates different agency authorization policies for different users in IAM, and configures the created agency for ModelArts users. When creating an agency for an IAM user, the administrator specifies the minimum permissions for the agency based on the user's permissions to control the resources that the user can access when they use ModelArts. For details, see **Assigning Basic Permissions for Using ModelArts**.

Risks in Unauthorized Operations

The agency authorization of a user is independent. Theoretically, the agency authorization scope of a user can be beyond the authorization scope of the authorization policy configured for the user group. Any improper configuration will result in unauthorized operations.

To prevent unauthorized operations, only a tenant administrator is allowed to configure agencies for users to ensure the security of agency authorization.

Minimal Agency Authorization

When configuring agency authorization, an administrator must strictly control the authorization scope.

ModelArts asynchronously and automatically performs operations such as job preparation and clearing. The required agency authorization is within the basic authorization scope. If you use only some functions of ModelArts, the administrator can filter out the basic permissions that are not used according to the agency authorization configuration. Conversely, if you need to obtain resource permissions beyond the basic authorization scope in a job, the administrator can add new permissions to the agency authorization configuration. In a word, the agency authorization scope must be minimized and customized based on service requirements.

Basic Agency Authorization Scope

To customize the permissions for an agency, select permissions based on your service requirements.

Table 8-16 Basic agencies and authorizations in the development environment

Application Scenario	Dependent Service	Agency Authorization	Description
Performing operations on OBS data in a notebook instance	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	You can use either of the following methods to perform operations on OBS data in a notebook instance: • Use ModelArts SDK to perform operations on OBS data. • Use the notebook file upload function to perform operations on OBS data. • On the ModelArts console, add an OBS bucket to the **/data** directory of a notebook instance, and perform operations on OBS data in file mode.
Reporting notebook instance events	AOM	aom:alarm:put	During the lifecycle of a notebook instance, some events are reported to the AOM account. For details, see **Viewing Notebook Events**.
Interconnecting VPC with a notebook instance	VPC	vpc:ports:create vpc:ports:get vpc:ports:delete vpc:subnets:get	Add a NIC in the notebook instance for interconnecting with specified services in the VPC.
Connecting to a notebook instance through VS Code with one click	ModelArts	modelarts:notebook:get	Manage notebook instance details. Click **VS Code** to obtain the instance details and easily modify the instance information by writing the SSH configuration to the local VS Code.

Application Scenario	Dependent Service	Agency Authorization	Description
Stopping a notebook instance	ModelArts	modelarts:notebook:stop	Stops a running notebook instance.
Updating the auto stop time of a notebook instance	ModelArts	modelarts:notebook:updateStopPolicy	Update the auto stop time of a notebook instance.
MindInsight/ TensorBoard used in OBS parallel file systems	ModelArts	modelarts:notebook:umountStorage modelarts:notebook:getMountedStorage modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage	If MindInsight or TensorBoard is enabled in a notebook instance, and you need to access the OBS parallel file system, configure the permissions on the left.

Table 8-17 Basic agency authorization for training jobs

Application Scenario	Dependent Service	Agency Authorization	Description
Accessing OBS files for training jobs	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	You need to obtain OBS operation permissions when configuring a training job, including the code directory, input, output, and the OBS bucket path for storing logs.

Application Scenario	Dependent Service	Agency Authorization	Description
Starting a training job using a custom container image.	SWR	SWR Admin	When a training job is started using a custom container image, you need to obtain a temporary login command of the SWR container image to download the container image. SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.
Notification of training job status changes	SMN	smn:template:list smn:template:create smn:topic:list smn:topic:publish	To configure training job status change notifications, you must have the SMN operation permissions to send template-based notifications.
Mounting SFS Turbo to a training job	SFS Turbo	SFS Turbo ReadOnlyAccess	To mount SFS Turbo to a training job, you must have the SFS Turbo read permission to obtain its details by ID.
Reporting audit logs	CTS	CTS Administrator	Configure the CTS permission to report events. CTS does not support fine-grained permissions for event reporting. Therefore, you need to configure the administrator permission.

Table 8-18 Basic agency authorization for inference deployment

Application Scenario	Dependent Service	Agency Authorization	Description
Real-time services	LTS	lts:groups:create lts:groups:list lts:topics:create lts:topics:delete lts:topics:list	Configure LTS for reporting logs of real-time services.
Batch services	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	This parameter is mandatory when a batch service is used.
Edge services	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	This parameter is mandatory when an edge service is used. The edge service is deployed through IEF.
Importing a model from OBS	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	(Mandatory) If a parallel file system is used, you need to configure obs:bucket:HeadBucket.
Importing a model from the container image	SWR	SWR Admin	(Mandatory) SWR shared edition does not support fine-grained permissions. Therefore, the administrator permission is required.

Application Scenario	Dependent Service	Agency Authorization	Description
Using ModelArts Edge	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	(Optional) This function must be enabled if ModelArts Edge is used.
AOM metric alarm events	AOM	aom:log:get aom:alarm:get aom:metric:put aom:alarm:put aom:event:put aom:event:list aom:event:get	Enable this function to view alarms and events on AOM.
Reporting monitoring metrics to CES	CES	ces:metricMeta:create	Enable this function to report monitoring metrics to CES.
Message subscription and push	SMN	smn:topic:list smn:topic:publish smn:application:publish	(Optional) Enable this function for message subscription and push.

Table 8-19 Basic agency authorization for managing data

Application Scenario	Dependent Service	Agency Authorization	Description
Data labeling and processing	ModelArts	modelarts:trainJob:create modelarts:trainJob:update modelarts:trainJob:delete modelarts:trainJob:get modelarts:trainJob:list modelarts:trainJob:logExport modelarts:aiAlgorithm:get modelarts:model:get modelarts:service:list modelarts:model:create modelarts:workspace:list modelarts:workspace:get modelarts:trainJobInnerModel:list	(Mandatory) Create and query training jobs, as well as querying algorithms.
Accessing OBS data	OBS	obs:bucket:GetBucketLocation obs:bucket:PutBucketAcl obs:object:PutObjectAcl obs:object:GetObjectVersion obs:object:GetObject obs:object:GetObjectVersionAcl obs:object:DeleteObject obs:object:ListMultipartUpload-Parts obs:bucket:HeadBucket obs:object:AbortMultipartUpload obs:object:DeleteObjectVersion obs:object:GetObjectAcl obs:bucket:ListAllMyBuckets obs:bucket:ListBucket obs:object:PutObject	(Mandatory) Store, query, and delete data in OBS.

Application Scenario	Dependent Service	Agency Authorization	Description
Accessing DLI data	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:table:dropTable	(Optional) Enable this function if you need to view the DLI data.
Accessing MRS data	MRS	mrs:job:submit mrs:job:list mrs:cluster:list mrs:file:list	(Optional) Enable this function if you need to view the MRS data.
Accessing GaussDB(DWS) data	GaussDB(DWS)	dws:openAPICluster:list dws:openAPICluster:getDetail dws:cluster:list	(Optional) Enable this function if you need to view the GaussDB(DWS) data.

Table 8-20 Basic agency authorization for managing dedicated resource pools

Application Scenario	Dependent Service	Agency Authorization	Description
Interconnecting a dedicated resource pool with SFS Turbo resources	SFS Turbo	sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic	Enable this function as needed.

Application Scenario	Dependent Service	Agency Authorization	Description
Interconnecting ModelArts network with VPC and adding related routes	VPC	vpc:vpcs:get vpc:subnets:get vpc:peerings:accept vpc:routes:create vpc:routes:delete vpc:routes:get vpc:routeTables:update vpc:routeTables:get vpc:routeTables:list vpc:routes:list	Enable this function as needed.
Using ModelArts Lite Cluster resource pools	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	This function must be enabled if ModelArts Lite Cluster resource pools are used. ModelArts uses an agency to manage CCE clusters, synchronize cluster information, and manage nodes.

Application Scenario	Dependent Service	Agency Authorization	Description
	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 ecs:cloudServers:list bms:servers:create bms:serverFlavors:get evs:types:get evs:volumes:list evs:quotas:get evs:volumes:get kps:domainKeypairs:get	This function must be enabled if ModelArts Lite Cluster resource pools are used. ModelArts uses an agency to manage the lifecycle of BMSs and ECSs.
	IMS	ims:images:get ims:images:share	This function must be enabled if ModelArts Lite Cluster resource pools are used. Share the node system image with your account before creating a ModelArts Lite Cluster dedicated resource pool node.

8.2.3 Workspace

ModelArts allows you to create multiple workspaces to develop algorithms and manage and deploy models for different service objectives. In this way, the

development outputs of different applications are allocated to different workspaces for simplified management.

Workspace supports the following types of access control:

- **PUBLIC**: publicly accessible to tenants (including both tenant accounts and all their user accounts)
- **PRIVATE**: accessible only to the creator and tenant accounts
- **INTERNAL**: accessible to the creator, tenant accounts, and specified IAM user accounts. When **Authorization Type** is set to **INTERNAL**, specify one or more accessible IAM user accounts.

A default workspace is allocated to each IAM project of each account. The access control of the default workspace is **PUBLIC**.

Workspace access control allows the access of only certain users. This function can be used in the following scenarios:

- **Education**: A teacher allocates an **INTERNAL** workspace to each student and allows the workspaces to be accessed only by specified students. In this way, students can separately perform experiments on ModelArts.
- **Enterprises**: An administrator creates a workspace for production tasks and allows only O&M personnel to use the workspace, and creates a workspace for routine debugging and allows only developers to use the workspace. In this way, different enterprise roles can use resources only in a specified workspace.

As an enterprise user, you can submit the request for enabling the workspace function to your technical support.

8.3 Configuration Practices in Typical Scenarios

8.3.1 Assigning Permissions to Individual Users for Using ModelArts

Certain ModelArts functions require access to Object Storage Service (OBS), Software Repository for Container (SWR), and Intelligent EdgeFabric (IEF). Before using ModelArts, your account must be authorized to access these services. Otherwise, these functions will be unavailable.

Constraints

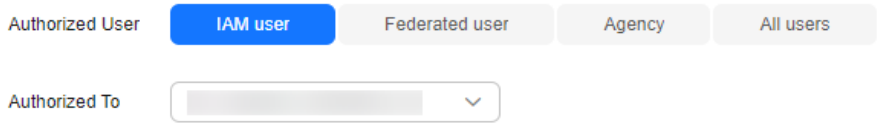
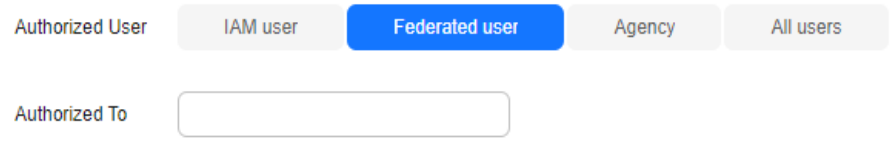
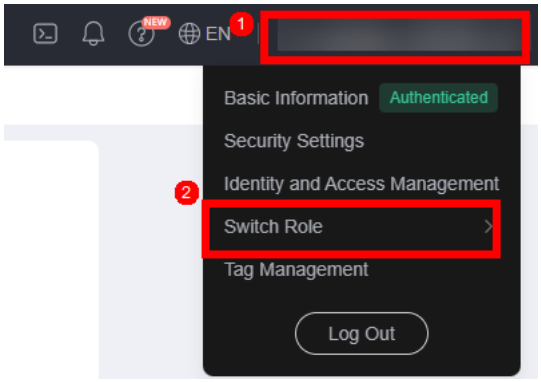
- Only a tenant account can perform agency authorization to authorize the current account or all IAM users under the current account.
- Multiple IAM users or accounts can use the same agency.
- A maximum of 50 agencies can be created under an account.
- If you use ModelArts for the first time, add an agency. Generally, common user permissions are sufficient for your requirements. You can configure permissions for refined permissions management.
- If you have not been authorized, ModelArts will display a message indicating that you have not been authorized when you access the **Add Authorization** page. In this case, contact your administrator to add authorization.

Adding Authorization

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**. The **Permission Management** page is displayed.
2. Click **Add Authorization**. On the **Add Authorization** page that is displayed, configure the parameters.

Table 8-21 Parameters

Parameter	Description
Authorized User	Options: **IAM user**, **Federated user**, **Agency**, and **All users** • **IAM user**: You can use a tenant account to create IAM users and assign permissions for specific resources. Each IAM user has their own identity credentials (password and access keys) and uses cloud resources based on assigned permissions. For details about IAM users, see **IAM User**. • **Federated user**: A federated user is also called a virtual enterprise user. For details about federated users, see **Configuring Federated Identity Authentication**. • **Agency**: You can create agencies in IAM. For details about how to create an agency, see **Creating an Agency**. • **All users**: If you select this option, the agency permissions will be granted to all IAM users under the current account, including those created in the future. For individual users, choose **All users**.

Parameter	Description
Authorized To	This parameter is not displayed when **Authorized User** is set to **All users**. • **IAM user**: Select an IAM user and configure an agency for the IAM user. **Figure 8-5** Selecting an IAM user  • **Federated user**: Enter the username or user ID of the target federated user. **Figure 8-6** Selecting a federated user  • **Agency**: Select an agency name. You can create an agency under account A and grant the agency permissions to account B. When using account B, you can switch the role in the upper right corner of the console to account A and use the agency permissions of account A. **Figure 8-7** Switch Role  **NOTE** ModelArts does not support the creation of agencies with identity policy permissions.
Agency	• **Use existing**: If there are agencies in the list, select an available one to authorize the selected user. Click the drop-down arrow next to an agency name to view its permission details. • **Add agency**: If there is no available agency, create one. If you use ModelArts for the first time, select **Add agency**.

Parameter	Description
Add agency > Agency Name	The system automatically creates a changeable agency name.
Add agency > Permissions > Common User	**Common User** provides the permissions to use all basic ModelArts functions. For example, you can access data, and create and manage training jobs. Select this option generally. Click **View permissions** to view common user permissions.
Add agency > Permissions > Custom	If you need refined permissions management, select **Custom** to flexibly assign permissions to the created agency. You can select permissions from the permission list as required.

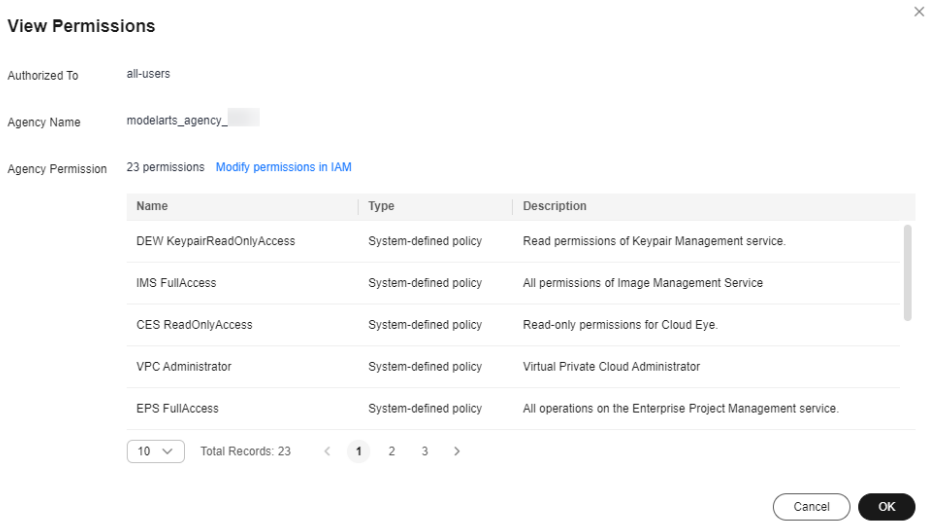
3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

Viewing Authorized Permissions

You can view the configured authorizations on the **Permission Management** page. Click **View Permissions** in the **Authorization Content** column to view the permission details.

Figure 8-8 View Permissions

Figure 8-9 Common user permissions



8.3.2 Assigning Basic Permissions for Using ModelArts

8.3.2.1 Scenario

Certain ModelArts functions require the permission to access other services. This section describes how to assign specific permissions to IAM users when they use ModelArts.

Permissions

The permissions of IAM users are controlled by their tenant user. Logging in as a tenant user, you can assign permissions to the target user group through IAM. Then, the permissions are assigned to all members in the user group. The following authorization list uses the system-defined policies of ModelArts and other services as an example.

Table 8-22 Service authorization

Target Service	Description	IAM Permission	Mandatory
ModelArts	Assign permissions to IAM users for using ModelArts. The users with the ModelArts CommonOperations permission can only use resources, but cannot create, update, or delete any dedicated resource pool. You are advised to assign this permission to IAM users.	ModelArts CommonOperations	Yes
	The users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when selecting this option.	ModelArts FullAccess	No Select either **ModelArts FullAccess** or **ModelArts CommonOperations**.
Object Storage Service (OBS)	Assign permissions to IAM users for using OBS. ModelArts data management, development environments, training jobs, and model deployment require OBS for forwarding data.	OBS OperateAccess	Yes

Target Service	Description	IAM Permission	Mandatory
Software Repository for Container (SWR)	Assign permissions to IAM users for using SWR. ModelArts custom images require the SWR FullAccess permission.	SWR OperateAccess	Yes
Key Management Service (KMS)	To use remote SSH of ModelArts notebook, IAM users require KMS authorization.	KMS CMKFullAccess	No
Intelligent EdgeFabric (IEF)	Assign permissions to IAM users for using IEF. Tenant administrator permissions are required so that ModelArts edge services depending on IEF can be used.	Tenant Administrator	No
Cloud Eye	Assign permissions to IAM users for using Cloud Eye. Using Cloud Eye, you can view the running statuses of ModelArts real-time services and AI application loads, and set monitoring alarms.	CES FullAccess	No
Simple Message Notification (SMN)	Assign permissions to IAM users for using SMN. SMN is used with Cloud Eye.	SMN FullAccess	No
Virtual Private Cloud (VPC)	During the creation of a dedicated resource pool for ModelArts, IAM users require VPC permissions so that they can customize networks.	VPC FullAccess	No

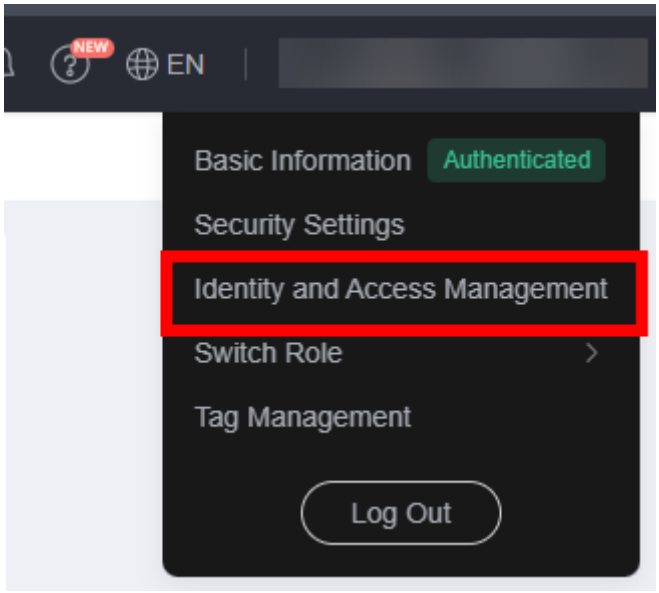
Target Service	Description	IAM Permission	Mandatory
SFS	Assign permissions to IAM users for using SFS. SFS file systems can be mounted to ModelArts dedicated resource pools to serve as storage for development environments or training.	SFS Turbo FullAccess SFS FullAccess	No

8.3.2.2 Step 1 Creating a User Group and Adding Users to the User Group

Multiple IAM users can be created under a tenant user, and the permissions of the IAM users are managed by group. This section describes how to create a user group and IAM users and add the IAM users to the user group.

1. Log in to the management console as a tenant user, hover over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.

   Figure 8-10 Identity and Access Management

2. Create a user group. In the navigation pane on the left, choose **User Groups**. Click **Create User Group** in the upper right corner. Then, set **Name** to **UserGroup-2** and click **OK**.

   After the user group is created, the system automatically switches to the user group list. Then, you can add existing IAM users to the user group through user group management. If there is no existing IAM user, create users and add them to the user group.

3. Create IAM users and add them to the user group. In the navigation pane on the left, choose **Users**. On the displayed page, click **Create User** in the upper right corner. On the **Create User** page, add multiple users.

   Set parameters as prompted and click **Next**.

4. On the **Add User to Group** page, select **UserGroup-2** and click **Create**.

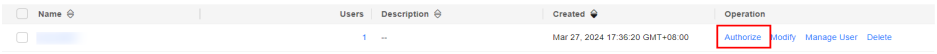
   The system will automatically add the two users to the target group one by one.

8.3.2.3 Step 2 Assigning Permissions for Using Cloud Services

An IAM user can use cloud services such as ModelArts and OBS only after they are assigned with permissions from the tenant user. This section describes how to assign the permissions to use cloud services to all IAM users in a user group.

1. On the user group list page of IAM, click **Authorize** of the target user group. The **Authorize User Group** page is displayed.

   Figure 8-11 Authorize

2. Before assigning permissions, learn about minimum permissions requirements of each ModelArts module, as shown in **Table 8-22**.
3. Assign permissions for using ModelArts. Search for ModelArts in the search box. Select either **ModelArts FullAccess** or **ModelArts CommonOperations**.

   The differences between the options are as follows:

   - The users with the ModelArts CommonOperations permission can only use resources, but cannot create, update, or delete any dedicated resource pool. You are advised to assign this permission to IAM users.
   - The users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when selecting this option.

4. Assign permissions for using OBS. Search for **OBS** and select **OBS Administrator**. ModelArts training jobs use OBS to forward data. Therefore, the permissions for using OBS are necessary.
5. Assign permissions for using SWR. Search for **SWR** and select **SWR FullAccess**. ModelArts custom images require the SWR FullAccess permission.
6. (Optional) Assign the key management permission. Remote SSH of ModelArts notebook requires the key management permission. Search for **DEW** and select **DEW KeypairFullAccess**.

   DEW key management permission is configured in the following regions: **CN North-Beijing1**, **CN North-Beijing4**, **CN East-Shanghai1**, **CN East-Shanghai2**, **CN South-Guangzhou**, **CN Southwest-Guiyang1**, **CN-Hong Kong**, and **AP-Singapore**. In other regions, the KMS key management permission is configured. In this example, the **CN-Hong Kong** region is used. Therefore, the DEW key management permission is to be configured.

7. (Optional) Assign permissions for using IEF. ModelArts requires the Tenant Administrator permission so that edge services depending on IEF can be used.

Tenant Administrator has the permission to manage all cloud services, not only ModelArts. Exercise caution when assigning the Tenant Administrator permission.

8. (Optional) Assign permissions for using Cloud Eye and SMN. On the details page of a ModelArts real-time service deployed for inference, the number of calls is available. Click **View Details** to obtain more information. If you want to view the overall running status of ModelArts real-time services and AI application loads on Cloud Eye, assign Cloud Eye permissions to IAM users.

   To view monitoring data only, select **CES ReadOnlyAccess**.

   To set alarm monitoring on Cloud Eye, you also need to add **CES FullAccess** and SMN permissions.

9. (Optional) Assign permissions for using VPC. To enable custom network configuration when creating a dedicated resource pool, assign permissions for using VPC.
10. (Optional) Assign permissions for using SFS and SFS Turbo. To mount an SFS system to a dedicated resource pool as the storage for the development environment or training, assign the permission to use the SFS system.
11. Click **View Selected** in the upper left corner and confirm the selected permissions.
12. Click **Next** and set the minimum authorization scope. Select **Region-specific projects**, select the region to be authorized, and click **OK**.
13. A message is displayed, indicating that the authorization is successful. View the authorization information and click **Finish**. It takes 15 to 30 minutes for the authorization to take effect.

8.3.2.4 Step 3 Configuring Agent-based ModelArts Access Authorization

After assigning IAM permissions, configure ModelArts access authorization for IAM users on the ModelArts page so that ModelArts can access dependent services such as OBS, SWR, and IEF.

In agent-based ModelArts access authorization, only tenant users are allowed to configure for their IAM users. In this example, use the administrator account to configure access authorization for all the users.

1. Use the tenant account to log in to the ModelArts management console. Select your region in the upper left corner.
2. In the navigation pane on the left, click **Permissions** to go to the permission management page.
3. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **All users** and click **Add agency** to configure the agency-based authorization for all IAM users under the account.
   - **Common User**: You can use basic ModelArts functions, for example, accessing data and creating and managing training jobs, but not to manage resources. Select this option generally.
   - **Custom**: You can flexibly assign permissions to the created agency. Select this option for refined permissions management. You can select permissions from the permission list as required.
4. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

8.3.2.5 Step 4 Verifying User Permissions

It takes 15 to 30 minutes for the permissions configured in **4** to take effect. Therefore, wait for 30 minutes after the configuration and then verify the configuration.

1. Log in to the ModelArts management console as an IAM in **UserGroup-2**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. Check ModelArts permissions.
   a. Select the target region in the upper left corner, which must be the same as that in the authorization configuration.
   b. In the navigation pane on the left of the ModelArts management console, choose **DevEnviron** > **Notebook**. The ModelArts permissions and agency authorization are configured correctly if no message shows insufficient permissions.

      If a message is displayed, indicating that you need to authorize access, the ModelArts agency authorization has not been configured. In this case, follow the instructions in **Step 3 Configuring Agent-based ModelArts Access Authorization** to configure the authorization.
   c. In the navigation pane on the left of the ModelArts management console, choose **DevEnviron** > **Notebook** and click **Create**. If this operation is successful, you have obtained ModelArts operation permissions.

      Alternatively, you can try other functions, such as **Training Management** > **Training Jobs**. If the operation is successful, you can use ModelArts properly.
3. Verify OBS permissions.
   a. In the service list in the upper left corner, select OBS. The OBS management console is displayed.
   b. Click **Create Bucket** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
4. Verify SWR permissions.
   a. In the service list in the upper left corner, select SWR. The SWR management console is displayed.
   b. If an SWR page can be properly displayed, you have obtained SWR operation permissions.
5. Verify other optional permissions.
6. Experience ModelArts.

8.3.3 Separately Assigning Permissions to Administrators and Developers

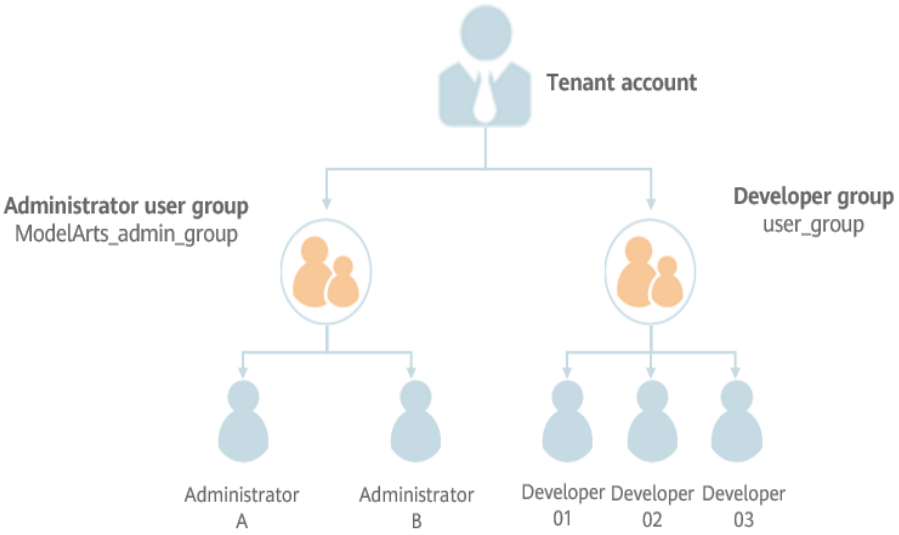
In small- and medium-sized teams, administrators need to globally control ModelArts resources, and developers only need to focus on their own instances. Generally, the **te_admin** permission of a developer account must be configured by the tenant account. This section uses notebook as an example to describe how to assign different permissions to administrators and developers through custom policies.

Scenarios

To develop a project using notebook, administrators need full control permissions for using ModelArts dedicated resource pools, and access and operation permissions on all notebook instances.

To use development environments, developers only need operation permissions for using their own instances and dependent services. They do not need to perform operations on ModelArts dedicated resource pools or view notebook instances of other users.

Figure 8-12 Account relationships



Configuring Permissions for an Administrator

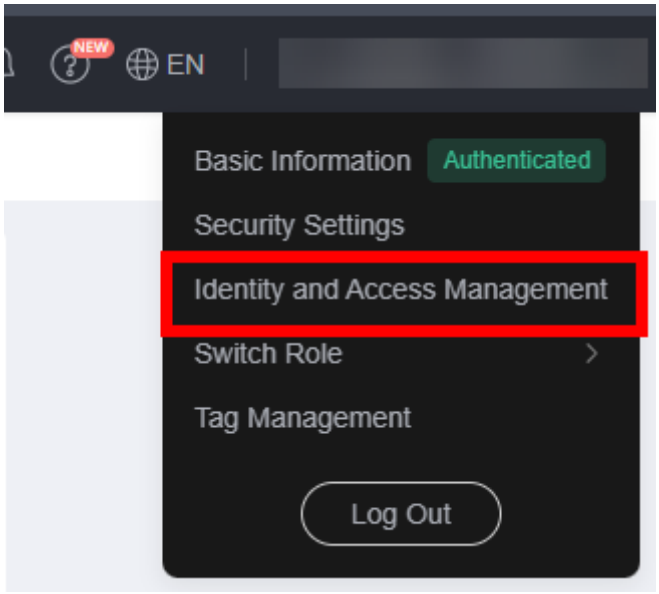
Assign full control permissions to administrators for using ModelArts dedicated resource pools and all notebook instances. The procedure is as follows:

Step 1 Use a tenant account to create an administrator user group **ModelArts_admin_group** and add administrator accounts to **ModelArts_admin_group**. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 2 Create a custom policy.

1. Log in to the management console using an administrator account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.

Figure 8-13 Identity and Access Management



2. Create custom policy 1 and assign IAM and OBS permissions to the user. In the navigation pane of the IAM console, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy1_IAM_OBS** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

   The custom policy **Policy1_IAM_OBS** is as follows, which grants IAM and OBS operation permissions to the user. You can directly copy and paste the content.

```
{
   "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. Repeat **Step 2.2** to create custom policy 2 and grant the user the permissions to perform operations on dependent services ECS, SWR, MRS, and SMN as well as ModelArts. Set **Policy Name** to **Policy2_AllowOperation** and **Policy View** to **JSON**, configure the policy content, and click **OK**.

   The custom policy **Policy2_AllowOperation** is as follows, which grants the user the permissions to perform operations on dependent services ECS, SWR, MRS, and SMN as well as ModelArts. You can directly copy and paste the content.

```
{
   "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:*:*"
        ]
    }
  ]
}
```

Step 3 Grant the policy created in **Step 2** to the administrator group **ModelArts_admin_group**.

1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **ModelArts_admin_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS** and **Policy2_AllowOperation**. Click **Next**.
2. Specify the scope as **All resources** and click **OK**.

Step 4 Configure agent-based ModelArts access authorization for an administrator to allow ModelArts to access dependent services such as OBS.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select an administrator account for **Authorized To**, select **Add agency**, and select **Common User** for **Permissions**. Permissions control is not required for administrators, so use default setting **Common User**.
3. Select **I have read and agree to the ModelArts Service Statement**. Click **Create**.

Step 5 Test administrator permissions.

1. Log in to the ModelArts management console as the administrator. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. In the navigation pane of the ModelArts management console, choose **Dedicated Resource Pools** and click **Create**. If the console does not display a message indicating insufficient permissions, the permissions have been assigned to the administrator.

----End

Configuring Permissions for a Developer

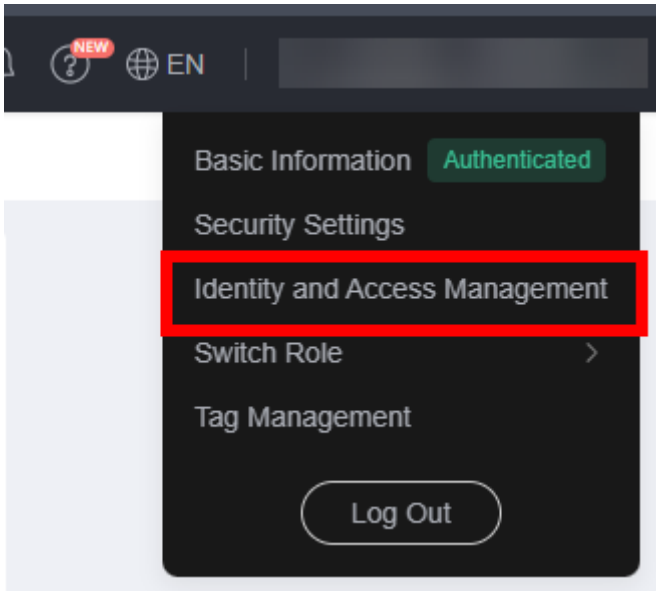
Use IAM for fine-grained control of developer permissions. The procedure is as follows:

Step 1 Use a tenant account to create a developer user group **user_group** and add developer accounts to **user_group**. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 2 Create a custom policy.

1. Log in to the management console using a tenant account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.

   Figure 8-14 Identity and Access Management

2. Create custom policy 3 to prevent users from performing operations on ModelArts dedicated resource pools and viewing notebook instances of other users.

   In the navigation pane of the IAM console, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy3_DenyOperation** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

   The custom policy **Policy3_DenyOperation** is as follows. You can copy and paste the content.

```
{
   "Version": "1.1",
   "Statement": [
         {
          "Effect": "deny",
          "Action": [
              "modelarts:pool:create",
              "modelarts:pool:update",
              "modelarts:pool:delete",
      "modelarts:notebook:listAllNotebooks"
            ]

         }
   ]
}
```

Step 3 Grant the custom policy to the developer user group **user_group**.

1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **user_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS**, **Policy2_AllowOperation**, and **Policy3_DenyOperation**. Click **Next**.
2. Specify the scope as **All resources** and click **OK**.

Step 4 Configure agent-based ModelArts access authorization for a developer to allow ModelArts to access dependent services such as OBS.

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select a developer account for **Authorized To**, add an agency **ma_agency_develop_user**, set **Permissions** to **Custom**, and select **OBS Administrator**. Developers only need OBS authorization to allow developers to access OBS when using notebook.
3. Click **Create**.
4. On the **Permission Management** page, click **Add Authorization** again. On the **Add Authorization** page that is displayed, configure an agency for other developer users.

   On the **Add Authorization** page, set **Authorized User** to **IAM user**, select a developer account for **Authorized To**, and select the existing agency **ma_agency_develop_user** created before.

Step 5 Test developer permissions.

1. Log in to the ModelArts management console as an IAM user in **user_group**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. In the navigation pane of the ModelArts management console, choose **Dedicated Resource Pools** and click **Create**. If the console does not display a message indicating insufficient permissions, the permissions have been assigned to the developer.

----End

8.3.4 Assigning the Required Permissions

Searching for an Instance

All created instances are displayed on the notebook page. To display a specific instance, search for it based on filter criteria.

- **Grant the permission to the IAM user for viewing all notebook instances**. Log in to the ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**. On the displayed page, enable **View all**.
- Set search criteria, such as name, ID, status, image, flavor, description, and creation time.

Assigning the Required Permissions

Any IAM user granted with the **listAllNotebooks** and **listUsers** permissions can click **View all** on the notebook page to view the instances of all IAM users in the

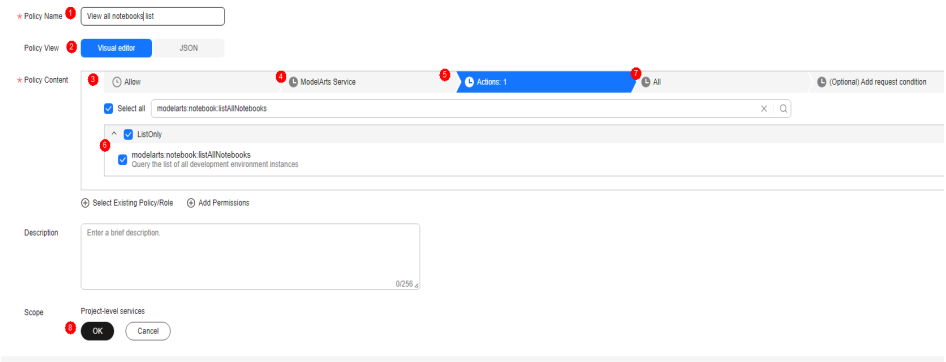
current IAM project. After the permission is granted, you can access OBS and SWR of IAM users in a notebook instance.

1. Log in to the ModelArts management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions** > **Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and create two policies.

   Policy 1: Create a policy that allows users to view all notebook instances of an IAM project, as shown in **Figure 8-15**.

   - **Policy Name**: Enter a custom policy name, for example, **Viewing all notebook instances**.
   - **Policy View**: Select **Visual editor**.
   - **Policy Content**: Select **Allow**, **ModelArts Service**, **modelarts:notebook:listAllNotebooks**, and default resources.

   Figure 8-15 Creating a custom policy

   Policy 2: Create a policy that allows users to view all users of an IAM project.

   - **Policy Name**: Enter a custom policy name, for example, **Viewing all users of the current IAM project**.
   - **Policy View**: Select **Visual editor**.
   - **Policy Content**: Select **Allow**, **Identity and Access Management**, **iam:users:listUsers**, and default resources.
3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in **2**, and click **Next**. Then, select the scope and click **OK**.

   After the configuration, all users in the user group have the permission to view all notebook instances created by users in the user group.

   If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

Starting Notebook Instances of Other IAM Users

If an IAM user wants to access another IAM user's notebook instance through remote SSH, they need to update the SSH key pair to their own. Otherwise, error **ModelArts.6786** will be reported. For details about how to update a key pair, see **Modifying the SSH Configuration for a Notebook Instance**. The error message is as follows: "ModelArts.6789: Failed to find SSH key pair *xxx* on the ECS key pair page. Update the key pair and try again later."

8.3.5 Logging In to a Training Container Using Cloud Shell

Application Scenario

You can use Cloud Shell provided by the ModelArts console to log in to a running training container.

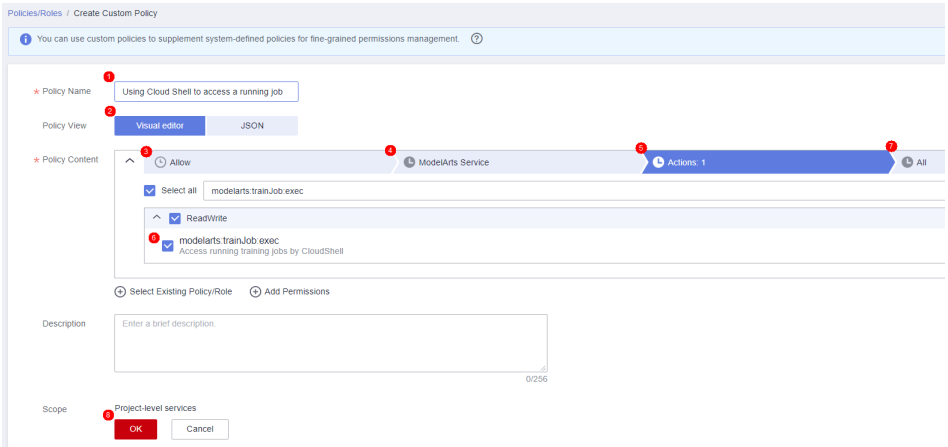
Constraints

Only dedicated resource pools support Cloud Shell. The training job must be in the **Running** state.

Preparation: Assigning the Cloud Shell Permission to an IAM User

1. Log in to the Huawei Cloud management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions** > **Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and configure the following parameters.
   - **Policy Name**: Enter a custom policy name, for example, **Using Cloud Shell to access a running job**.
   - **Policy View**: Select **Visual editor**.
   - **Policy Content**: Select **Allow**, **ModelArts Service**, **modelarts:trainJob:exec**, and default resources.

Figure 8-16 Creating a custom policy



3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in 2, and click **Next**. Then, select the scope and click **OK**.

   After the configuration, all users in the user group have the permission to use Cloud Shell to log in to a running training container.

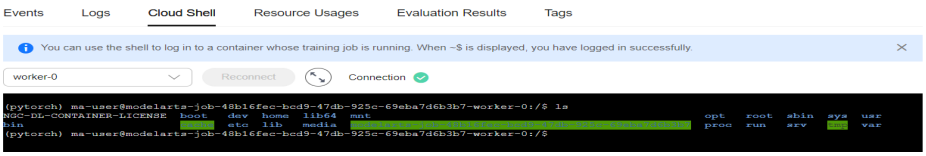
   If no user group is available, create a user group, add users using the user group management function, and configure authorization. If the target user is not in a user group, you can add the user to a user group through the user group management function.

Using Cloud Shell

1. Configure parameters based on **Preparation: Assigning the Cloud Shell Permission to an IAM User**.
2. On the ModelArts console, choose **Model Training** > **Training Jobs** from the navigation pane.
3. In the training job list, click the name of the target job to go to the training job details page.
4. On the training job details page, click the **Cloud Shell** tab and log in to the training container.

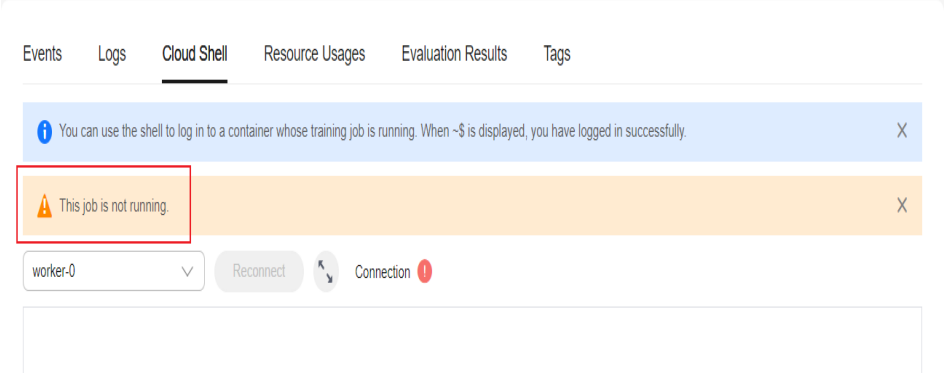
   Verify that the login is successful, as shown in the following figure.

Figure 8-17 Cloud Shell page



If the job is not running or the permission is insufficient, Cloud Shell cannot be used. In this case, locate the fault as prompted.

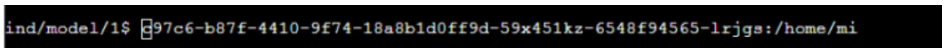
Figure 8-18 Error message



NOTE

A path display exception may occur when you log in to the Cloud Shell page. In this case, press **Enter** to rectify the fault.

Figure 8-19 Abnormal path



8.3.6 Prohibiting a User from Using a Public Resource Pool

This section describes how to control the ModelArts permissions of a user so that the user is not allowed to use a public resource pool to create training jobs, create notebook instances, or deploy inference services.

Context

Through permission control, ModelArts dedicated resource pool users can be prohibited from using a public resource pool to create training jobs, create notebook instances, or deploy inference services.

To control the permissions, configure the following permission policy items:

- **modelarts:notebook:create**: allows you to create a notebook instance.
- **modelarts:trainJob:create**: allows you to create a training job.
- **modelarts:service:create**: allows you to create an inference service.

Procedure

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. In the navigation pane, choose **Permissions** > **Policies/Roles**. On the **Policies/Roles** page, click **Create Custom Policy** in the upper right corner, configure parameters, and click **OK**.
   - **Policy Name**: Configure the policy name.
   - **Policy View**: Select **Visual editor** or **JSON**.
   - **Policy Content**: Select **Deny**. In **Select service**, search for **ModelArts** and select it. In **ReadWrite** under **Actions**, search for **modelarts:trainJob:create**, **modelarts:notebook:create**, and **modelarts:service:create** and select them. **All**: Retain the default setting. In **Add request condition**, click **Add Request Condition**. In the displayed dialog box, set **Condition Key** to **modelarts:poolType**, **Operator** to **StringEquals**, and **Value** to **public**.

     The policy content in JSON view is as follows:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "modelarts:trainJob:create",
                "modelarts:notebook:create",
```



```
            "modelarts:service:create"
        ],
        "Condition": {
            "StringEquals": {
                "modelarts:poolType": [
                    "public"
                ]
            }
        }
    }
  ]
}
```

3. In the navigation pane, choose **User Groups**. On the **User Groups** page, locate the row containing the target user group and click **Authorize** in the **Operation** column. On the **Authorize User Group** page, select the custom policy created in 2 and click **Next**. Then, select the scope and click **OK**.

   After the configuration, all users in the user group have the permission to view all notebook instances created by users in the user group.

   If no user group is available, create one, add users to it through user group management, and configure authorization for the user group. If the target user is not in a user group, add the user to a user group through user group management.

4. Add the policy to the user's agency authorization. This prevents the user from breaking the permission scope through a token on the tenant plane.

   In the navigation pane, choose **Agencies**. Locate the agency used by the user group on ModelArts and click **Modify** in the **Operation** column. In the **Permissions** tab, click **Authorize**, select the created custom policy, and click **Next**. Select the scope for authorization and click **OK**.

Verification

Log in to the ModelArts console as an IAM user, choose **Model Training** > **Training Jobs**, and click **Create Training Job**. On the page for creating a training job, only a dedicated resource pool can be selected for **Resource Pool**.

Log in to the ModelArts console as an IAM user, choose **Development Workspace** > **Notebook**, and click **Create**. On the page for creating a notebook instance, only a dedicated resource pool can be selected for **Resource Pool**.

Log in to the ModelArts console as an IAM user, choose **Model Deployment** > **Real-Time Services**, and click **Deploy**. On the page for service deployment, only a dedicated resource pool can be selected for **Resource Pool**.

8.3.7 Authorizing ModelArts to Use SFS Turbo

This section describes how to configure ModelArts agency permissions to enable you to associate and disassociate your dedicated resource pool network with an SFS Turbo file system.

- To add an agency and authorize it to perform operations on SFS Turbo, see **Adding an Agency and Granting Permissions for SFS Turbo**.
- To add permissions to an existing agency for SFS Turbo, see **Adding Permissions to an Existing Agency for SFS Turbo**.

Context

After creating a network on the console, you can click **More** in the **Operation** column to access the options of **Associate SFS Turbo** and **Disassociate SFS Turbo** on the **Network** page. When you associate a network with an SFS Turbo file system, the file system joins the network and becomes available for training and development purposes.

Before you can associate or disassociate an SFS Turbo file system, you must grant ModelArts the permissions to perform operations on SFS Turbo.

To control the permissions, configure the following permission policy items:

- sfsturbo:shares:addShareNic: allows you to create NICs.
- sfsturbo:shares:deleteShareNic: allows you to delete NICs.
- sfsturbo:shares:showShareNic: allows you to obtain NIC details.
- sfsturbo:shares:listShareNics: allows you to obtain NICs.

Constraints

This function is available in certain regions.

Adding an Agency and Granting Permissions for SFS Turbo

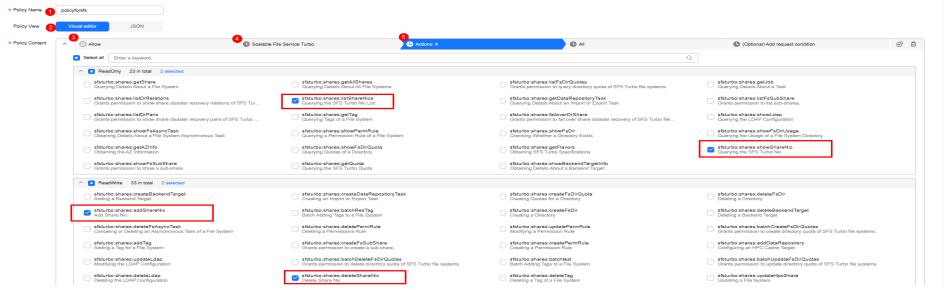
1. Log in to the ModelArts console. In the navigation pane on the left, choose **Permission Management**. The **Permission Management** page is displayed.
2. Click **Add Authorization**. On the displayed page, configure parameters.
   - **Authorized User**: Select **IAM user**, **Federated user**, **Agency**, or **All users** as required.
   - **Authorized To**: Choose an authorized object.
   - **Agency**: Select **Add agency**.
   - **Permissions**: In common mode, select **sfsturbo:shares:addShareNic**, **sfsturbo:shares:deleteShareNic**, **sfsturbo:shares:showShareNic**, and **sfsturbo:shares:listShareNics** under SFS Turbo.
3. Click **Create**. After you authorize the agency, any user with the permission can associate or disassociate SFS Turbo.

Adding Permissions to an Existing Agency for SFS Turbo

1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. In the navigation pane, choose **Permissions** > **Policies/Roles**. On the **Policies/Roles** page, click **Create Custom Policy** in the upper right corner, configure parameters, and click **OK**.
   - Policy Name: Enter a custom policy name, for example, **Authorizing ModelArts to use SFS Turbo**.
   - **Policy View**: Select **Visual editor** or **JSON**.
   - **Policy Content**: Select **Allow**. In **Select service**, search for **SFSTurbo** and select it. In **ReadOnly** under **Actions**, search for

sfsturbo:shares:showShareNic and **sfsturbo:shares:listShareNics** and select them. In **ReadWrite** under **Actions**, search for **sfsturbo:shares:addShareNic** and **sfsturbo:shares:deleteShareNic** and select them. Select **All** for **All**.

Figure 8-20 Creating a custom policy (visual editor)



The policy content in JSON view is as follows:

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sfsturbo:shares:addShareNic",
                "sfsturbo:shares:listShareNics",
                "sfsturbo:shares:deleteShareNic",
                "sfsturbo:shares:showShareNic"
            ]
        }
    ]
}
```

3. The **modelarts_agency** agency is used as an example. On the IAM console, click **Agencies**. Locate **modelarts_agency** and click **Authorize** in the **Operation** column. Choose the custom policy created in **2** and click **Next**. Then, specify the authorization scope and click **OK**.

   After you authorize the agency, any user with the permission can associate or disassociate SFS Turbo.

Verification

Log in to the ModelArts management console, choose **Dedicated Resource Pools** > **Network**. Then, click **More** in the **Operation** column of the target network, and choose **Associate SFS Turbo**. The operation is successful.

Log in to the ModelArts management console, choose **Dedicated Resource Pools** > **Network**. Then, click **More** in the **Operation** column of the target network, and choose **Disassociate SFS Turbo**. The operation is successful.

8.3.8 Assigning SFS Turbo Folder-Level Access Permissions to an IAM User

Scenario

Grant access permission of specific SFS Turbo folders to IAM users.

> **NOTE**
>
> Granting the IAM user the SFS Turbo folder-level access permission is a whitelist function. Submit a service ticket to apply for the permission as needed.

Constraints

- Ensure that you have enabled strict authorization. Log in to the ModelArts console. In the navigation pane on the left, choose **Settings**. On the **Permission Management** page, click **Enable strict authorization**.
- If the ModelArts permission for the IAM user were not configured prior to enabling strict authorization mode, the IAM user might lose access to ModelArts once that mode is activated. Configure ModelArts permissions based on the service requirements. For details, see **Dependencies and Agencies**.

Procedure

Step 1 Log in to the management console using the main account, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.

Step 2 On the IAM console, choose **Permissions** > **Policies/Roles** from the navigation pane on the left, click **Create Custom Policy** in the upper right corner, and configure the policy as follows:

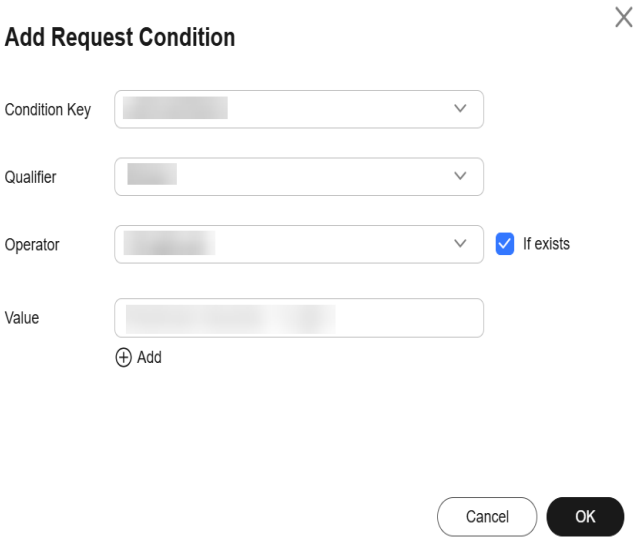
- **Policy Name**: Enter a policy name, for example, **ma_sfs_turbo**.
- **Policy View**: Select **JSON**.
- **Policy Content**: Enter the following information:

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "<modelarts_action>"
         ],
         "Condition": {
            "StringEqualsIfExists": {
               "modelarts:sfsId": [
                  "<your_ssf_id>"
               ],
               "modelarts:sfsPath": [
                  "<sfs_path>"
               ],
               "modelarts:sfsOption": [
                  "<sfs_option>"
               ]
            }
         }
      }
   ]
}
```


NOTE

- Before the preceding policies are created, all IAM users can mount to SFS Turbo by default. After you create the preceding SFS permission control policies, the IAM users who are not granted with the permission cannot mount to SFS Turbo when creating a training job on the ModelArts console, except the IAM users with the **Tenant Administrator** permission.
- Currently, you can configure only permit policies, that is, the **Effect** in the policy can only be set to **Allow**. Do not configure deny policies.
- The **Condition** parameter must use the **StringEqualsIfExists** field, and **If exists** must be enabled for the corresponding visualized view.

Figure 8-21 Enabling **If exists**



Replace *<modelarts_action>*, *<your_ssf_id>*, *<sfs_path>*, and *<sfs_option>* with actual parameters as you need. The following table describes the parameters.

Table 8-23 Parameter description

Parameter	Description

Action	Scenario in which the SFS Turbo folder access permission is granted.
	• **modelarts:notebook:create**: The permission is granted during development environment instance creation.
	• **modelarts:trainJob:create**: The permission is granted during training job creation.
	Multiple actions are supported, the following shows an example:
	`"Action": [`
	`  "modelarts:trainJob:create",`
	`  "modelarts:notebook:create"`
	`],`

Parameter	Description
modelarts:sfsId	SFS Turbo ID, which can be obtained on the SFS Turbo details page. You can enter multiple IDs, the following shows an example: "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"],
modelarts:sfsPath	Path of the SFS Turbo folder whose permissions need to be configured. You can enter multiple paths, the following shows an example: "modelarts:sfsPath": ["/path1", "/path2/path2-1"], If there are multiple SFS IDs, the SFS paths will apply to all SFS IDs. As shown in the following example, permission to access **/path1** and **/path2/path2-1** of both **0e51c7d5-d90e-475a-b5d0-ecf896da3b0d** and **2a70da1e-ea87-4ee4-ae1e-55df846e7f41** are configured. "modelarts:sfsId": ["0e51c7d5-d90e-475a-b5d0-ecf896da3b0d", "2a70da1e-ea87-4ee4-ae1e-55df846e7f41"], "modelarts:sfsPath": ["/path1", "/path2/path2-1"],

Parameter	Description

modelarts:sfsOption	Type of the access permission. The following parameters are supported:
	• **readonly**: Read-only permission
	• **readwrite**: Read and write permission. When you create a development environment instance, **modelarts:notebook:create** must be set to **readwrite**.
	To add multiple SFS options to a custom policy, add a JSON structure to Statement, the following shows an example:
	{
	"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"
	]
	}
	}
	}
	]
	}

Step 3 Create a user group and add the user to the user group. For details, see **Step 1 Creating a User Group and Adding Users to the User Group**.

Step 4 Grant a policy to the user group. On the user group list page of IAM, click **Authorize** of the target user group. The **Authorize User Group** page is displayed. Select the **ma_sfs_turbo** policy created in **Step 2**. Click **Next** and then **OK**.

Step 5 Add the **IAM ReadOnlyAccess** permission to an existing ModelArts agency.

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Permission Management**. On the displayed page, locate the target agency, choose **View Permissions** in the **Operation** column, and click **Modify permission in IAM**.
2. On the IAM console, choose **Agencies** from the navigation pane on the left, and choose **Permissions** > **Authorize**. Search for **IAM ReadOnlyAccess**, enable it, and click **Next** and **OK**.

Step 6 Verify that the permission is granted.

Log in to ModelArts as the IAM user, only the configured SFS Turbo folders are displayed during training job creation and notebook creation.

----End

9 Notebook

9.1 Migrating the Conda Environment on a Notebook Instance to an SFS Disk

This section describes how to migrate the Conda environment on a notebook instance to an SFS disk. In this way, the Conda environment will not be lost after the notebook instance is restarted.

The procedure is as follows:

1. **Creating a Virtual Environment and Saving It to the SFS Directory**
2. **Cloning the Existing Virtual Environments to the SFS Disk**
3. **Restarting the Image to Activate the Virtual Environment in the SFS Disk**
4. **Saving and Sharing the Virtual Environment**

Prerequisites

You have created a notebook instance by setting **Resource Type** to **Dedicated resource pool** and **Storage** to **SFS** and opened the terminal.

Creating a Virtual Environment and Saving It to the SFS Directory

Create a conda virtual environment.

```
# shell
conda create --prefix /home/ma-user/work/envs/user_conda/sfs-new-env python=3.7.10 -y
```

View the existing conda virtual environments. The name of the newly created virtual environment may be empty in the output.

```
# 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
```

Append the new virtual environment to conda envs.


```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
```

View the existing conda virtual environments. The new virtual environment is properly displayed, and you can switch to it by name.

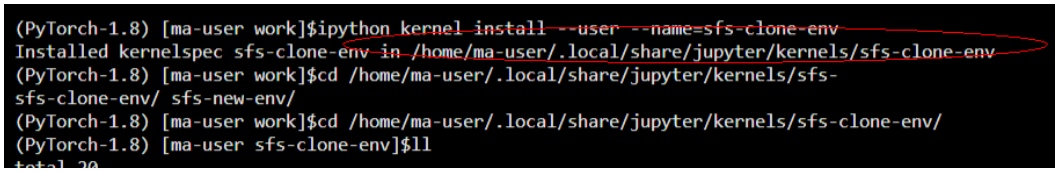
```
# shell
conda env list
conda activate sfs-new-env
# conda environments:
#
base                  /home/ma-user/anaconda3
PyTorch-1.8              /home/ma-user/anaconda3/envs/PyTorch-1.8
python-3.7.10         *  /home/ma-user/anaconda3/envs/python-3.7.10
sfs-new-env              /home/ma-user/work/envs/user_conda/sfs-new-env
```

(Optional) Register the new virtual environment with the JupyterLab kernel, so that you can directly use it in JupyterLab.

```
# 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-*
```

Note: **.local/share/jupyter/kernels/sfs-new-env** is used as an example only. Replace it with the actual installation path.

Figure 9-1 Installation path output



Refresh the JupyterLab page. The new kernel is displayed.

NOTE

After the notebook instance is restarted, the kernel needs to be registered again.

Cloning the Existing Virtual Environments to the SFS Disk

```
# 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
```


View the cloned virtual environments. If the name of the newly created virtual environment is empty, handle the issue according to **Append the new virtual environment to conda envs**.

```
# 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
```

(Optional) Register the new virtual environment with the JupyterLab kernel, so that you can directly use it in JupyterLab.

```
# 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-*
```

Note: **.local/share/jupyter/kernels/sfs-clone-env** is used as an example only. Replace it with the actual installation path.

Refresh the JupyterLab page. The new kernel is displayed.

Restarting the Image to Activate the Virtual Environment in the SFS Disk

Method 1: Use the complete conda env path.

```
# shell
conda activate /home/ma-user/work/envs/user_conda/sfs-new-env
```

Method 2: Append the virtual environment to conda envs and activate it using its name.

```
# shell
conda config --append envs_dirs /home/ma-user/work/envs/user_conda/
conda activate sfs-new-env
```

Method 3: Use Python or pip in the virtual environment.

```
# 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
```

Saving and Sharing the Virtual Environment

Package the virtual environment to be migrated.

```
# 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
```

Decompress the package to the SFS directory.

```
# 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
```

View the existing conda virtual environments.

```
# 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
```


10 Model Training

10.1 Building a Handwritten Digit Recognition Model with ModelArts Standard

This section describes how to modify a local custom algorithm to train and deploy models on ModelArts.

Scenarios

This case describes how to use PyTorch 1.8 to recognize handwritten digit images. An official MNIST dataset is used in this case.

Through this case, you can learn how to train jobs, deploy an inference model, and perform prediction on ModelArts.

Process

Before performing the following operations, complete necessary operations by referring to **Preparations**.

1. **Step 1 Prepare Training Data**: Download the MNIST dataset.
2. **Step 2: Preparing Training Files and Inference Files**: Write training and inference code.
3. **Step 3: Creating an OBS Bucket and Upload Files to OBS**: Create an OBS bucket and folder, and upload the dataset, training script, inference script, and inference configuration file to OBS.
4. **Step 4 Create a Training Job**: Train a model.
5. **Step 5 Deploying the Model for Inference**: Import the trained model to ModelArts, create a model, and deploy the model as a real-time service.
6. **Step 6 Performing Prediction**: Upload a handwritten digit image and send an inference request to obtain the inference result.
7. **Step 7 Releasing Resources**: Stop the service and delete the data in OBS to stop billing.

Preparations

- You have registered a Huawei ID and enabled Huawei Cloud services, and the account is not in arrears or frozen.
- Configure an agency.

  To use ModelArts, access to Object Storage Service (OBS), SoftWare Repository for Container (SWR), and Intelligent EdgeFabric (IEF) is required. If this is the first time you use ModelArts, configure an agency to authorize access to these services.

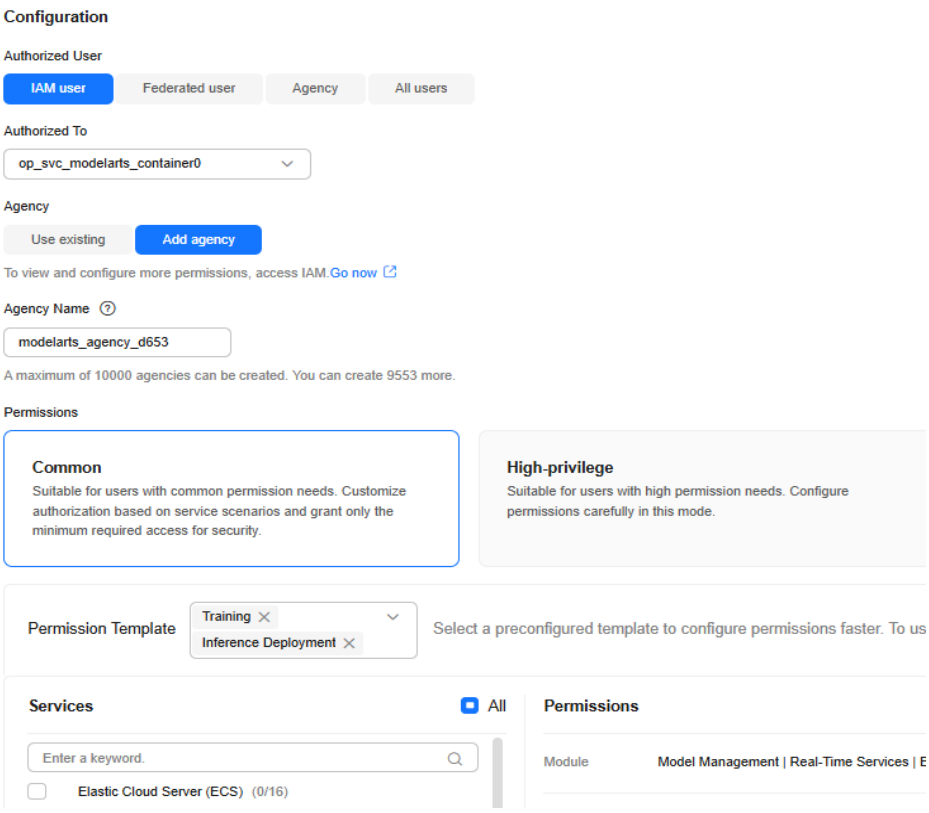
  a. Log in to the **ModelArts console** using your Huawei Cloud account. In the navigation pane on the left, choose **Permission Management**. On the **Permission Management** page, click **Add Authorization**.

  b. In the displayed dialog box, configure the following parameters:

     - **Authorized User**: **IAM user**.
     - **Agency**: Select **Add agency**.
     - **Permissions**: **Common**.
     - **Permission Template**: Select **Training** and **Inference Deployment**.
     - **Services**: The required minimum permissions are automatically selected based on the items selected in **Permission Template**.

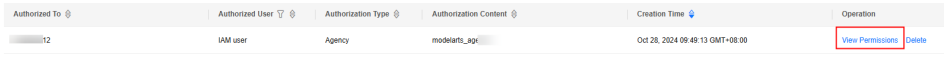
  Select "I have read and agree to the ModelArts Service Statement" and click **Create**.

Figure 10-1 Configuring an agency



c. After the configuration, view the agency configurations of your account on the **Permission Management** page.

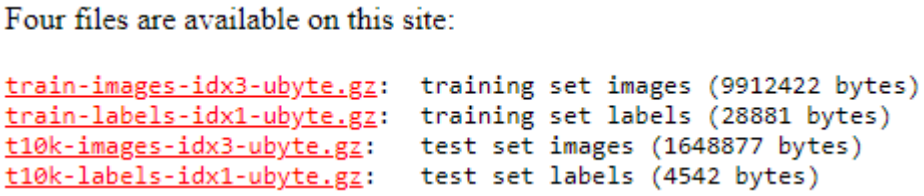
Figure 10-2 Viewing agency configurations



Step 1 Prepare Training Data

Download the MNIST dataset from a web browser. Ensure the four files in **Figure 10-3** are all downloaded.

Figure 10-3 MNIST dataset



- **train-images-idx3-ubyte.gz**: compressed package of the training set, which contains 60,000 samples.
- **train-labels-idx1-ubyte.gz**: compressed package of the training set labels, which contains the labels of the 60,000 samples
- **t10k-images-idx3-ubyte.gz**: compressed package of the validation set, which contains 10,000 samples.
- **t10k-labels-idx1-ubyte.gz**: compressed package of the validation set labels, which contains the labels of the 10,000 samples

Step 2: Preparing Training Files and Inference Files

In this case, ModelArts provides the training script, inference script, and inference configuration file.

NOTE

When pasting code from a .py file, create a .py file. Otherwise, the error message "SyntaxError: 'gbk' codec can't decode byte 0xa4 in position 324: illegal multibyte sequence" may be displayed.

Create the training script **train.py** on the local host. The content is as follows:

```
# 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


# Define a network model.
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


# Train the model. Set the model to the training mode, load the training data, calculate the loss function, and perform gradient descent.
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


# Validate the model. Set the model to the validation mode, load the validation data, and calculate the loss function and accuracy.
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)))


# The following is 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 an 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())
# The above is pytorch mnist.
# --- end


# Raw MNIST dataset processing
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():
    # Define the preset running parameters of the training job.
    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)

    # Set whether to use GPU or CPU to run the algorithm.
    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)

    # Define the data preprocessing method.
    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

    # Convert the raw MNIST dataset to a PyTorch MNIST dataset.
    convert_raw_mnist_dataset_to_pytorch_mnist_dataset(args.data_url)

    # Create a training dataset and a validation dataset.
    dataset1 = datasets.MNIST(args.data_url, train=True, download=False,
                       transform=transform)
    dataset2 = datasets.MNIST(args.data_url, train=False, download=False,
                       transform=transform)
```



```
    # Create iterators for the training dataset and the validation dataset.
    train_loader = torch.utils.data.DataLoader(dataset1, **train_kwargs)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    # Initialize the neural network model and copy the model to the compute device.
    model = Net().to(device)
    # Define the training optimizer and learning rate for gradient descent calculation.
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma=args.gamma)

    # Train the neural network and perform validation in each epoch.
    for epoch in range(1, args.epochs + 1):
        train(args, model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    # Save the model and make it adapted to the ModelArts inference model package specifications.
    if args.save_model:

        # Create the model directory in the path specified in train_url.
        model_path = os.path.join(args.train_url, 'model')
        os.makedirs(model_path, exist_ok = True)

        # Save the model to the model directory based on the ModelArts inference model package
specifications.
        torch.save(model.state_dict(), os.path.join(model_path, 'mnist_cnn.pt'))

        # Copy the inference code and configuration file to the model directory.
        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()
```

Create the inference script **customize_service.py** on the local host. The content is as follows:

```
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__)

# Define model preprocessing.
infer_transformation = transforms.Compose([
    transforms.Resize(28),
    transforms.CenterCrop(28),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# Model inference service
class PTVisionService(PTServingBaseService):

    def __init__(self, model_name, model_path):
        # Call the constructor of the parent class.
        super(PTVisionService, self).__init__(model_name, model_path)
```



```
        # Call the customized function to load the model.
        self.model = Mnist(model_path)

         # Load labels.
        self.label = [0,1,2,3,4,5,6,7,8,9]

    # Receive the request data and convert it to the input format acceptable to the model.
    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:
                    # Gray processing
                    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

    # Post-process the inference result to obtain the expected output format. The result is the returned value.
    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

    # Perform forward inference on the input data to obtain the inference result.
    def _inference(self, data):

        result = {}
        for k, v in data.items():
            result[k] = self.model(v)

        return result

# Define a network.
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):
    # Generate a network.
    model = Net()

    # Load the model.
    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 or GPU mapping
    model.to(device)

    # Turn the model to inference mode.
    model.eval()

    return model
```

Infer the configuration file **config.json** on the local host. The content is as follows:

```
{
    "model_algorithm": "image_classification",
    "model_type": "PyTorch",
    "runtime": "pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64"
}
```

Step 3: Creating an OBS Bucket and Upload Files to OBS

Upload the data, code file, inference code file, and inference configuration file obtained in the previous step to an OBS bucket. When running a training job on ModelArts, read data and code files from the OBS bucket.

1. Log in to **OBS console** and create an OBS bucket and folder.

   ```
   {OBS bucket}                    # OBS bucket name, which is customizable, for example, test-modelarts-xx
       -{OBS folder}          # OBS folder name, which is customizable, for example, pytorch
           - mnist-data       # OBS folder, which is used to store the training dataset. The folder name is customizable, for example, mnist-data.
           - mnist-code       # OBS folder, which is used to store training script train.py. The folder name is customizable, for example, mnist-code.
               - infer        # OBS folder, which is used to store inference script customize_service.py and configuration file config.json
           - mnist-output     # OBS folder, which is used to store trained models. The folder name is customizable, for example, mnist-output.
   ```

   CAUTION

   - The region where the created OBS bucket resides must be the same as that where ModelArts is used. Otherwise, the OBS bucket will be unavailable for training. For details, see **Check whether the OBS bucket and ModelArts are in the same region.**
   - When creating an OBS bucket, do not set the archive storage class. Otherwise, training models will fail.

2. Upload the MNIST dataset package obtained in **Step 1 Prepare Training Data** to the **mnist-data** folder on OBS.

CAUTION

- When uploading data to OBS, do not encrypt the data. Otherwise, the training will fail.
- Files do not need to be decompressed. Directly upload compressed packages to OBS.

3. Upload the training script **train.py** to the **mnist-code** folder.
4. Upload the inference script **customize_service.py** and inference configuration file **config.json** to the **infer** folder in **mnist-code**.

Step 4 Create a Training Job

1. Log in to the **ModelArts console** and select the same region as the OBS bucket.
2. In the navigation pane on the left, choose **Permission Management** and check whether access authorization has been configured for the current account. For details, see **Configuring Agency Authorization for ModelArts with One Click**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
3. In the navigation pane, choose **Model Training** > **Training Jobs**. On the **Training Jobs** page, click **Create Training Job**.
4. Set parameters.
   - **Algorithm Type**: Select **Custom algorithm**.
   - **Boot Mode**: Select **Preset image** and then select **PyTorch** and **pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64** for the engine its version.
   - **Code Directory**: Select the created OBS code directory, for example, **/test-modelarts-xx/pytorch/mnist-code/** (replace **test-modelarts-xx** with your OBS bucket name).
   - **Boot File**: Select the training script **train.py** uploaded to the code directory.
   - **Input**: Add one input and set its name to **data_url**. Set the data path to your OBS directory, for example, **/test-modelarts-xx/pytorch/mnist-data/** (replace **test-modelarts-xx** with your OBS bucket name).
   - **Output**: Add one output and set its name to **train_url**. Set the data path to your OBS directory, for example, **/test-modelarts-xx/pytorch/mnist-output/** (replace **test-modelarts-xx** with your OBS bucket name). Set **Predownload** to **No**.
   - **Resource Type**: Select the specifications of a single GPU.
   - Retain default settings for other parameters.

     NOTE

     The sample code runs on a single node with a single PU. If you select a flavor with multiple PUs, the training will fail.
5. Click **Submit**, confirm parameter settings for the training job, and click **Yes**.

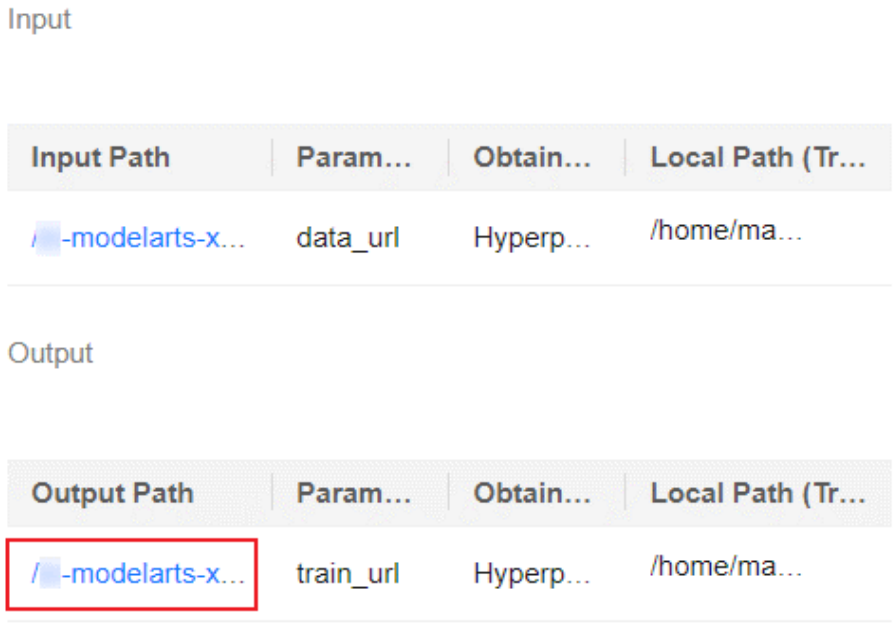
   The system automatically switches back to the **Training Jobs** page. When the training job status changes to **Completed**, the model training is completed.

NOTE

In this case, the training job will take about 10 minutes.

6. Click the training job name. On the job details page that is displayed, check whether there are error messages in logs. If so, the training failed. Identify the cause and locate the fault based on the logs.
7. In the lower left corner of the training details page, click the training output path to go to OBS, as shown in **Figure 10-4**. Then, check whether the **model** folder is available and whether there are any trained models in the folder. If there is no **model** folder or trained model, the training input may be incomplete. In this case, completely upload the training data and train the model again.

Figure 10-4 Output path



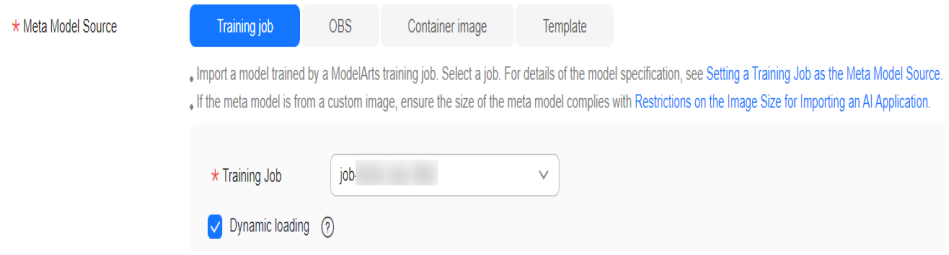
Step 5 Deploying the Model for Inference

After the model training is completed, create a model and deploy the model as a real-time service.

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Model Management**. On the displayed page, click **Create Model**.
2. On the **Create Model** page, configure the parameters and click **Create now**.

   Choose **Training Job** for **Meta Model Source**. Select the training job completed in **Step 4 Create a Training Job** from the drop-down list and enable **Dynamic loading**. The values of **AI Engine** will be automatically configured.

Figure 10-5 Meta Model Source

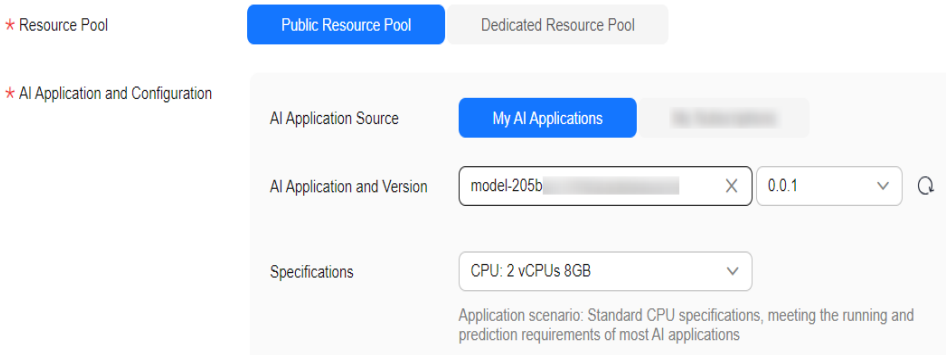


3. Wait until the model status changes to **Normal**. Then, the model is created. Locate the target model and click **Deploy** in the **Operation** column. On the displayed **Versions** page, locate the version and choose **Deploy** > **Real-Time Services** in the **Operation** column.

Figure 10-6 Deploying a real-time service

4. On the **Deploy** page, configure parameters and create a real-time service as prompted. In this example, use CPU specifications. If there are free CPU specifications, you can select them for deployment. (Each user can deploy only one real-time service for free. If you have deployed one, delete it first before deploying a new one for free.) Retain default settings for other parameters. Confirm the information and click the submit button.

Figure 10-7 Deploying a model



After you submit the service deployment request, the system automatically switches to the **Real-Time Services** page. When the service status changes to **Running**, the service has been deployed.

Step 6 Performing Prediction

1. On the **Real-Time Services** page, click the name of the real-time service. The real-time service details page is displayed.
2. Click the **Prediction** tab, set **Request Type** to **multipart/form-data**, **Request Parameter** to **image**, click **Upload** to upload a sample image, and click **Predict**.

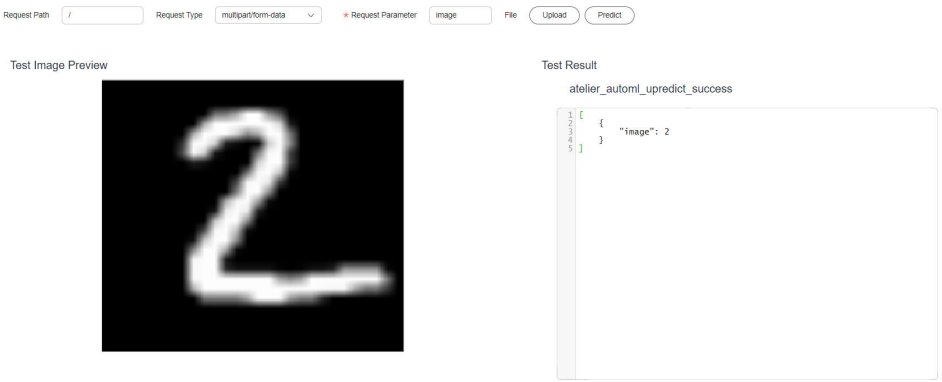
After the prediction is complete, the prediction result is displayed in the **Test Result** pane. According to the prediction result, the digit on the image is **2**.

NOTE

The MNIST used in this case is a simple dataset used for demonstration, and its algorithms are also simple neural network algorithms used for teaching. The models generated using such data and algorithms are applicable only to teaching but not to complex prediction scenarios. The prediction is accurate only if the image used for prediction is similar to the image in the training dataset (white characters on black background).

Figure 10-8 Example

Figure 10-9 Prediction results



Step 7 Releasing Resources

If you do not need to use this model and real-time service anymore, release the resources to stop billing.

- On the **Real-Time Services** page, locate the target service and click **Stop** or **Delete** in the **Operation** column.
- In the **My Model** tab, locate the target model and click **Delete** in the **Operation** column.
- On the **Training Jobs** page, delete the completed training job.
- Go to OBS and delete the OBS bucket, folders, and files used in this example.

FAQs

- Why Is a Training Job Always Queuing?

  If the training job is always queuing, the selected resources are limited in the resource pool, and the job needs to be queued. In this case, wait for resources. For details, see **Why Does a Job in ModelArts Stay in the Pending State?**

- Why Can't I Find My Created OBS Bucket After I Select an OBS Path in ModelArts?

  Ensure that the created bucket is in the same region as ModelArts. For details, see **Incorrect OBS Path on ModelArts**.

10.2 Running a Training Job on ModelArts Standard

10.2.1 Scenarios

The data volume and compute used for training vary among AI models. Select a proper storage and training solution to improve training efficiency and resource cost-effectiveness. ModelArts Standard supports multiple training scenarios to meet different requirements, including single-node single-PU, single-node multi-PU, and multi-node multi-PU.

There are public resource pools and dedicated resource pools. If you use a dedicated resource pool, the resources are not shared with other users, ensuring high efficiency. For enterprises with multiple users, use a dedicated resource pool for AI model training.

This section provides an E2E guidance to help you understand how to select a proper training solution and perform model training on ModelArts Standard.

For different data volumes and algorithms, the solutions are as follows:

- Single-node single-PU: If the data volume is small (1 GB training data) and the compute is low (one Vnt1), use OBS parallel file system to store data and code.
- Single-node multi-PU: If the data volume is medium (50 GB training data) and the compute is medium (eight Vnt1s), use SFS to store data and code.
- Multi-node multi-PU: If the data volume is large (1 TB training data) and the compute is high (four nodes with eight Vnt1s), use SFS to store data and a common OBS bucket to store code, and use distributed training.

Table 10-1 Services required in different scenarios and purchase recommendations

Scenario	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
Single-node single-PU	Pay-per-use (parallel file system)	×	Free	Free	Monthly	Free	×	Pay-per-use

Scenario	OBS	SFS	SWR	DEW	ModelArts	VPC	ECS	EVS
Single-node multi-PU	×	Monthly (HPC 500 GB)	Free	Free	Monthly	Free	Monthly (Ubuntu 18.04, at least 2 vCPUs and 8 GB memory, 100 GB local storage space, dynamic BGP with EIP, and 10 Mbit/s bandwidth)	×
Multi-node multi-PU	Pay-per-use (common OBS bucket)	Monthly (HPC 500 GB)	Free	Free	Monthly	Free	Monthly (Ubuntu 18.04, at least 2 vCPUs and 8 GB memory, 100 GB local storage space, dynamic BGP with EIP, and 10 Mbit/s bandwidth)	×

Table 10-2 Training performance of different open-source datasets

Algorithm and Data	Resource Flavor	Number of Epochs	Estimated Running Duration (hh:mm:ss)
Algorithm: PyTorch official example for ImageNet Data: ImageNet classification data subset	One node with one Vnt1	10	0:05:03
Algorithm: YOLOX Data: COCO 2017 dataset	One node with one Vnt1	10	03:33:13
	One node with eight Vnt1s	10	01:11:48
	Four nodes with eight Vnt1s	10	0:36:17
Algorithm: Swin-Transformer Data: ImageNet21K	One node with one Vnt1	10	197:25:03
	One node with eight Vnt1s	10	26:10:25
	Four nodes with eight Vnt1s	10	07:08:44

Table 10-3 Average response time for different training operations

Operation	Description	Estimated Duration
Downloading an image	Time when an image (25 GB) is downloaded for the first time	8 minutes
Scheduling resources	Duration from the time when a training job starts to be created to the time when the training job becomes running (resources are sufficient and the image is cached)	20 seconds
Accessing the training list page	Time to access the training job list page with 50 records on it	6 seconds
Loading logs	Time to load 1 MB logs on the training details page	2.5 seconds

Operation	Description	Estimated Duration
Accessing the training details page	Time to access the training details page where there are no logs	2.5 seconds
Accessing the JupyterLab page	Time to access the JupyterLab page and load the page content	0.5 seconds
Accessing the notebook list page	Time to access the notebook list page with 50 instances on it	4.5 seconds

NOTE

The preceding data is for reference only. The image download time depends on the node specifications, disk type (high I/O or common I/O), and whether SSD is used.

10.2.2 Preparations

Before using dedicated resource pools for training on ModelArts Standard, perform the following operations.

Purchasing Service Resources

Table 10-4 Purchasing service resources

Service	Description	Details
SFS Turbo	SFS charges you based on the storage capacity and usage duration you select. You can also buy a yearly or monthly package that suits your resource needs and plans. In case of arrears, you have 15 days to renew the service. Otherwise, your file system resources will be removed. You can use SFS to store data and code.	**How Do I Purchase SFS?**
SWR	SWR has two editions: Enterprise Edition and Shared Edition. SWR Shared Edition is free, metered by storage space and traffic. SWR Enterprise Edition supports pay-per-use billing mode. You can use SWR to upload custom images.	**Uploading an Image**

Service	Description	Details
OBS	OBS offers two billing modes: pay-per-use and yearly/monthly. Choose the one that suits your needs. OBS has two storage modes. For single-node and single-PU training, use a file system. For multi-node and multi-PU training, use a common OBS bucket.	• Creating a Bucket • Creating a Parallel File System
Virtual Private Cloud (VPC)	A VPC enables you to provision logically isolated, configurable, and manageable virtual networks. VPC interconnection allows you to use resources across VPCs, improving resource utilization.	Creating a VPC
Elastic Cloud Server (ECS)	ECSs are more cost-effective than physical servers. Within minutes, you can obtain ECS resources from the cloud service platform. ECS resources are flexible and on-demand. You can use ECS to mount SFS Turbo storage. **NOTE** The ECS and SFS must be in the same VPC for mounting SFS.	Purchasing an ECS in Custom Config Mode
Data Encryption Workshop (DEW)	When you use a notebook instance for code debugging, if you want to enable remote SSH development, you need DEW to select a key pair. In this way, you can log in to the ECS using the key pair, improving security. Key pairs can be created free of charge.	Creating a Key Pair

Assigning Permissions

Step 1 Configure IAM permissions.

1. Use a Huawei Cloud tenant account to create a developer user group **user_group** and add developer accounts to **user_group**. For details, see **Step 1 Create a User Group and Add Users to the User Group**.
2. Create a custom policy.
   a. Log in to the **Huawei Cloud console** as a Huawei Cloud tenant user, hover over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to go to the IAM console.
   b. In the navigation pane on the left, choose **Permissions** > **Policies/Roles**. Click **Create Custom Policy** in the upper right corner. On the displayed page, enter **Policy1** or **Policy2** for **Policy Name**, select **JSON** for **Policy View**, configure the policy content, and click **OK**.

NOTE

To define permissions required to access both global and project-level services, enclose the permissions in two separate custom policies for refined authorization. **Learn more**.

- The content of **Policy1** is as follows:

```
{
   "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"
        }
    ]
}
```

- The content of **Policy2** is as follows:

```
{
    "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"
        }
    ]
}
```

3. Grant the custom policy to the developer user group **user_group**.
   a. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **user_group**, click **Authorize** in the **Operation** column, and select **Policy1**, **Policy2**, and **SWR Admin**. Click **Next**.

      NOTE

      SoftWare Repository for Container (SWR) permissions include SWR FullAccess, SWR OperateAccess, and SWR ReadOnlyAccess. However, these permissions are available only for SWR Enterprise Edition, which has been suspended OBT. You need to select **SWR Admin**.

   b. Set the minimum authorization scope, select **All resources** for **Scope** and click **OK**.

For details about permission management, see **Basic Concepts**.

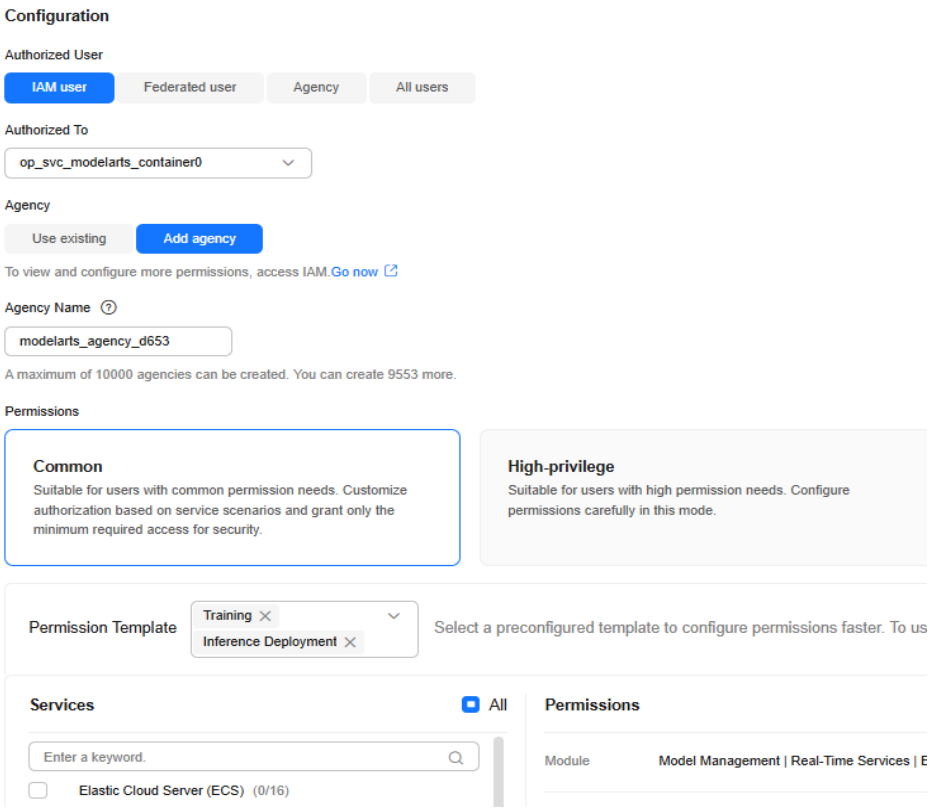
Step 2 Configure ModelArts agency permissions.

Configure ModelArts agency permissions to allow ModelArts to access dependent services such as OBS.

1. Log in to the **ModelArts console** using your Huawei Cloud account. In the navigation pane on the left, choose **Permission Management**. On the **Permission Management** page, click **Add Authorization**.
2. In the displayed dialog box, configure the following parameters:
   - **Authorized User**: **IAM user**.
   - **Agency**: Select **Add agency**.
   - **Permissions**: **Common**.
   - **Permission Template**: Select **Training** and **Inference Deployment**.
   - **Services**: The required minimum permissions are automatically selected based on the items selected in **Permission Template**.

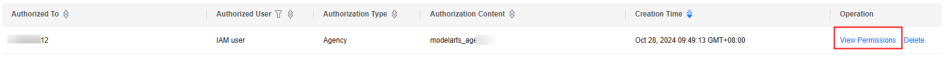
   Select "I have read and agree to the ModelArts Service Statement" and click **Create**.

   Figure 10-10 Configuring an agency



3. After the configuration, view the agency configurations of your account on the **Permission Management** page.

   Figure 10-11 Viewing agency configurations



Step 3 Configure SWR organization permissions.

Grant permissions to IAM users in an organization so that they can read, edit, and manage all images in the organization.

Only accounts and IAM users who have the **Manage** permission can add permissions for other users.

1. Log in to the **SWR console**.
2. In the navigation pane on the left, choose **Organization Management**. On the displayed page, click the target organization in the list.
3. In the **Users** tab, click **Add Permission**. In the displayed dialog box, enter an IAM username, select permissions for the user and click **OK**.

For details about SWR authorization management, see **Granting Permissions of an Organization**.

NOTE

You need to grant the SWR organization permission to the IAM user if they do not have the SWR Admin permission.

Step 4 Verify user permissions.

You may need to wait for 30 minutes before the permission settings are applied. Then, verify the configuration.

1. Log in to the **ModelArts console** as an IAM in **UserGroup-2**. On the login page, ensure that **IAM User Login** is selected.

   Change the password as prompted upon the first login.
2. Verify ModelArts permissions.
   a. In the upper left corner of the service list, choose ModelArts. The **ModelArts console** is displayed.
   b. On the ModelArts management console, check whether you can create notebook instances and training jobs, and register images.
3. Verify SFS permissions.
   a. In the upper left corner of the service list, choose SFS. The **SFS console** is displayed.
   b. Click **Create File System** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
4. Verify ECS permissions.
   a. In the upper left corner of the service list, choose ECS. The **ECS console** is displayed.
   b. Click **Buy ECS** in the upper right corner. If this operation is successful, you have obtained ECS operation permissions.
5. Verify VPC permissions.
   a. In the upper left corner of the service list, choose VPC. The **VPC console** is displayed.
   b. Click **Create VPC** in the upper right corner. If this operation is successful, you have obtained VPC operation permissions.

6. Verify DEW permissions.
   a. In the upper left corner of the service list, choose DEW. The **DEW console** is displayed.
   b. Choose **Key Pair Service** > **Private Key Pairs** and click **Create Key Pair**. If this operation is successful, you have obtained DEW operation permissions.
7. Verify OBS permissions.
   a. In the upper left corner of the service list, choose OBS. The **OBS console** is displayed.
   b. Click **Create Bucket** in the upper right corner. If this operation is successful, you have obtained OBS operation permissions.
8. Verify SWR permissions.
   a. In the upper left corner of the service list, choose SWR. The **SWR console** is displayed.
   b. If an SWR page can be properly displayed, you have obtained SWR operation permissions.
   c. Click **upload an image** in the upper right corner. If authorized organizations are displayed, you have obtained the SWR organization permission.

----End

Creating a Dedicated Resource Pool

ModelArts provides dedicated compute resources, which can be used for notebook instances, training jobs, and model deployment. The resources provided in a dedicated resource pool are exclusive, featuring higher resource efficiency than a public resource pool. To use a dedicated resource pool, create one. For details, see **Creating a Standard Dedicated Resource Pool**.

- **Network**: Select the network that has been connected to the VPC. To create a network and interconnect with VPC, see **Configuring the Standard Dedicated Resource Pool to Access the Internet**.
- **Specifications Type** and **Nodes**: Configure the values based on your needs.

Mounting an SFS Turbo File System to an ECS

After you mount an SFS Turbo file system to an ECS, you can upload the training data to SFS Turbo through the ECS. To do so, perform the following operations:

1. Check the cloud service environment.
   - The ECS and the shared SFS disk belong to the same VPC or interconnected VPCs.
   - The base image of the ECS server is Ubuntu 18.04.
   - The ECS and SFS Turbo are in the same subnet.
2. Set the Huawei Cloud image source in the ECS.
   ```
   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
   ```
3. Install the NFS client and mount the target disk.


```
sudo apt-get update
sudo apt-get install nfs-common
```

4. Obtain the command for mounting the SFS Turbo file system.
   a. Log in to the **SFS console**.
   b. In the navigation pane on the left, choose **SFS Turbo** > **File Systems**. Then, click the file system name to view its details.
   c. In the **Basic Info** tab, obtain and copy the Linux mounting command.
5. Mount the NFS storage to the ECS server.

   Ensure that the corresponding directory exists and run the following commands:

```
mkdir -p /mnt/sfs_turbo
mount -t nfs -o vers=3,nolock 192.168.0.169:/ /mnt/sfs_turbo
```

Granting the Read Permission to ModelArts Users on the ECS

When a custom image is used for training on ModelArts, the default user is **ma-user** and the default user group is **ma-group**. If a file in the ECS is called during training, grant the read permission to **ma-user**. Otherwise, error "Permission denied" will be displayed.

1. Create **ma-user** and **ma-group** in the ECS.

```
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
```

2. Check the created user information.

```
id ma-user
```

   If the following information is displayed, the creation is successful:

```
uid=1000(ma-user) gid=100(ma-group) groups=100(ma-group)
```

Installing and Configuring the OBS CLI

obsutil is a command line tool for accessing and managing OBS. You can use this tool to perform common configuration and management operations on OBS, such as creating buckets, as well as uploading, downloading, and deleting files/folders.

For details about how to install and configure obsutil, see **obsutil Quick Start**.

NOTICE

Replace the AK/SK and endpoint in the commands with the actual values.

(Optional) Configuring Workspaces

ModelArts allows you to set fine-grained permissions for IAM users and resource isolation between different workspaces. ModelArts workspaces support project resource isolation and settlement of different projects.

If you have enabled enterprise project management, you can bind an enterprise project ID when creating a workspace, add a user group to the enterprise project, and set fine-grained permissions for users in the group.

If you have not enabled enterprise project management, you can create your own workspace on ModelArts, but enterprise project features will remain unavailable.

10.2.3 Running a Single-Node Single-PU Training Job on ModelArts Standard

Process

1. Preparations
   a. **Purchasing Service Resources** (OBS and SWR)
   b. **Assigning Permissions**
   c. **Creating a Dedicated Resource Pool** (The VPC does not need to be connected.)
   d. **Installing and Configuring the OBS CLI**
   e. **(Optional) Configuring Workspaces**
2. Model training
   a. **Building and Debugging an Image Locally**
   b. **Uploading an Image**
   c. **Uploading Data and Algorithms to OBS**
   d. **Debugging Code with Notebook**
   e. **Creating a Single-Node Single-PU Training Job**
   f. **Monitoring Resources**

Building and Debugging an Image Locally

In this section, the Conda environment is packaged to set up the runtime. You can also install Conda dependencies manually using **pip install** or **conda install**.

NOTE

- The container image should be smaller than 15 GB. For details, see **Constraints on Custom Images of the Training Framework**.
- Build an image through the official open-source website, for example, PyTorch.
- Containers should be built by layer. Each layer must have no more than 1 GB of capacity or 100,000 files. You need to start with the layers that change less frequently. For example, build the OS, CUDA driver, Python, PyTorch, and other dependency packages in sequence.
- If the training data and code change frequently, do not store them in the container image in case you need to build container images frequently.
- The containers can meet the isolation requirements. Do not create conda environments in a container.

1. Export the conda environment.
   a. Start the offline container image:
   ```
   # run on terminal
   docker run -ti ${your_image:tag}
   ```
   b. Obtain **pytorch.tar.gz**:
   ```
   # run on container

   # Create a conda environment named pytorch based on the target base environment.
   conda create --name pytorch --clone base

   pip install conda-pack

   # Pack pytorch env to generate pytorch.tar.gz.
   conda pack -n pytorch -o pytorch.tar.gz
   ```
   c. Upload the package to a local path.
   ```
   # run on terminal
   docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
   ```
   d. Upload **pytorch.tar.gz** to OBS and **set it to public read**. Obtain, decompress, and clear **pytorch.tar.gz** using the **wget** commands during creation.
2. Create an image.

   Choose either the official Ubuntu 18.04 image or the image with the CUDA driver from NVIDIA as the base image. Obtain the images on the Docker Hub official website.

   To create an image, do as follows: Install the required apt package and driver, configure the **ma-user** user, import the conda environment, and configure the notebook dependency.

   NOTE

   - Creating images with a Dockerfile is recommended. This ensures Dockerfile traceability and archiving, as well as image content without redundancy or residue.
   - To reduce the final image size, delete intermediate files such as TAR packages when building each layer. For details about how to clear the cache, see **conda clean**.
3. Refer to the following example.

   Dockerfile example:
   ```
   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 already 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
```

NOTE

Replace **https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz** in the Dockerfile with the OBS path of **pytorch.tar.gz** in 1 (the file must be set to public read).

Go to the Dockerfile directory and run the following commands to create an image:


```
# Use the cd command to navigate to the directory that contains the Dockerfile and run the build command.
# docker build -t ${image_name}:${image_version} .
# Example:
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. Debug an image.

   NOTE

   It is recommended that you use a Dockerfile to include the changes made during debugging in the official container building process and test it again.

   a. Ensure that the corresponding script, code, and process are running properly on the Linux server.

      If the running fails, commission the container and then create a container image.

   b. Ensure that the image file is located correctly and that you have the required file permission.

      Before training, verify that the custom dependency package is normal, that the required packages are in the pip list, and that the Python used by the container is the correct one. (If you have multiple Pythons installed in the container image, you need to set the Python path environment variable.)

   c. Test the training boot script.

      i. Data copy and verification

         Generally, the image does not contain training data and code. You need to copy the required files to the image after starting the image. To avoid running out of disk space, store data, code, and intermediate data in the **/cache** directory. It is recommended that the Linux server have sufficient memory (more than 8 GB) and hard disk (more than 100 GB).

         The following command enables file interaction between Docker and Linux:

         ```
         docker cp data/ 39c9ceedb1f6:/cache/
         ```

         Once you have prepared the data, run the training script and verify that the training starts correctly. Generally, the boot script is as follows:

         ```
         cd /cache/code/
         python start_train.py
         ```

         To troubleshoot the training process, you can access the logs and errors in the container instance and adjust the code and environment variables accordingly.

      ii. Preset script testing

         The **run.sh** script is typically used to copy data and code from OBS to containers, and to copy output results from containers to OBS.

         You can edit and iterate the script in the container instance if the preset script does not produce the desired result.

      iii. Dedicated pool scenario

         Mounting SFS in dedicated pools allows you to import code and data without worrying about OBS operations.

You can either mount the SFS directory to the **/mnt/sfs_turbo** directory of the debugging node, or sync the directory content with the SFS disk.

To start a container instance during debugging, use the **-v** parameter to mount a directory from the host machine to the container environment.

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

The command above mounts the **/mnt/sfs_turbo** directory of the host machine to the **/sfs** directory of the container. Any changes in the corresponding directories of the host machine and container are synchronized in real time.

d. To locate faults, check the logs for the training image, and check the API response for the inference image.

Run the following command to view all stdout logs output by the container:

```
docker logs -f 39c9ceedb1f6
```

Some logs are stored in the container when creating an inference image. You need to access the container to view these logs. Note: You need to check whether the logs contain errors (including when the container is started and when the API is executed).

e. To modify the user groups of some files that are inconsistent, run the following command as the **root** user on the host machine.

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. To fix an error during debugging, edit it in the container instance. You can run the **commit** command to save the changes.

Uploading an Image

Uploading an image through the client is to run Docker commands on the machine where container engine client is installed to push the image to an image repository of SWR.

If your container engine client is an ECS or CCE node, you can push an image over two types of networks.

- If your client and the image repository are in the same region, you can push an image over private networks.
- If your client and the image repository are in different regions, you can push an image over public networks and the client needs to be bound to an EIP.

NOTE

- Each image layer uploaded through the client cannot be larger than 10 GB.
- Your container engine client version must be 1.11.2 or later.

1. Access SWR.

   a. Log in to the SWR console.

   b. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

c. In the navigation pane on the left, choose **Dashboard** and click **Generate Login Command** in the upper right corner. On the displayed page, click to copy the login command.

NOTE

- The validity period of the generated login command is 24 hours. To obtain a long-term valid login command, see **Obtaining a Long-Term Login or Image Push/Pull Command**. After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.
- The domain name at the end of the login command is the image repository address. Record the address for later use.

d. Run the login command on the machine where the container engine is installed.

The message **Login Succeeded** will be displayed upon a successful login.

2. Run the following command on the device where the container engine is installed to label the image:

docker tag *[image_name_1:tag_1] [image_repository_address]/[organization_name]/[image_name_2:tag_2]*

- *[image_name_1:tag_1]*: Replace it with the actual name and tag of the image to be uploaded.
- [image_repository_address]: You can query the address on the SWR console, that is, the domain name at the end of the login command in **1.c**.
- *[organization_name]*: Replace it with the name of the organization created.
- *[image_name_2:tag_2]*: Replace it with the desired image name and tag.

Example:

```
docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

3. Upload the image to the image repository.

docker push [*image_repository_address*]/[*organization_name*]/[*image_name_2:tag 2*]

Example:

```
docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

To view the pushed image, go to the SWR console and refresh the **My Images** page.

Uploading Data and Algorithms to OBS

- A parallel file system has been created on OBS. For details, see **Creating a Parallel File System**.
- obsutil has been installed and configured. For details, see **Installing and Configuring the OBS CLI**.

Step 1 Prepare data.

1. Download the **animal dataset** to the local PC and decompress it.

2. Use obsutil to upload the dataset to an OBS bucket.

```
./obsutil cp ./dog_cat_1w obs://${your_obs_buck}/demo/ -f -r
```

NOTE

OBS supports multiple file upload modes. If the number of files is less than 100, you can upload them on the OBS console. If the number of files is greater than 100, you are advised to use **OBS Browser+** (Windows) and **obsutil** (Linux). The preceding example shows how to use obsutil.

Step 2 Prepare an algorithm.

Upload the **main.py** file to the **demo** folder in the OBS bucket. The **main.py** file contains the following content:

```
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()
```

----End

Debugging Code with Notebook

- Notebook billing is as follows:
  - A running notebook instance will be billed based on used resources. The fees vary depending on your selected resources. For details, see **Product Pricing Details**. When a notebook instance is not used, stop it.
  - If you select EVS for storage when creating a notebook instance, the EVS disk will be continuously billed. Stop and delete the notebook instance if it is not required.
- When a notebook instance is created, auto stop is enabled by default. The notebook instance will automatically stop at the specified time.
- Only running notebook instances can be accessed or stopped.
- A maximum of 10 notebook instances can be created under one account.

Follow these steps:

1. Register the image. Log in to the ModelArts console. In the navigation pane on the left, choose **Image Management**. Click **Register**. Set **SWR Source** to the image pushed to SWR. Paste the complete SWR address, or click  to select a private image from SWR for registration, and add **GPU** to **Type**.
2. Log in to the ModelArts console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create Notebook**. On the displayed page, configure the parameters.
   a. Configure basic information of the notebook instance, including its name, description, and auto stop status. For details, see **Table 10-5**.

      Table 10-5 Basic parameters

Parameter	Description
Name	Name of a notebook instance, which can contain 1 to 64 characters, including letters, digits, hyphens (-), and underscores (_).
Description	Brief description of a notebook instance.
Auto Stop	Automatically stops the notebook instance at a specified time. This function is enabled by default. The default value is **1 hour**, indicating that the notebook instance automatically stops after running for 1 hour and its resource billing will stop then. The options are **1 hour**, **2 hours**, **4 hours**, **6 hours**, and **Custom**. You can select **Custom** to specify any integer from 1 to 24 hours.

   b. Select an image and configure resource specifications for the instance.
      - **Image**: In the **Custom Images** tab, select the uploaded custom image.

- **Resource Type**: Select a created dedicated resource pool based on site requirements.
- **Instance Specifications**: Select 1-GPU specifications.
- **Storage**: Select **EVS**.

NOTE

To use VS Code to connect to a notebook instance for code debugging, enable **Remote SSH** and select a key pair. For details, see **Connecting to a Notebook Instance Through VS Code**.

4. Click **Next**.
5. Confirm the information and click **Submit**.

   Switch to the notebook instance list. The notebook instance is being created. It will take several minutes before its status changes to **Running**.

   If the created notebook instance fails to be started, refer to the key points for debugging described in **Building and Debugging an Image Locally**.
6. In the notebook instance list, click the instance name. On the instance details page that is displayed, view the instance configuration.
7. Mounting an OBS parallel file system: On the notebook instance details page, click the **Storage** tab, then click **Mount Storage**, and set mounting parameters.
   a. Set a local mounting directory. Enter a folder name in **/data/**, for example, **demo**. The system will automatically create the folder in **/data/** of the notebook container to mount the OBS file system.
   b. Select the folder for storing the OBS parallel file system and click **OK**.
8. View the mounting result on the notebook instance details page.
9. Debug the code.

   Open your notebook instance and access a terminal, and go to the mounting directory set in Step 7.

   ```
   cd /data/demo
   ```

   Command for executing training:

   ```
   /home/ma-user/anaconda3/envs/pytorch/bin/python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
   ```

   The alarm "RequestsDependencyWarning: urllib3 (1.26.8) or chardet (5.0.0)/charset_normalizer (2.0.12) doesn't match a supported version!" does not affect training and can be ignored.

   NOTE

   After debugging in a notebook instance, if the image is modified, you can save the image for subsequent training. For details, see **Saving a Notebook Environment Image**.

Creating a Single-Node Single-PU Training Job

NOTE

For training using a dedicated pool, the mounted directory must be the same as that during debugging.

1. Log in to the ModelArts console and check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization for ModelArts with One Click**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane on the left, choose **Model Training** > **Training Jobs**. The training job list is displayed by default. Click **Create Training Job**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - **Image**: custom image you have uploaded
   - **Boot Command**:
     ```
     cd ${MA_JOB_DIR}/demo && python main.py -a resnet50 -b 128 --epochs 5 dog_cat_1w/
     ```
     demo (customizable) is the last-level directory of the OBS path.
   - **Resource Pool**: In the **Dedicated Resource Pool** tab, select a GPU dedicated resource pool.
   - **Specifications**: Select 1-GPU specifications.
4. Click **Submit**. On the information confirmation page, check the parameters, and click **OK**.
5. Wait until the training job is created.

   Once you submit the job creation request, the system handles tasks like downloading the container image and code directory, and executing the boot command in the backend. Training jobs take varying amounts of time, from tens of minutes to several hours, depending on the service logic and chosen resources.

Monitoring Resources

You can view the resource usage of compute nodes in the **Resource Usages** window. The data of at most the last three days can be displayed. When the resource usage window is opened, the data is loading and refreshed periodically.

Operation 1: If a training job uses multiple compute nodes, choose a node from the drop-down list box to view its metrics.

Operation 2: Click **cpuUsage**, **memUsage**, **npuMemUsage**, or **npuUtil** to show or hide the usage chart of that metric.

Operation 3: Hover over the graph to view the usage at the specific time.

Table 10-6 Parameters

Parameter	Description
cpuUsage	CPU usage
gpuMemUsage	GPU memory usage
gpuUtil	GPU usage
memUsage	Memory usage

Parameter	Description
npuMemUsage	NPU memory usage
npuUtil	NPU usage

10.2.4 Running a Single-Node Multi-PU Training Job on ModelArts Standard

Process

1. Preparations
   a. **Purchasing Service Resources** (VPC, SFS, SWR, and ECS)
   b. **Assigning Permissions**
   c. **Creating a Dedicated Resource Pool** (interconnecting the VPC)
   d. **Mounting an SFS Turbo File System to an ECS**
   e. **Granting the Read Permission to ModelArts Users on the ECS**
   f. **Installing and Configuring the OBS CLI**
   g. **(Optional) Configuring Workspaces**
2. Model training
   a. **Building and Debugging an Image Locally**
   b. **Uploading an Image**
   c. **Uploading Data and Algorithms to SFS**
   d. **Debugging Code with Notebook**
   e. **Creating a Single-Node Multi-PU Training Job**

Building and Debugging an Image Locally

In this section, the Conda environment is packaged to set up the runtime. You can also install Conda dependencies manually using **pip install** or **conda install**.

NOTE

- The container image should be smaller than 15 GB. For details, see **Constraints on Custom Images of the Training Framework**.
- Build an image through the official open-source website, for example, PyTorch.
- Containers should be built by layer. Each layer must have no more than 1 GB of capacity or 100,000 files. You need to start with the layers that change less frequently. For example, build the OS, CUDA driver, Python, PyTorch, and other dependency packages in sequence.
- If the training data and code change frequently, do not store them in the container image in case you need to build container images frequently.
- The containers can meet the isolation requirements. Do not create conda environments in a container.

1. Export the conda environment.

a. Start the offline container image:

```
# run on terminal
docker run -ti ${your_image:tag}
```

b. Obtain **pytorch.tar.gz**:

```
# run on container

# Create a conda environment named pytorch based on the target base environment.
conda create --name pytorch --clone base

pip install conda-pack

# Pack pytorch env to generate pytorch.tar.gz.
conda pack -n pytorch -o pytorch.tar.gz
```

c. Upload the package to a local path.

```
# run on terminal
docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
```

d. Upload **pytorch.tar.gz** to OBS and **set it to public read**. Obtain, decompress, and clear **pytorch.tar.gz** using the **wget** commands during creation.

2. Create an image.

   Choose either the official Ubuntu 18.04 image or the image with the CUDA driver from NVIDIA as the base image. Obtain the images on the Docker Hub official website.

   To create an image, do as follows: Install the required apt package and driver, configure the **ma-user** user, import the conda environment, and configure the notebook dependency.

   NOTE

   - Creating images with a Dockerfile is recommended. This ensures Dockerfile traceability and archiving, as well as image content without redundancy or residue.
   - To reduce the final image size, delete intermediate files such as TAR packages when building each layer. For details about how to clear the cache, see **conda clean**.

3. Refer to the following example.

   Dockerfile example:

```
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 already 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
```

NOTE

Replace **https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz** in the Dockerfile with the OBS path of **pytorch.tar.gz** in 1 (the file must be set to public read).

Go to the Dockerfile directory and run the following commands to create an image:

```
# Use the cd command to navigate to the directory that contains the Dockerfile and run the build
command.
# docker build -t ${image_name}:${image_version} .
# Example:
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. Debug an image.

NOTE

It is recommended that you use a Dockerfile to include the changes made during debugging in the official container building process and test it again.

a. Ensure that the corresponding script, code, and process are running properly on the Linux server.

   If the running fails, commission the container and then create a container image.

b. Ensure that the image file is located correctly and that you have the required file permission.

   Before training, verify that the custom dependency package is normal, that the required packages are in the pip list, and that the Python used by the container is the correct one. (If you have multiple Pythons installed in the container image, you need to set the Python path environment variable.)

c. Test the training boot script.

   i. Data copy and verification

      Generally, the image does not contain training data and code. You need to copy the required files to the image after starting the image. To avoid running out of disk space, store data, code, and intermediate data in the **/cache** directory. It is recommended that the Linux server have sufficient memory (more than 8 GB) and hard disk (more than 100 GB).

      The following command enables file interaction between Docker and Linux:

      ```
      docker cp data/ 39c9ceedb1f6:/cache/
      ```

      Once you have prepared the data, run the training script and verify that the training starts correctly. Generally, the boot script is as follows:

      ```
      cd /cache/code/
      python start_train.py
      ```

      To troubleshoot the training process, you can access the logs and errors in the container instance and adjust the code and environment variables accordingly.

   ii. Preset script testing

      The **run.sh** script is typically used to copy data and code from OBS to containers, and to copy output results from containers to OBS.

      You can edit and iterate the script in the container instance if the preset script does not produce the desired result.

   iii. Dedicated pool scenario

      Mounting SFS in dedicated pools allows you to import code and data without worrying about OBS operations.

      You can either mount the SFS directory to the **/mnt/sfs_turbo** directory of the debugging node, or sync the directory content with the SFS disk.

      To start a container instance during debugging, use the **-v** parameter to mount a directory from the host machine to the container environment.

      ```
      docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
      ```

      The command above mounts the **/mnt/sfs_turbo** directory of the host machine to the **/sfs** directory of the container. Any changes in the corresponding directories of the host machine and container are synchronized in real time.

d. To locate faults, check the logs for the training image, and check the API response for the inference image.

Run the following command to view all stdout logs output by the container:

```
docker logs -f 39c9ceedb1f6
```

Some logs are stored in the container when creating an inference image. You need to access the container to view these logs. Note: You need to check whether the logs contain errors (including when the container is started and when the API is executed).

e. To modify the user groups of some files that are inconsistent, run the following command as the **root** user on the host machine.

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. To fix an error during debugging, edit it in the container instance. You can run the **commit** command to save the changes.

Uploading an Image

Uploading an image through the client is to run Docker commands on the machine where container engine client is installed to push the image to an image repository of SWR.

If your container engine client is an ECS or CCE node, you can push an image over two types of networks.

- If your client and the image repository are in the same region, you can push an image over private networks.
- If your client and the image repository are in different regions, you can push an image over public networks and the client needs to be bound to an EIP.

NOTE

- Each image layer uploaded through the client cannot be larger than 10 GB.
- Your container engine client version must be 1.11.2 or later.

1. Access SWR.
   a. Log in to the **SWR console**.
   b. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.
   c. In the navigation pane on the left, choose **Dashboard** and click **Generate Login Command** in the upper right corner. On the displayed page, click to copy the login command.

      NOTE

      - The validity period of the generated login command is 24 hours. To obtain a long-term valid login command, see **Obtaining a Long-Term Login or Image Push/Pull Command**. After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.
      - The domain name at the end of the login command is the image repository address. Record the address for later use.

   d. Run the login command on the machine where the container engine is installed.

The message **Login Succeeded** will be displayed upon a successful login.

2. Run the following command on the device where the container engine is installed to label the image:

   docker tag *[image_name_1:tag_1] [image_repository_address]/[organization_name]/[image_name_2:tag_2]*

   - *[image_name_1:tag_1]*: Replace it with the actual name and tag of the image to be uploaded.
   - *[image_repository_address]*: You can query the address on the SWR console, that is, the domain name at the end of the login command in **1.c**.
   - *[organization_name]*: Replace it with the name of the organization created.
   - *[image_name_2:tag_2]*: Replace it with the desired image name and tag.

   Example:

   ```
   docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

3. Upload the image to the image repository.

   docker push [*image_repository_address*]/[*organization_name*]/[*image_name_2:tag 2*]

   Example:

   ```
   docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
   ```

   To view the pushed image, go to the SWR console and refresh the **My Images** page.

Uploading Data and Algorithms to SFS

- SFS has been mounted to the ECS. For details, see **Mounting an SFS Turbo File System to an ECS**.
- Permissions have been configured on the ECS. For details, see **Granting the Read Permission to ModelArts Users on the ECS**.
- obsutil has been installed and configured. For details, see **Installing and Configuring the OBS CLI**.

Step 1 Prepare data.

1. Visit the official COCO dataset website and download the dataset from **https://cocodataset.org/#download**.
2. Download **Train images** (18 GB), **Val images** (1 GB) and **Train/Val annotations** (241 MB) of the 2017 COCO dataset, decompress them, and save them to the **coco** folder.
3. After the download is complete, upload the data to the target directory of SFS. The dataset is very large, so you are advised to use obsutil to upload it to an OBS bucket first, and then move it to SFS.

   a. Run the following commands on the local host and use obsutil to upload the local dataset to an OBS bucket.

   ```
   # Upload local data to OBS.
   # ./obsutil cp ${Path of the local folder where the dataset is stored} ${Path of the OBS folder where the dataset is stored} -f -r
   # Example:
   ./obsutil cp ./coco obs://your_bucket/ -f -r
   ```


b. Log in to the ECS and use obsutil to migrate the dataset to SFS. The sample code is as follows:

```
# Transfer OBS data to SFS.
# ./obsutil cp ${Path of the OBS folder where the dataset is located} ${Path of the SFS folder} -f -r
# Example:
./obsutil cp obs://your_bucket/coco/ /mnt/sfs_turbo/ -f -r
```

The directory structure in the **/mnt/sfs_turbo/coco** folder is as follows:

```
coco
|---annotations
|---train2017
|---val2017
```

For more obsutil operations, see **Introduction to obsutil**.

c. Set the file owner to **ma-user**.

```
chown -R ma-user:ma-group coco
```

Step 2 Prepare an algorithm.

1. Download the YOLOX code. Code repository address: **https://github.com/Megvii-BaseDetection/YOLOX.git**.

```
git clone https://github.com/Megvii-BaseDetection/YOLOX.git
cd YOLOX
git checkout 4f8f1d79c8b8e530495b5f183280bab99869e845
```

2. Change the onnx version in the **requirements.txt** file to at least 1.12.0.
3. Change **data_dir = os.path.join(get_yolox_datadir(), "COCO")** in line 59 of **yolox/data/datasets/coco.py** to **data_dir = '/home/ma-user/coco'**.

```
# data_dir = os.path.join(get_yolox_datadir(), "COCO")
data_dir = '/home/ma-user/coco'
```

4. Add the following code before line 13 in **tools/train.py**:

```
# Add the two lines of code to avoid the yolox module not found error during execution.
import sys
sys.path.append(os.getcwd())

# line13
from yolox.core import launch
from yolox.exp import Exp, get_exp
```

5. Change **fast_cocoeval** to **fast_coco_eval_api** in line 122 of **jit_ops.py** in **yolox/layers**.

```
# def __init__(self, name="fast_cocoeval"):
def __init__(self, name="fast_coco_eval_api"):
```

6. Change **from yolox.layers import COCOeval_opt as COCOeval** to **from pycocotools.cocoeval import COCOeval** in line 294 of **coco_evaluator.py** in **yolox\evaluators**.

```
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. Create a **run.sh** script in the **tools** directory and use it as the boot script. The following is a reference for the **run.sh** script:

```
#!/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
```

NOTE

Some environment variables do not exist in the notebook environment. Default values are required.

8. Save the code to OBS and upload the code to the target directory of SFS through OBS.

   a. Run the following commands on the local host and use obsutil to upload the local dataset to an OBS bucket.

   ```
   # Upload local code to OBS.
   ./obsutil cp ./YOLOX obs://your_bucket/ -f -r
   ```

   b. Log in to the ECS and use obsutil to migrate the dataset to SFS. The sample code is as follows:

   ```
   # Upload code from OBS to SFS.
   ./obsutil cp obs://your_bucket/YOLOX/ /mnt/sfs_turbo/code/ -f -r
   ```

   NOTE

   In this example, obsutils is used to upload files. You can also use SCP to upload files. For details, see **How Can I Use SCP to Transfer Files Between a Local Linux Computer and a Linux ECS?**

9. Set the file owner to **ma-user** in SFS.

   ```
   chown -R ma-user:ma-group YOLOX
   ```

10. Remove **\r** from the shell script:

    ```
    cd YOLOX
    sed -i 's/\r//' run.sh
    ```

    NOTE

    Shell scripts written in Windows have **\r\n** as the line ending, but Linux uses **\n** as the line ending. This means that Linux treats **\r** as part of the script and shows the error message "$'\r': command not found" when running it. To fix this, you need to remove **\r** from the shell script.

----End

Debugging Code with Notebook

- Notebook billing is as follows:

  - A running notebook instance will be billed based on used resources. The fees vary depending on your selected resources. For details, see **Product Pricing Details**. When a notebook instance is not used, stop it.
  - If you select EVS for storage when creating a notebook instance, the EVS disk will be continuously billed. Stop and delete the notebook instance if it is not required.
- When a notebook instance is created, auto stop is enabled by default. The notebook instance will automatically stop at the specified time.
- Only running notebook instances can be accessed or stopped.
- A maximum of 10 notebook instances can be created under one account.

Follow these steps:

1. Register the image. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Image Management**. Click **Register**. Set **SWR Source** to the image pushed to SWR. Paste the complete SWR address, or click  to select a private image from SWR for registration, and add **GPU** to **Type**.
2. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create Notebook**. On the displayed page, configure the parameters.
   a. Configure basic information of the notebook instance, including its name, description, and auto stop status. For details, see **Table 10-7**.

Table 10-7 Basic parameters

Parameter	Description
Name	Name of a notebook instance, which can contain 1 to 64 characters, including letters, digits, hyphens (-), and underscores (_).
Description	Brief description of a notebook instance.
Auto Stop	Automatically stops the notebook instance at a specified time. This function is enabled by default. The default value is **1 hour**, indicating that the notebook instance automatically stops after running for 1 hour and its resource billing will stop then. The options are **1 hour**, **2 hours**, **4 hours**, **6 hours**, and **Custom**. You can select **Custom** to specify any integer from 1 to 24 hours.

   b. Select an image and configure resource specifications for the instance.
      - **Image**: In the **Custom Images** tab, select the uploaded custom image.
      - **Resource Type**: Select a created dedicated resource pool based on site requirements.

- **Instance Specifications**: Select 8-GPUs specifications. This matches the default setting of **MA_NUM_GPUS** in the **run.sh** file, which is 8 PUs.
- **Storage**: Select **SFS**. **Mounted Subdirectory** is optional.

NOTE

To use VS Code to connect to a notebook instance for code debugging, enable **Remote SSH** and select a key pair. For details, see **Connecting to a Notebook Instance Through VS Code**.

4. Click **Next**.
5. Confirm the information and click **Submit**.

   Switch to the notebook instance list. The notebook instance is being created. It will take several minutes before its status changes to **Running**.
6. In the notebook instance list, click the instance name. On the instance details page that is displayed, view the instance configuration.
7. Open a terminal in notebook and enter the boot command to debug the code.

```
# Create a dataset soft link.
# ln -s /home/ma-user/work/${Path of the COCO dataset on SFS} /home/ma-user/coco
# Go to the corresponding directory.
# cd /home/ma-user/work/${YOLOX path on SFS}
# Install the environment and run the script.
# /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh tools/run.sh

# Example:
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
```

NOTE

After debugging in a notebook instance, if the image is modified, you can save the image for subsequent training. For details, see **Saving a Notebook Environment Image**.

Creating a Single-Node Multi-PU Training Job

1. Log in to the **ModelArts console** and check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization for ModelArts with One Click**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane on the left, choose **Model Training** > **Training Jobs**. The training job list is displayed by default. Click **Create Training Job**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - **Image**: custom image you have uploaded
   - **Boot Command**:

     ```
     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
     ```
   - **Resource Pool**: In the **Dedicated Resource Pool** tab, select a GPU dedicated resource pool.

- **Specifications**: Select 8-GPU specifications.
- **Compute Nodes**: Enter **1**.
- **SFS Turbo**: Add the mount configuration and select the SFS name. The cloud mount path is **/home/ma-user/work**.

  NOTE

  To ensure that the code path and boot command are the same as those for notebook debugging, set the cloud mount path to **/home/ma-user/work**.

4. Click **Submit**. On the information confirmation page, check the parameters, and click **OK**.
5. Wait until the training job is created.

   Once you submit the job creation request, the system handles tasks like downloading the container image and code directory, and executing the boot command in the backend. Training jobs take varying amounts of time, from tens of minutes to several hours, depending on the service logic and chosen resources.

10.2.5 Running a Multi-Node Multi-PU Training Job on ModelArts Standard

Process

1. Preparations
   a. **Purchasing Service Resources** (VPC, SFS, OBS, SWR, and ECS)
   b. **Assigning Permissions**
   c. **Creating a Dedicated Resource Pool** (connecting to the VPC)
   d. **Mounting an SFS Turbo File System to an ECS**
   e. **Granting the Read Permission to ModelArts Users on the ECS**
   f. **Installing and Configuring the OBS CLI**
   g. **(Optional) Configuring Workspaces**
2. Model training
   a. **Building and Debugging an Image Locally**
   b. **Uploading an Image**
   c. **Uploading data to OBS**
   d. **Uploading the Algorithm to SFS**
   e. **Debugging Code with Notebook**
   f. **Creating a Multi-Node Multi-PU Training Job**

Building and Debugging an Image Locally

In this section, conda env is packaged to build the environment. You can also install conda environment dependencies using **pip install** or **conda install**.

NOTE

- The container image should be smaller than 15 GB. For details, see **Constraints on Custom Images of the Training Framework**.
- Build an image through the official open-source website, for example, PyTorch.
- Containers should be built by layer. Each layer must not have more than 1 GB of capacity or 100,000 files. You need to start with the layers that change less frequently. For example, build the OS, CUDA driver, Python, PyTorch, and other dependency packages in sequence.
- If the training data and code change frequently, do not store them in the container image in case you need to build container images frequently.
- The containers can meet the isolation requirements. Do not create conda environments in a container.

1. Export the conda environment.
   a. Start the offline container image.
      ```
      # run on terminal
      docker run -ti ${your_image:tag}
      ```
   b. Obtain **pytorch.tar.gz**.
      ```
      # run on container

      # Create a conda environment named pytorch based on the target base environment.
      conda create --name pytorch --clone base

      pip install conda-pack

      # Pack pytorch env to generate pytorch.tar.gz.
      conda pack -n pytorch -o pytorch.tar.gz
      ```
   c. Upload the package to a local path.
      ```
      # run on terminal
      docker cp ${your_container_id}:/xxx/xxx/pytorch.tar.gz .
      ```
   d. Upload **pytorch.tar.gz** to OBS and **set it to public read**. Obtain, decompress, and clear **pytorch.tar.gz** using the **wget** commands during creation.
2. Create an image.

   Choose either the official Ubuntu 18.04 image or the image with the CUDA driver from NVIDIA as the base image. Obtain the images on the Docker Hub official website.

   To create an image, do as follows: Install the required apt package and driver, configure the **ma-user** user, import the conda environment, and configure the notebook dependency.

   NOTE

   - Creating images with a Dockerfile is recommended. This ensures Dockerfile traceability and archiving, as well as image content without redundancy or residue.
   - To reduce the final image size, delete intermediate files such as TAR packages when building each layer. For details about how to clear the cache, see **conda clean**.
3. Refer to the following example.

   Dockerfile example:
   ```
   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 already 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
```

NOTE

Replace **https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/pytorch.tar.gz** in the Dockerfile with the OBS path of **pytorch.tar.gz** in 1 (the file must be set to public read).

Go to the Dockerfile directory and run the following commands to create an image:


```
# Use the cd command to navigate to the directory that contains the Dockerfile and run the build command.
# docker build -t ${image_name}:${image_version} .
# For example:
docker build -t pytorch-1.13-cuda11.3-cudnn8-ubuntu18.04:v1 .
```

4. Debug an image.

   NOTE

   It is recommended that you use a Dockerfile to include the changes made during debugging in the official container building process and test it again.

   a. Ensure that the corresponding script, code, and process are running properly on the Linux server.

      If the running fails, commission the container and then create a container image.

   b. Ensure that the image file is located correctly and that you have the required file permission.

      Before training, verify that the custom dependency package is normal, that the required packages are in the pip list, and that the Python used by the container is the correct one. (If you have multiple Pythons installed in the container image, you need to set the Python path environment variable.)

   c. Test the training boot script.

      i. Data copy and verification

         Generally, the image does not contain training data and code. You need to copy the required files to the image after starting the image. To avoid running out of disk space, store data, code, and intermediate data in the **/cache** directory. It is recommended that the Linux server have sufficient memory (more than 8 GB) and hard disk (more than 100 GB).

         The following command enables file interaction between Docker and Linux:

         ```
         docker cp data/ 39c9ceedb1f6:/cache/
         ```

         Once you have prepared the data, run the training script and verify that the training starts correctly. Generally, the boot script is as follows:

         ```
         cd /cache/code/
         python start_train.py
         ```

         To troubleshoot the training process, you can access the logs and errors in the container instance and adjust the code and environment variables accordingly.

      ii. Preset script testing

         The **run.sh** script is typically used to copy data and code from OBS to containers, and to copy output results from containers to OBS. For details about how to build **run.sh**, see **Running a Training Job on ModelArts Standard**.

         You can edit and iterate the script in the container instance if the preset script does not produce the desired result.

      iii. Dedicated pool scenario

         Mounting SFS in dedicated pools allows you to import code and data without worrying about OBS operations.

You can either mount the SFS directory to the **/mnt/sfs_turbo** directory of the debugging node, or sync the directory content with the SFS disk.

To start a container instance during debugging, use the **-v** parameter to mount a directory from the host machine to the container environment.

```
docker run -ti -d -v /mnt/sfs_turbo:/sfs my_deeplearning_image:v1
```

The command above mounts the **/mnt/sfs_turbo** directory of the host machine to the **/sfs** directory of the container. Any changes in the corresponding directories of the host machine and container are synchronized in real time.

d. To locate faults, check the logs for the training image, and check the API response for the inference image.

Run the following command to view all stdout logs output by the container:

```
docker logs -f 39c9ceedb1f6
```

Some logs are stored in the container when creating an inference image. You need to access the container to view these logs. Note: You need to check whether the logs contain errors (including when the container is started and when the API is executed).

e. To modify the user groups of some files that are inconsistent, run the following command as the **root** user on the host machine.

```
docker exec -u root:root 39c9ceedb1f6 bash -c "chown -R ma-user:ma-user /cache"
```

f. To fix an error during debugging, edit it in the container instance. You can run the **commit** command to save the changes.

Uploading an Image

Uploading an image through the client is to run Docker commands on the machine where container engine client is installed to push the image to an image repository of SWR.

If your container engine client is an ECS or CCE node, you can push an image over two types of networks.

- If your client and the image repository are in the same region, you can push an image over private networks.
- If your client and the image repository are in different regions, you can push an image over public networks and the client needs to be bound to an EIP.

NOTE

- Each image layer uploaded through the client cannot be larger than 10 GB.
- Your container engine client version must be 1.11.2 or later.

1. Access SWR.
   a. Log in to the **SWR console**.
   b. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

c. In the navigation pane on the left, choose **Dashboard** and click **Generate Login Command** in the upper right corner. On the displayed page, click to copy the login command.

NOTE

- The validity period of the generated login command is 24 hours. To obtain a long-term valid login command, see **Obtaining a Login Command with Long-Term Validity**. After you obtain a long-term valid login command, your temporary login commands will still be valid as long as they are in their validity periods.
- The domain name at the end of the login command is the image repository address. Record the address for later use.

d. Run the login command on the machine where the container engine is installed.

The message **Login Succeeded** will be displayed upon a successful login.

2. Run the following command on the device where the container engine is installed to label the image:

docker tag *[Image name 1:tag 1] [Image repository address]/[Organization name]/[Image name 2:tag 2]*

- [Image name 1:tag 1]: Replace it with the actual name and tag of the image to be uploaded.
- [Image repository address]: You can query the address on the SWR console, that is, the domain name at the end of the login command in **1.c**.
- [Organization name]: Replace it with the name of the organization created.
- [Image name 2:tag 2]: Replace it with the desired image name and tag.

Example:

```
docker tag ${image_name}:${image_version} swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

3. Run the following command to push the image to the image repository:

docker push [*Image repository address*]/[*Organization name*]/[*Image name 2:tag 2*]

Example:

```
docker push swr.cn-north-4.myhuaweicloud.com/${organization_name}/${image_name}:${image_version}
```

To view the pushed image, go to the SWR console and refresh the **My Images** page.

Uploading data to OBS

- A common OBS bucket has been created. For details, see **Creating a Bucket**.
- obsutil has been installed. For details, see **Installing and Configuring the OBS CLI**.
- For details about the data transmission principle between OBS and the training container, see **Running a Training Job on ModelArts Standard**.

1. Visit the official ImageNet website and download the ImageNet-21K dataset from **http://image-net.org/**.

2. After converting the format, download the annotation files: **ILSVRC2021winner21k_whole_map_train.txt** and **ILSVRC2021winner21k_whole_map_val.txt**.
3. Upload the downloaded files to the **imagenet21k_whole** folder in the OBS bucket. For details about how to upload files to an OBS bucket, see **Uploading Data and Algorithms to OBS**.

Uploading the Algorithm to SFS

1. Download the Swin-Transformer code.
   ```
   git clone --recursive https://github.com/microsoft/Swin-Transformer.git
   ```
2. Comment out line 27 **t_mul=1** of the **lr_scheduler.py** file.
3. Comment out line 28 **print("ERROR IMG LOADED: ", path)** of the **imagenet22k_dataset.py** file in the **data** folder.
4. Change **prefix = 'ILSVRC2011fall_whole'** to **prefix = 'ILSVRC2021winner21k_whole'** in line 112 of the **build.py** file in the **data** folder.
5. Create the **requirements.txt** file in the **Swin-Transformer** directory to specify the Python dependency library.
   ```
   # The content of the requirements.txt file is as follows:

   timm==0.4.12
   termcolor==1.1.0
   yacs==0.1.8
   ```
6. Prepare the OBS paths required in the **run.sh** file.
   a. Prepare the URL for sharing the ImageNet dataset.

      Select the **imagenet21k_whole** dataset folder to be shared, click the share button, select the validity period of the share URL, enter the access code, for example, **123456**, click **Copy Link**, and record the URL.

   b. Prepare the **obsutil_linux_amd64.tar.gz** share URL.

      Download **obsutil_linux_amd64.tar.gz** by referring to **Downloading and Installing obsutil**, upload it to the OBS bucket, and set it to public read. Click **Object Properties** and copy the URL.

      Sample link:

      ```
      https://${bucketname_name}.obs.cn-north-4.myhuaweicloud.com/${folders_name}/pytorch.tar.gz
      ```
7. In the **Swin-Transformer** directory, create the **run.sh** boot script.

   NOTE

   - Replace **SRC_DATA_PATH=${***URL for sharing the ImageNet dataset in OBS***}** in the script with the share URL of the **imagenet21k_whole** folder in the previous step.
   - Replace **https://${bucket_name}.obs.cn-north-4.myhuaweicloud.com/${folder_name}/obsutil_linux_amd64.tar.gz** in the script with the OBS path of **obsutil_linux_amd64.tar.gz** in the previous step (the file must be set to public read).

   Boot script for single-node single-PU training:

   ```
   # Create a run.sh script in the home directory of the code. The script content is as follows:

   #!/bin/bash

   # Download data from OBS to the local SSD.
   DIS_DATA_PATH=/cache
   SRC_DATA_PATH=${URL for sharing the ImageNet dataset in OBS}
   ```



```
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

# Path for storing the model weights and training configurations during model training
OUTPUT_PATH=/cache/output

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 --output $OUTPUT_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml --local_rank 0
```

Boot script for multi-node multi-PU training:

```
# Create a run.sh script.

#!/bin/bash

# Download data from OBS to the local SSD.
DIS_DATA_PATH=/cache
SRC_DATA_PATH=${URL for sharing the ImageNet dataset in OBS}
OBSUTIL_PATH=https://${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

# Path for storing the model weights and training configurations during model training
OUTPUT_PATH=/cache/output

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 $IMAGE_DATA_PATH --output=$OUTPUT_PATH --cfg ./configs/swin/swin_base_patch4_window7_224_22k.yaml
```

NOTE

- You are advised to use the boot script for single-node single-PU training. After the script runs properly, use the boot script for multi-node multi-PU training.
- **VC_WORKER_HOSTS**, **VC_WORKER_NUM**, **VC_TASK_INDEX**, and **MA_NUM_GPUS** in **run.sh** for multi-node multi-PU training are environment variables preset in the ModelArts training container. For details about environment variables of a training container, see **Viewing Environment Variables of a Training Container**.
- **OUTPUT_PATH** in **run.sh** is the path for storing intermediate results such as model weights and training configurations during training. If **config.TRAIN.AUTO_RESUME** in the training script is set to **True** (default value), the latest model weight will be automatically loaded in the **OUTPUT_PATH** directory during training.

8. Save the code folder to OBS through obsutils and upload the code to the target directory of SFS through OBS.
9. In SFS, set the owner of the code file **Swin-Transformer-main** to **ma-user**.
   ```
   chown -R ma-user:ma-group Swin-Transformer
   ```
10. Run the following command to remove **\r** from the shell script:
    ```
    cd Swin-Transformer
    sed -i 's/\r//' run.sh
    ```


> **NOTE**
>
> Shell scripts written in Windows have **\r\n** as the line ending, but Linux uses **\n** as the line ending. This means that Linux treats **\r** as part of the script and shows the error message "$'\r': command not found" when running it. To fix this, you need to remove **\r** from the shell script.

Debugging Code with Notebook

A notebook instance includes a **/cache** directory limited to 500 GB. Exceeding this limit causes the instance to restart. Since the ImageNet dataset surpasses 500 GB, consider using offline resources or a smaller portion for debugging within a notebook instance. (For details about debugging in a notebook instance, see **Debugging Code with Notebook**.)

Creating a Multi-Node Multi-PU Training Job

1. Log in to the **ModelArts console** and check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization for ModelArts with One Click**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane on the left, choose **Model Training** > **Training Jobs**. The training job list is displayed by default.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Algorithm Type**: **Custom algorithm**
   - **Boot Mode**: **Custom image**
   - **Image**: Select the uploaded custom image.
   - **Boot Command**:
     ```
     cd /home/ma-user/work/code/Swin-Transformer && /home/ma-user/anaconda3/envs/pytorch/bin/pip install -r requirements.txt && /bin/sh run.sh
     ```
   - **Auto Restart**: Enable this function and set **Restarts**. In this way, when a node is faulty, the platform automatically restarts the job and isolates the faulty node.

     If this function is enabled, you need to configure the output path in the **run.sh** script. In this way, when a fault occurs, the training can be continued based on the model weight saved in the previous round, minimizing resource waste.
   - **Resource Pool**: In the **Dedicated Resource Pool** tab, select dedicated resource pool of the GPU flavor.
   - **Specifications**: Select required GPU specifications.
   - **Compute Nodes**: Select the number of required nodes.
   - **SFS Turbo**: Add the mount configuration and select the SFS name. The cloud mount path is **/home/ma-user/work**.

     > **NOTE**
     >
     > To ensure that the code path and boot command are the same as those for notebook debugging, set the cloud mount path to **/home/ma-user/work**.
4. Click **Submit**. On the information confirmation page, check the parameters, and click **OK**.

5. Wait until the training job is created.

   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, depending on the service logic and selected resources.

11 Model Inference

11.1 Enabling a ModelArts Standard Inference Service to Access the Internet

This section describes how to enable an inference service to access the Internet.

Applications

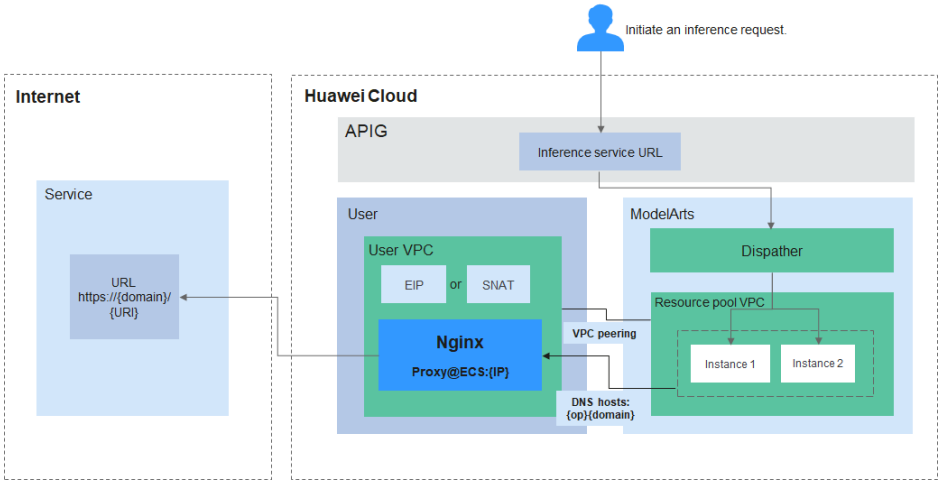
An inference service accesses the Internet in the following scenarios:

- After an image is input, the inference service calls OCR on the Internet and then processes data using NLP.
- The inference service downloads files from the Internet and analyzes the files.
- The inference service sends back the analysis result to the terminal on the Internet.

Solution Design

Use the algorithm on the instance where the inference service is deployed to access the Internet.

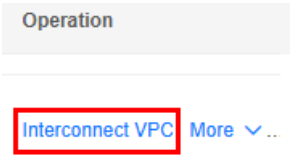
Figure 11-1 Networking for an inference service to access the Internet



Step 1: Interconnecting a ModelArts Dedicated Resource Pool to a VPC

1. Create a VPC and subnet first. For details, see **Creating a VPC and Subnet**.
2. Create a ModelArts dedicated resource pool network.
   a. Log in to the ModelArts console. In the navigation pane on the left, choose **Network** under **Resource Management**.
   b. Click **Create Network**.
   c. In the displayed **Create Network** dialog box, configure the parameters.
   d. Confirm the settings and click **OK**.
3. Interconnect the dedicated resource pool to the VPC.
   a. On the ModelArts console, choose **Network** under **Resource Management** from the navigation pane.
   b. Locate the created network and click **Interconnect VPC** in the **Operation** column.

      Figure 11-2 Interconnecting the VPC

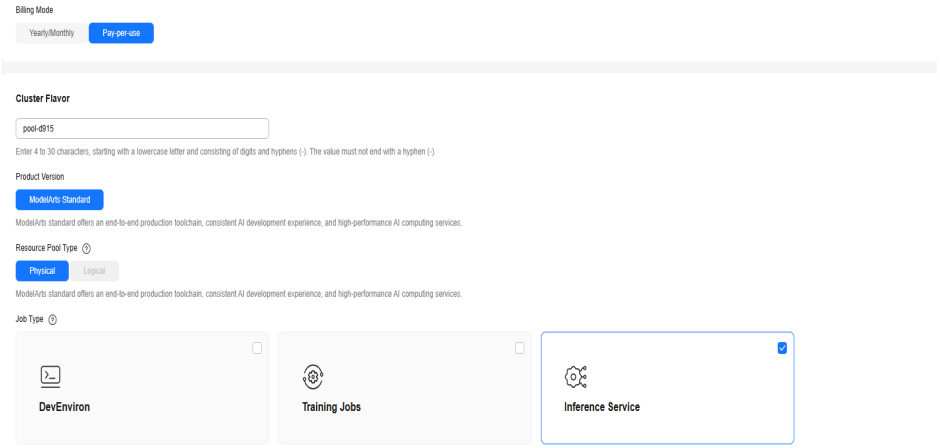
   c. In the displayed dialog box, enable **Interconnect VPC**, and choose the created VPC and subnet from the drop-down lists.

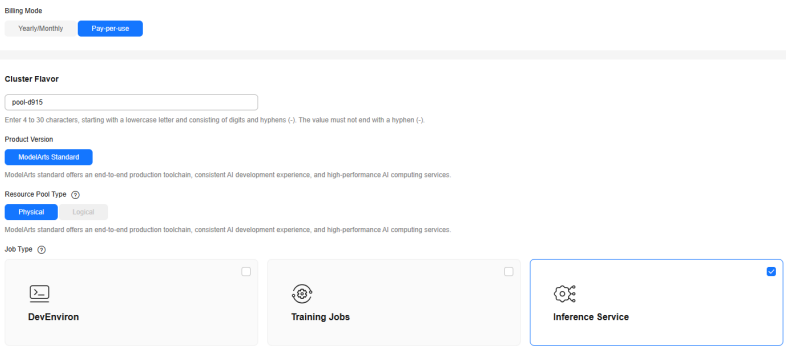
      NOTE

      The peer network to be interconnected cannot overlap with the current CIDR block.
4. Create a ModelArts dedicated resource pool.
   a. On the ModelArts console, choose **Standard Cluster** under **Resource Management** from the navigation pane.
   b. Click **Buy Standard Cluster** and configure the parameters on the displayed page.

      Job Type includes **Inference Service**. For **Network**, choose the network that has been interconnected to the VPC.

      Figure 11-3 Job types



c. Click Buy Now. Confirm the information and click **Submit**.

Step 2: Using Docker to Install and Configure a Forward Proxy

1. Purchase an ECS. For details, see **Purchasing an ECS**. You can select the latest Ubuntu image and the created VPC.
2. Assign an EIP. For details, see **Assigning an EIP**.
3. Bind the EIP to the ECS. For details, see **Binding an EIP to an Instance**.
4. Log in to the ECS and run the following command to install Docker. If it has been installed, skip this step.
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
5. Install the Squid container.
   ```
   docker pull ubuntu/squid
   ```
6. Create a host directory.
   ```
   mkdir –p /etc/squid/
   ```
7. Open and configure the **whitelist.conf** file.
   ```
   vim whitelist.conf
   ```
   The configuration content is the addresses that can be accessed by security control. Wildcard characters can be configured. For example:
   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```
   NOTE

   If the address cannot be accessed, configure the access domain name in the browser.
8. Open and configure the **squid.conf** file.
   ```
   vim squid.conf
   ```
   The configuration details are as follows:
   ```
   # 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
   ```
9. Set the permissions on the host directory and configuration files as follows:
   ```
   chmod 640 -R /etc/squid
   ```
10. Start the Squid instance.
    ```
    docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
    ```
11. Go to Docker and refresh Squid.


```
docker exec –it squid bash
root@{container_id}:/# squid -k reconfigure
```

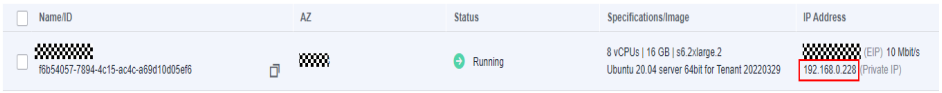
Step 3: Setting the DNS Proxy and Calling a Public IP Address

1. When you customize a model image, set the proxy to the private IP address and port of the proxy server.

   ```
   proxies = {
     "http": "http://{proxy_server_private_ip}:3128",
     "https": "http://{proxy_server_private_ip}:3128"
   }
   ```

   The IP address of the proxy service is the private IP address of the ECS created in **Step 2: Using Docker to Install and Configure a Forward Proxy**. For details about how to obtain the IP address, see **Viewing ECS Details (List View)**.

   Figure 11-4 Private IP address

2. When you call the public IP address, use the service URL to send the service request. For example:

   ```
   https://e8a048ce25136addbbac23ce6132a.apig.cn-east-3.huaweicloudapis.com
   ```

11.2 E2E O&M Solution of ModelArts Inference Services

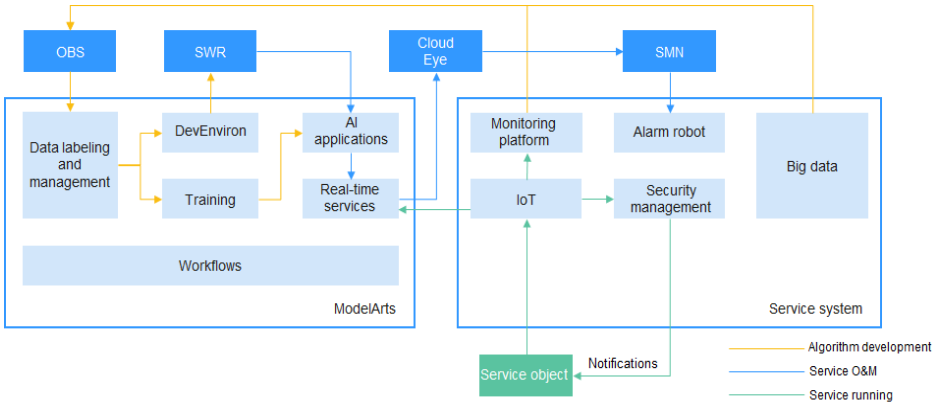
The end-to-end O&M of ModelArts inference services involves the entire AI process including algorithm development, service O&M, and service running.

Overview

End-to-End O&M Process

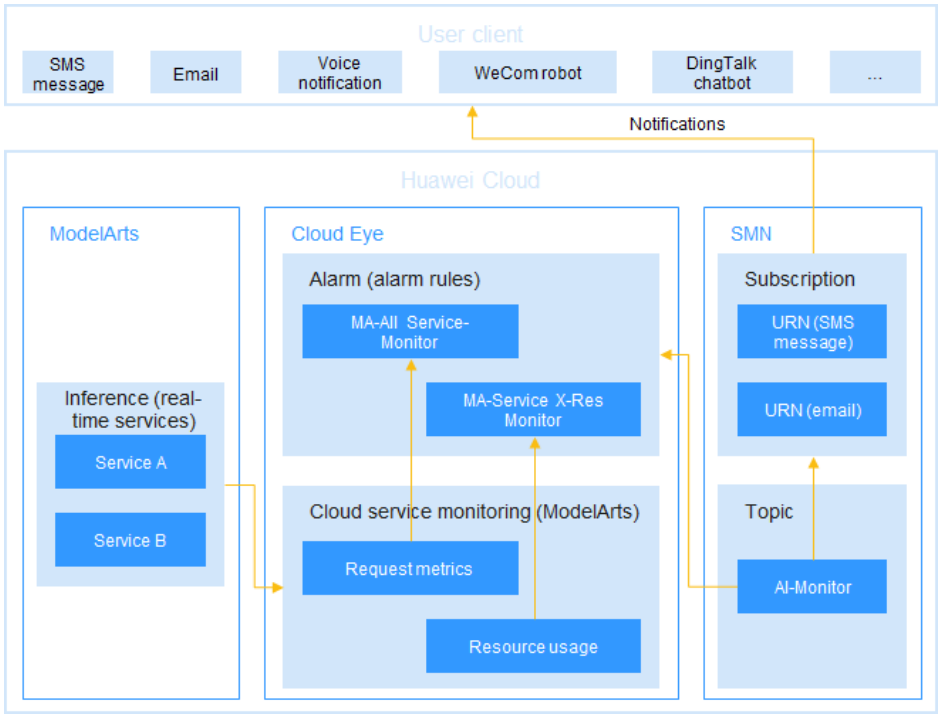
- During algorithm development, store service data in Object Storage Service (OBS), and then label and manage the data using ModelArts data management. After the data is trained, obtain an AI model and create model images using a development environment.
- During service O&M, use an image to create a model and deploy the model as a real-time service. You can obtain the monitoring data of the ModelArts real-time service on the Cloud Eye management console. Configure alarm rules so that you can be notified of alarms in real time.
- During service running, access real-time service requests into the service system and then configure service logic and monitoring.

Figure 11-5 End-to-end O&M process for inference services



During the entire O&M process, service request failures and high resource usage are monitored. When the resource usage threshold is reached, the system will send an alarm notification to you.

Figure 11-6 Alarming process



Advantages

End-to-end service O&M enables you to easily check service running at both peak and off-peak hours and detect the health status of real-time services in real time.

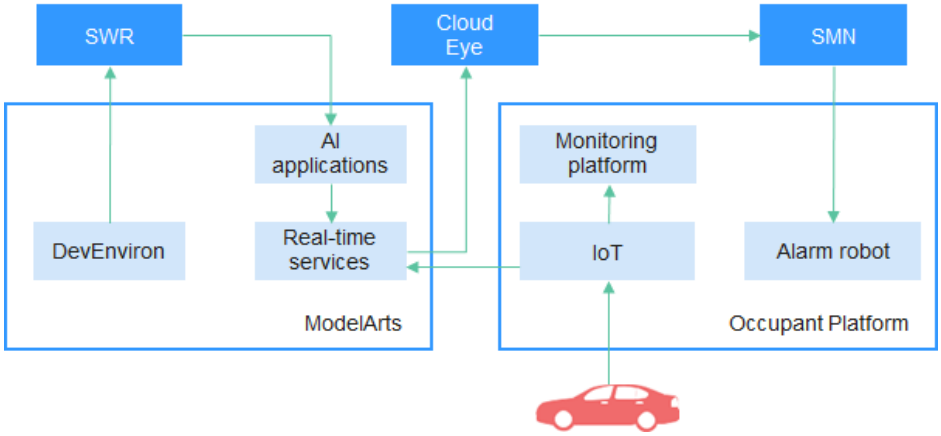
Constraints

End-to-end service O&M applies only to real-time services because Cloud Eye does not monitor batch or edge inference services.

Procedure

This section uses an occupant safety algorithm in travel as an example to describe how to use ModelArts for process-based service deployment and update, as well as automatic service O&M and monitoring.

Figure 11-7 Occupant safety algorithm implementation



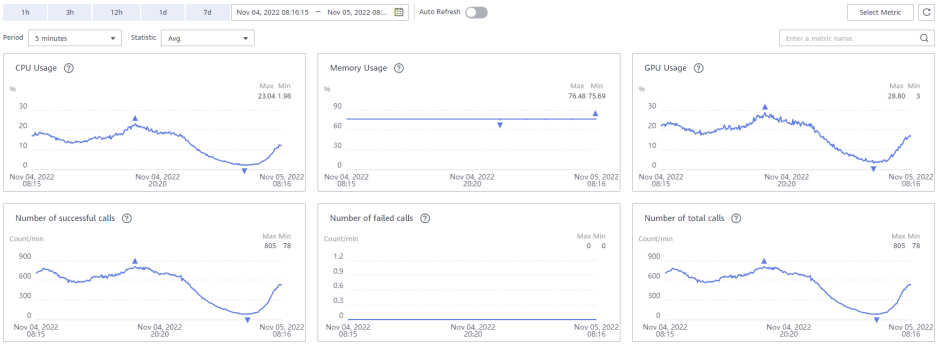
Step 1 Use a custom image to build a model that can be used on the ModelArts Standard inference platform based on the locally developed model. For details, see **Creating a Custom Image on ECS**.

Step 2 On the ModelArts management console, deploy the created model as a real-time service.

Step 3 Log in to the Cloud Eye management console, configure ModelArts alarm rules and enable notifications with a topic subscribed to. For details, see **Setting Alarm Rules**.

After the configuration, choose **Cloud Service Monitoring** > **ModelArts** in the navigation pane on the left to view the requests and resource usage of the real-time service.

Figure 11-8 Viewing service monitoring metrics



When an alarm is triggered based on the monitored data, the object who has subscribed to the target topic will receive a message notification.

----End

11.3 Creating a Model Using a Custom Engine

When using a custom engine to create a model, you can select your image stored in SWR as the engine and specify a file directory in OBS as the model package. In this way, bring-your-own images can be used to meet your dedicated requirements.

Specifications for Using a Custom Engine to Create a Model

To use a custom engine to create a model, ensure the SWR image, OBS model package, and file size comply with the following requirements:

- SWR image specifications
  - A common user named **ma-user** in group **ma-group** must be built in the SWR image. Additionally, the UID and GID of the user must be 1000 and 100, respectively. The following is the dockerfile command for the built-in user:
    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```
  - Specify a command for starting the image. In the dockerfile, specify **cmd**. The following shows an example:
    ```
    CMD sh /home/mind/run.sh
    ```
    Customize the startup entry file **run.sh**. The following is an example.
    ```
    #!/bin/bash

    # User-defined script content
    ...

    # run.sh calls app.py to start the server. For details about app.py, see "HTTPS Example".
    python app.py
    ```
    NOTE

    You can also customize the boot command for starting an image. Enter the customized command during model creation.
  - The service can use the HTTPS/HTTP protocol and listening container port. The port and protocol can be set based on the site requirements. The default request protocol and port number provided by ModelArts are HTTPS and 8080, respectively. For details, see the **HTTPS example**.
  - (Optional) The health check URL must be **/health**.
- OBS model package specifications

  The name of the model package must be **model**. For details about model package specifications, see **Model Package Structure**.
- File size specifications

  When a public resource pool is used, the total size of the downloaded SWR image (not the compressed image displayed on the SWR page) and the OBS model package cannot exceed 30 GB.

HTTPS Example

Use Flask to start HTTPS. The following is an example of the web server code:


```
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')
```

Debugging on a Local Computer

Perform the following operations on a local computer with Docker installed to check whether a custom engine complies with specifications:

1. Download the custom image, for example, **custom_engine:v1** to the local computer.
2. Copy the model package folder **model** to the local host.
3. Run the following command in the same directory as the model package folder to start the service:
   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   NOTE

   This command is used for simulation only because the directory mounted to **-v** is assigned the root permission. In the cloud environment, after the model file is downloaded from OBS to **/home/mind/model**, the file owner will be changed to **ma-user**.

4. Start another terminal on the local computer and run the following command to obtain the expected inference result:
   ```
   curl https://127.0.0.1:8080/${Request path to the inference service}
   ```

Deployment Example

The following section describes how to use a custom engine to create a model.

1. Create a model and view model details.

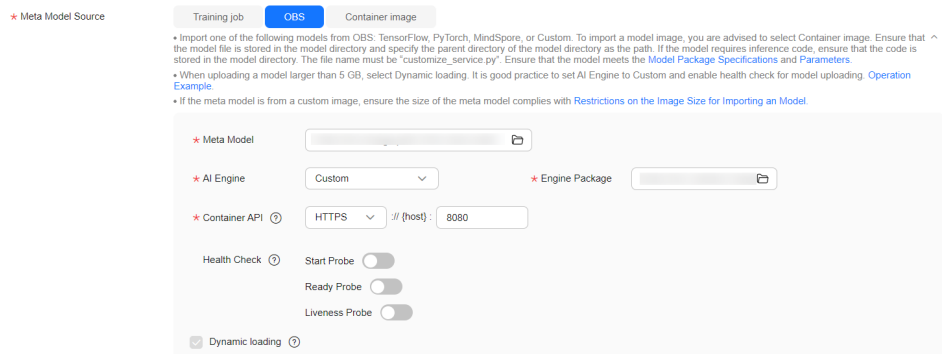
Log in to the ModelArts console. In the navigation pane on the left, choose **Model Management**. Then, click **Create Model** and configure the parameters as follows:

- **Meta Model Source**: Choose **OBS**.
- **Meta Model**: Select a model package selected from OBS
- **AI Engine**: Choose **Custom**.
- **Engine Package**: Select an SWR image.
- **Container API**: Set the port and protocol based on the actual image usage.

  Retain the default settings for other parameters.

Click **Create now**. Wait until the model status changes to **Normal**.

Figure 11-9 Creating a model



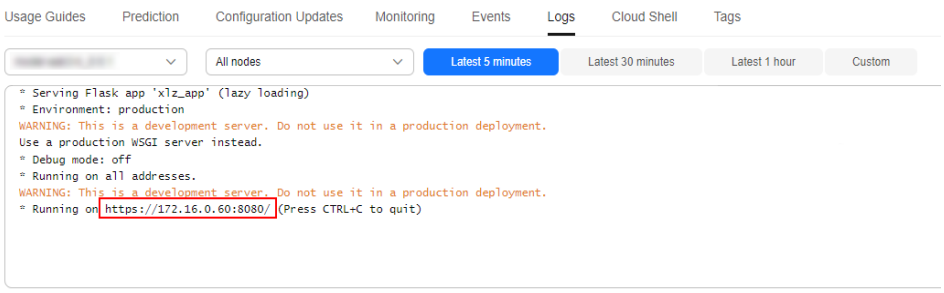
Click the model name to access its details page.

2. Deploy the AI application as a service and view service details.

   On the model details page, choose **Deploy** > **Real-Time Services** in the upper right corner. On the **Deploy** page, select a proper instance flavor (for example, **CPU: 2 vCPUs 8 GB**), retain default settings for other parameters, and click **Next**. When the service status changes to **Running**, the service has been deployed.

   Click the service name. On the page that is displayed, view the service details. Click the **Logs** tab to view the service logs.

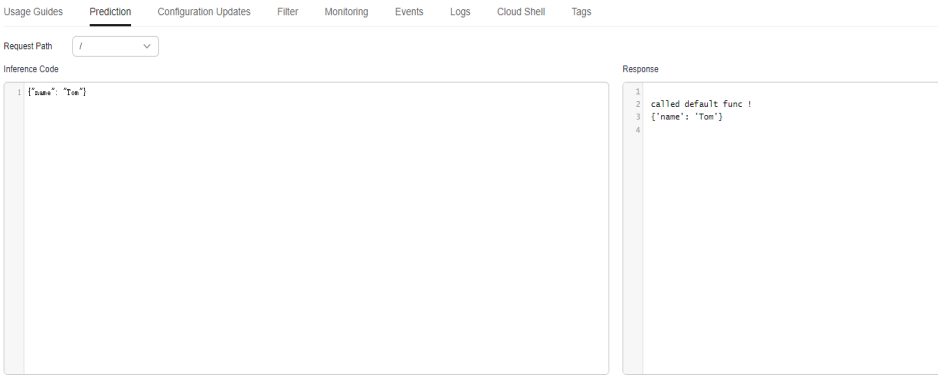
   Figure 11-10 Logs

3. Use the service for prediction.

   On the service details page, click the **Prediction** tab to use the service for prediction.

Figure 11-11 Prediction



11.4 Using a Large Model to Create a Model on ModelArts Standard and Deploy It as a Real-Time Service

Context

Currently, a large model can have hundreds of billions or even trillions of parameters, and its size becomes larger and larger. A large model with hundreds of billions of parameters exceeds 200 GB, and poses new requirements for version management and production deployment of the platform. For example, importing models requires dynamic adjustment of the tenant storage quota. Slow model loading and startup require a flexible timeout configuration in the deployment. The service recovery time needs to be shortened in the event that the model needs to be reloaded upon a restart caused by a load exception.

To address the preceding requirements, the ModelArts inference platform provides a solution to model management and service deployment in large model application scenarios.

Constraints

- You need to apply for the size quota of a model and add the whitelist cached using the local storage of the node.
- You need to use the custom engine **Custom** to configure dynamic loading.
- A dedicated resource pool is required to deploy the service.
- The disk space of the dedicated resource pool must be greater than 1 TB.

Procedure

1. **Applying for Increasing the Size Quota of a Model and Using the Local Storage of the Node to Cache the Whitelist**
2. **Uploading Model Data and Verifying the Consistency of Uploaded Objects**
3. **Creating a Dedicated Resource Pool**
4. **Creating a Model**

5. **Deploying a Real-Time Service**

Applying for Increasing the Size Quota of a Model and Using the Local Storage of the Node to Cache the Whitelist

During service deployment, the dynamically loaded model package is stored in the temporary disk space by default. When the service is stopped, the loaded files are deleted, and they need to be reloaded when the service is restarted. To avoid repeated loading, the platform allows the model package to be loaded from the local storage space of the node in the resource pool and keeps the loaded files valid even when the service is stopped or restarted (using the hash value to ensure data consistency).

To use a large model, you need to use a custom engine and enable dynamic loading when importing the model. In this regard, you need to perform the following operations:

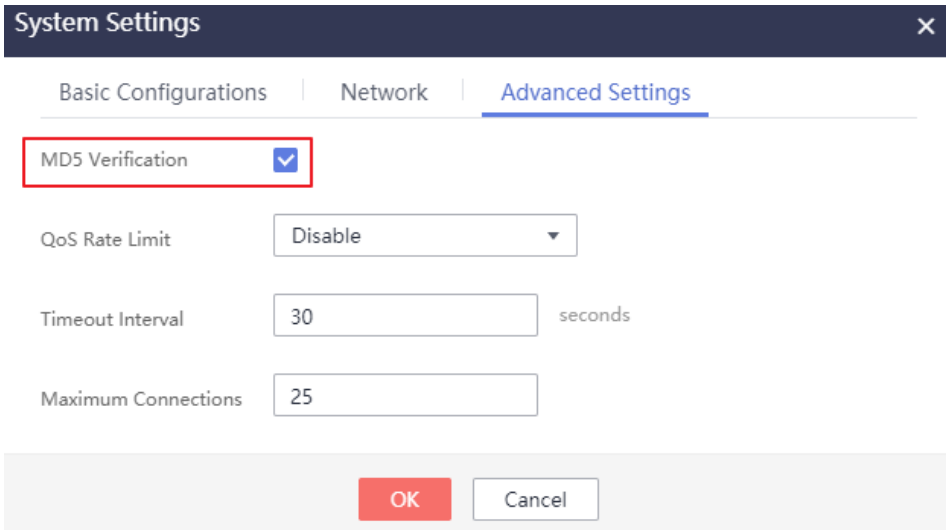
- If the model size exceeds the default quota, submit a service ticket to increase the size quota of a single model. The default size quota of a model is 20 GB.
- Submit a service ticket to add the whitelist cached using the local storage of the node.

Uploading Model Data and Verifying the Consistency of Uploaded Objects

To ensure data integrity during dynamic loading, you need to verify the consistency of uploaded objects when uploading model data to OBS. obsutil, OBS Browser+, and OBS SDKs support verification of data consistency during upload. You can select a method that meets your requirements. For details, see **Verifying Data Consistency During Upload**.

For example, if you upload data via OBS Browser+, enable MD5 verification, as shown in **Figure 11-12**. When dynamic loading is enabled and the local persistent storage of the node is used, OBS Browser+ checks data consistency during data upload.

Figure 11-12 Configuring MD5 verification for OBS Browser+



Creating a Dedicated Resource Pool

To use the local persistent storage, you need to create a dedicated resource pool whose disk space is greater than 1 TB. You can view the disk information on the **Specifications** tab of the **Basic Information** page of the dedicated resource pool. If a service fails to be deployed and the system displays a message indicating that the disk space is insufficient, see **What Do I Do If Resources Are Insufficient When a Real-Time Service Is Deployed, Started, Upgraded, or Modified**.

Figure 11-13 Viewing the disk information of the dedicated resource pool

Creating a Model

If you use a large model to create a model and import the model from OBS, complete the following configurations:

1. Use a custom engine and enable dynamic loading.

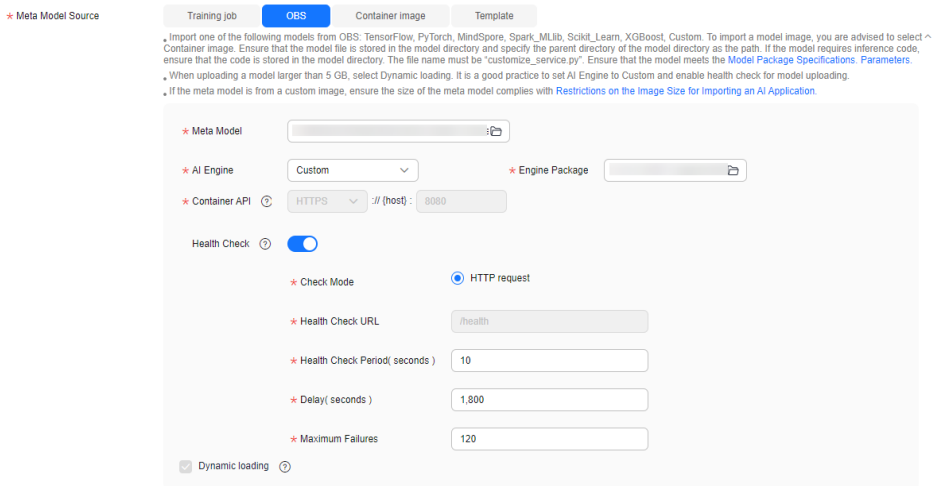
   To use a large model, you need to use a custom engine and enable dynamic loading when importing the model. You can create a custom engine to meet special requirements for image dependency packages and inference frameworks in large model scenarios. For details about how to create a custom engine, see **Creating a Model Using a Custom Engine**.

   When you use a custom engine, dynamic loading is enabled by default. The model package is separated from the image, and the model is dynamically loaded to the service load during service deployment.

2. Configure health check.

   Health check is mandatory for the models imported using a large model to identify unavailable services that are displayed as started.

   Figure 11-14 Using a custom engine, enabling dynamic loading, and configuring health check



Deploying a Real-Time Service

When deploying the service, complete the following configurations:

1. Customize the deployment timeout interval.

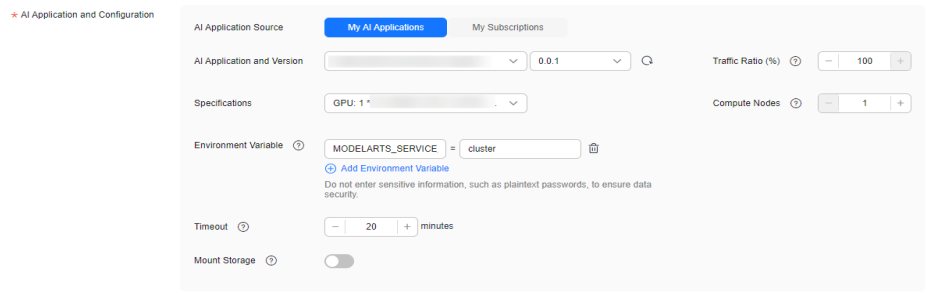
   Generally, the time for loading and starting a large model is longer than that for a common model. Set **Timeout** to a proper value. Otherwise, the timeout may elapse prior to the completion of the model startup, and the deployment may fail.

2. Add an environment variable.

   During service deployment, add the following environment variable to set the service traffic load balancing policy to cluster affinity, preventing unready service instances from affecting the prediction success rate:

   ```
   MODELARTS_SERVICE_TRAFFIC_POLICY: cluster
   ```

   Figure 11-15 Customizing the deployment timeout interval and adding an environment variable

You are advised to deploy multiple instances to improve service reliability.

11.5 Migrating a Third-Party Inference Framework to a Custom Inference Engine

Context

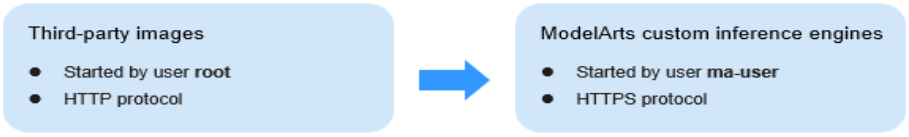
ModelArts allows the deployment of third-party inference frameworks. This section describes how to migrate TF Serving and Triton to a custom inference engine.

- TensorFlow Serving (TF Serving) is a flexible, high-performance model deployment system for machine learning. It provides model version management and service rollback capabilities. By configuring parameters such as the model path, model port, and model name, native TF Serving images can quickly start providing services which can be accessed through gRPC and HTTP RESTful APIs.
- Triton is a high-performance inference service framework. It supports multiple service protocols, including HTTP and gRPC. Additionally, Triton is compatible with various inference engine backends such as TensorFlow, TensorRT, PyTorch, and ONNX Runtime. Notably, it enables multi-model concurrency and dynamic batching, effectively optimizing GPU utilization and enhancing inference service performance.

The migration of a third-party framework to a ModelArts inference framework requires reconstruction of the native third-party framework image. After that, ModelArts model version management and dynamic model loading can be used. This section shows how to complete such a reconstruction. After an image of the custom engine is created, you can use it to create a model and deploy and manage services using the model.

The following figure shows the reconstruction items.

Figure 11-16 Reconstruction items



The reconstruction process may differ for images from various frameworks. For details, see the migration procedure specific to the target framework.

- **Migrating TF Serving**
- **Migrating Triton**

Migrating TF Serving

Step 1 Add user **ma-user**.

The image is built based on the native **tensorflow/serving:2.8.0** image. The user group **100** exists in the image by default. Run the following command in the Dockerfile to add user **ma-user**:

```
RUN useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

Step 2 Set up an Nginx proxy to support HTTPS.

After the protocol is converted to HTTPS, the exposed port changes from 8501 of TF Serving to 8080.

1. Run the following commands in the Dockerfile to install and configure Nginx:

```
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. Create the Nginx directory.

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. Write the nginx.conf file.


```
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. Write the modelarts-model-server.conf configuration file.

```
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. Create a startup script.

   NOTE

   Before executing the TF Serving startup script, you must create an SSL certificate.

   The sample code of the startup script **run.sh** is as follows:

```
#!/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
```

Step 3 Modify the default model path to support ModelArts model dynamic loading.

Run the following commands in the Dockerfile to change the default model path:

```
ENV MODEL_BASE_PATH /home/mind
ENV MODEL_NAME model
```

----End

Dockerfile example:

```
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
```

Migrating Triton

This section uses the **nvcr.io/nvidia/tritonserver:23.03-py3** image provided by NVIDIA for adaptation and the open-source foundation model LLaMA 7B for inference.

Step 1 Add user **ma-user**.

The **triton-server** user, whose ID is 1000, exists in the Triton image by default. Change the **triton-server** user ID and add the **ma-user** user by running this command in the Dockerfile.

```
RUN usermod -u 1001 triton-server && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
```

Step 2 Set up an Nginx proxy to support HTTPS.

1. Run the following commands in the Dockerfile to install and configure Nginx:

```
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. Create the Nginx directory as follows:

```
nginx
├──nginx.conf
└──conf.d
    ├── modelarts-model-server.conf
```

3. Write the nginx.conf file.

```
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. Write the modelarts-model-server.conf configuration file.

```
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. Create a startup script **run.sh**.

   NOTE

   Before executing the Triton startup script, you must create an SSL certificate.

```
#!/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
```

Step 3 Set up **tensorrtllm_backend**.

1. Obtain the source code of **tensorrtllm_backend**; install dependencies (TensorRT, CMake, and PyTorch); compile and install.

```
# 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. Create the startup script **triton_serving.sh** of Triton serving. The following is an example for the LLaMA model:

```
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
```

Description of some parameters:

- **MODEL_NAME**: name of the OBS folder where the model weight file in Hugging Face format is stored.
- **OUTPUT_DIR**: path to the model file converted by TensorRT-LLM in the container.

The complete Dockerfile is as follows:

```
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 && \
```



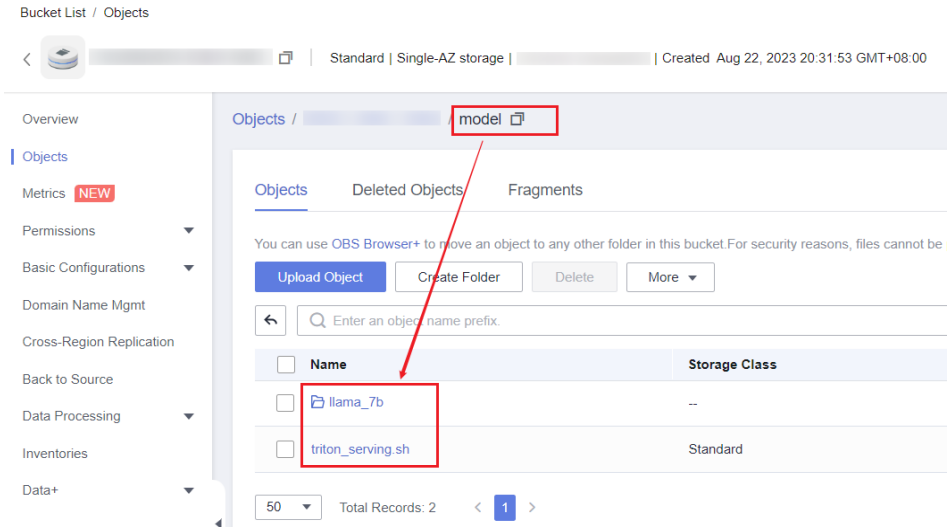
```
    cp ./build/libtriton_tensorrtllm.so /opt/tritonserver/backends/tensorrtllm/ && \
    chown -R ma-user:100 /opt/tritonserver

ADD nginx /etc/nginx
ADD run.sh /home/mind/
CMD /bin/bash /home/mind/run.sh
```

After the image is created, register the image with Huawei Cloud SWR for deploying inference services on ModelArts.

Step 4 Use the adapted image to deploy a real-time inference service on ModelArts.

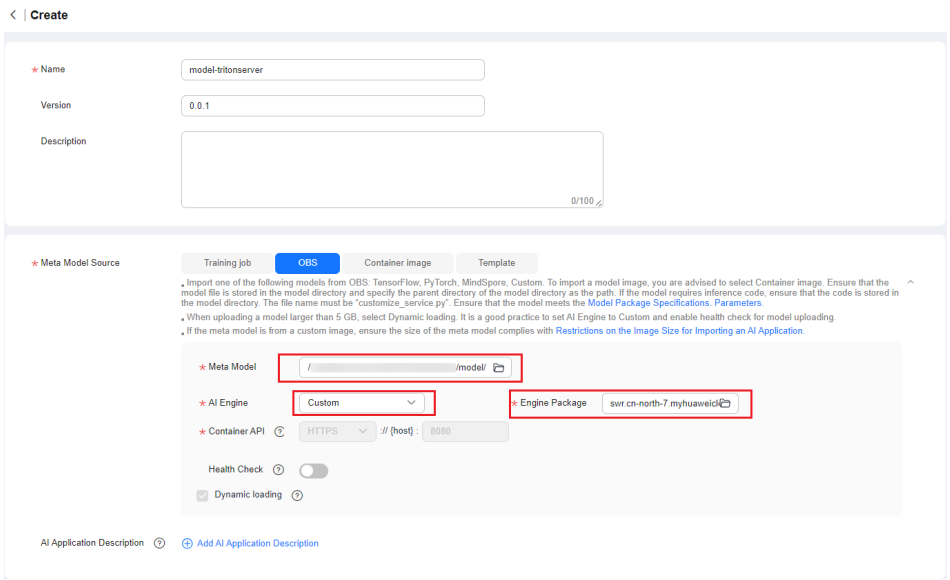
1. Create a **model** directory in OBS and upload the **triton_serving.sh** file and **llama_7b** folder to the **model** directory.

Figure 11-17 Uploading files to the **model** directory



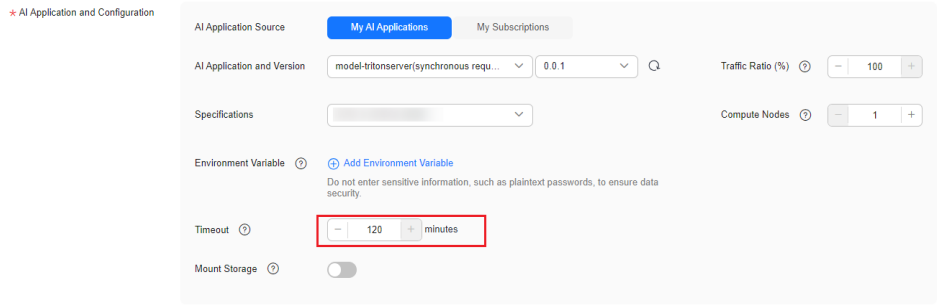
2. Create a model. Set **Meta Model Source** to **OBS** and select the meta model from the **model** directory. Set **AI Engine** to **Custom**. Set **Engine Package** to the image created in **Step 3**.

Figure 11-18 Creating a model



3. Deploy the created model as a real-time service. Generally, the time for loading and starting a large model is longer than that for a common model. Set **Timeout** to a proper value. Otherwise, the timeout may elapse prior to the completion of the model startup, and the deployment may fail.

Figure 11-19 Deploying a real-time service



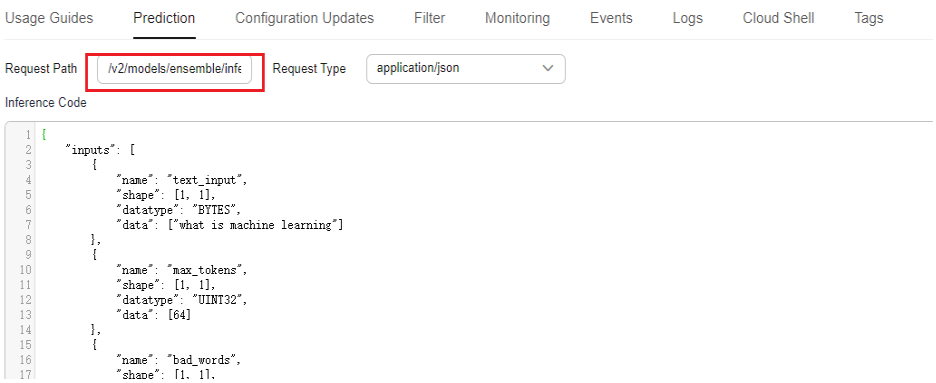
4. Call the real-time service for foundation model inference. Set the request path to **/v2/models/ensemble/infer**. The following is an example call:

```
{
  "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"
    }
  ]
}
```


NOTE

- In "inputs", the element with the "name" "text_input" represents the input, and its "data" field specifies a specific input statement. In this example, the input statement is "what is machine learning".
- The element with the "name" "max_tokens" indicates the maximum number of output tokens. In this case, the value is **64**.

Figure 11-20 Calling a real-time service



----End

11.6 Enabling High-Speed Access to an Inference Service Through VPC Peering

Context

When accessing a real-time service, you may require:

- High throughput and low latency
- TCP or RPC requests

To meet these requirements, ModelArts enables high-speed access through VPC peering.

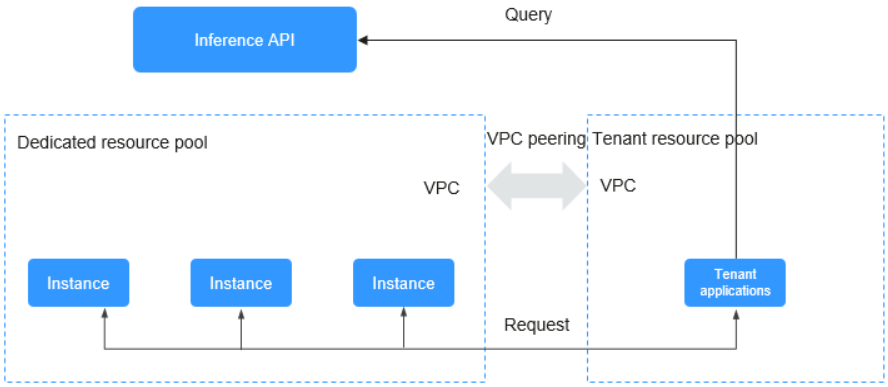
In high-speed access through VPC peering, your service requests are directly sent to instances through VPC peering but not through the inference platform. This accelerates service access. This method works best when you need high bandwidth with minimal delays, like handling real-time data or video streams.

WARNING

When using a high-speed VPC access channel for a real-time service, requests do not reach the inference platform, making these features inaccessible:

- Authentication
- Traffic distribution by configuration
- Load balancing
- Alarm, monitoring, and statistics

Figure 11-21 High-speed access through VPC peering



Constraints

When you call an API to access a real-time service, the size of the prediction request body and the prediction time are subject to the following limitations:

- The size of a request body cannot exceed 12 MB. Otherwise, the request will fail.
- Due to the limitation of API Gateway, the prediction duration of each request does not exceed 40 seconds.
- Only the services deployed in a dedicated resource pool support high-speed access through VPC peering.
- High-speed access through VPC peering is available only for real-time services.
- Due to traffic control, there is a limit on how often you can get the IP address and port number of a real-time service. The number of calls of each tenant account cannot exceed 2000 per minute, and that of each IAM user account cannot exceed 20 per minute.
- High-speed access through VPC peering is available only for the services deployed using the AI applications imported from custom images.

Preparations

Deploy a real-time service in a dedicated resource pool and ensure the service is running.

Procedure

To enable high-speed access to a real-time service through VPC peering, perform the following operations:

1. **Interconnect the dedicated resource pool to the VPC.**
2. **Create an ECS in the VPC.**
3. **Obtain the IP address and port number of the real-time service.**
4. **Access the service through the IP address and port number.**

Step 1 Interconnect the dedicated resource pool to the VPC.

Log in to the ModelArts console, choose **Resource Management** > **Standard Cluster**, find the dedicated resource pool where the service is deployed, and click its name/ID to go to the resource pool details page. Obtain the network configuration. Return to the homepage, choose **Network** in the navigation pane, find the network associated with the dedicated resource pool, and interconnect your VPC. After the VPC is accessed, the VPC will be displayed on the network list and resource pool details pages. Click the VPC to go to the details page.

Figure 11-22 Obtaining the network configuration

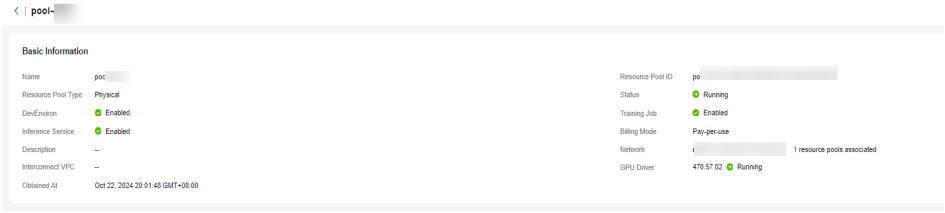


Figure 11-23 Interconnecting the VPC

Step 2 Create an ECS in the VPC.

Log in to the ECS management console and click **Buy ECS** in the upper right corner. On the **Buy ECS** page, configure basic settings and click **Next: Configure Network**. On the **Configure Network** page, select the VPC connected in **Step 1**, configure other parameters, confirm the settings, and click **Submit**. When the ECS status changes to **Running**, the ECS has been created. Click its name/ID to go to the server details page and view the VPC configuration.

Figure 11-24 Selecting a VPC when purchasing an ECS

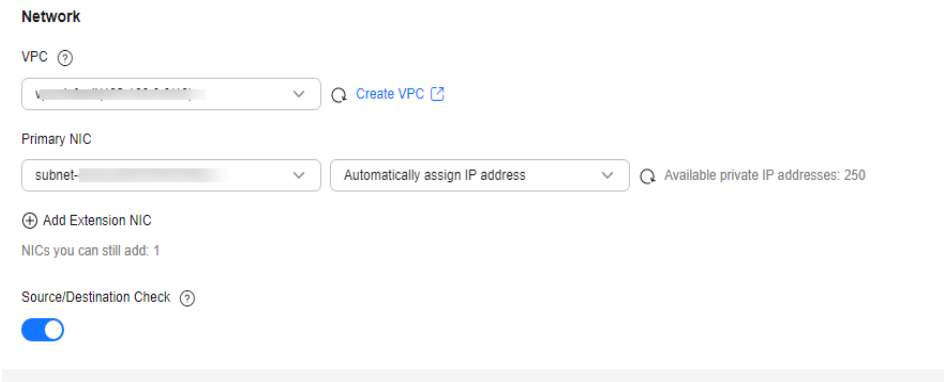
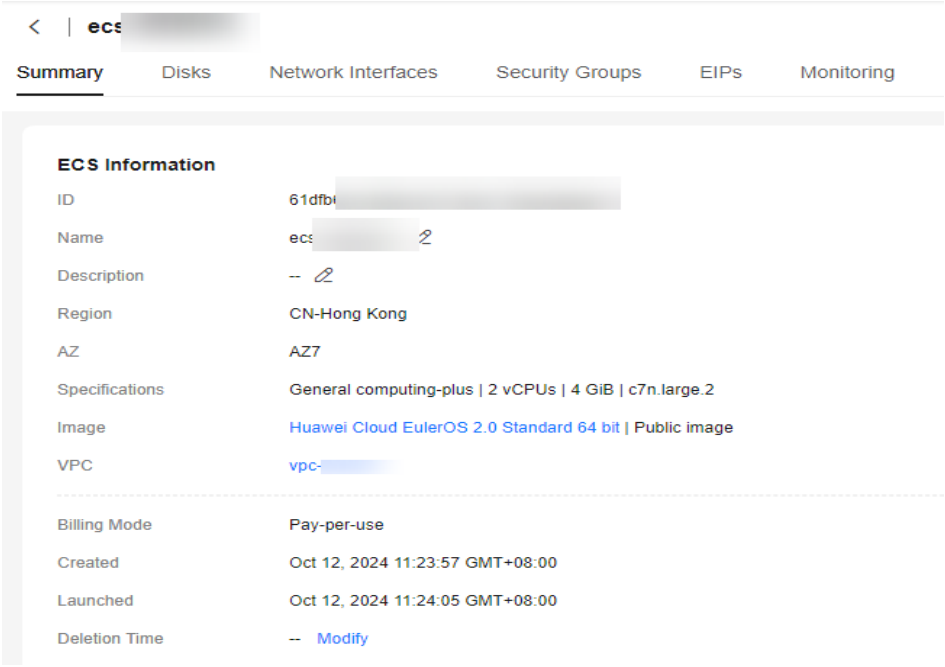


Figure 11-25 Viewing VPC information



Step 3 Obtain the IP address and port number of the real-time service.

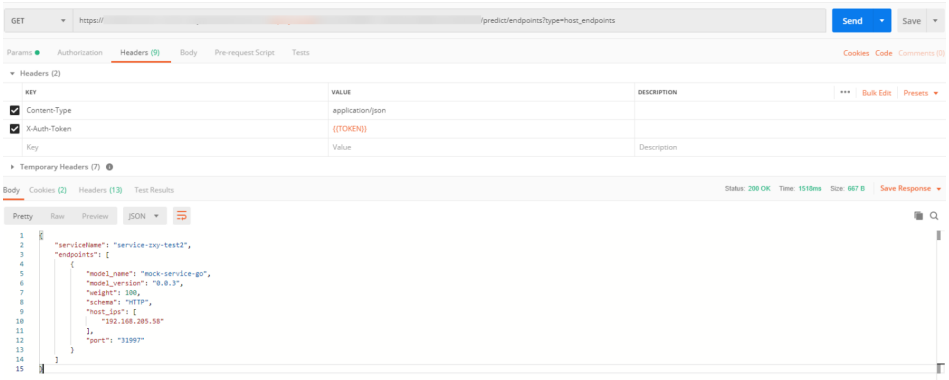
GUI software, for example, Postman can be used to obtain the IP address and port number. Alternatively, log in to the ECS, create a Python environment, and execute code to obtain the service IP address and port number.

API:

```
GET /v1/{project_id}/services/{service_id}/predict/endpoints?type=host_endpoints
```

- Method 1: Obtain the IP address and port number using GUI software.

  Figure 11-26 Example response

- Method 2: Obtain the IP address and port number using Python.

  The following parameters in the Python code below need to be modified:

  – **project_id**: your project ID. To obtain it, see **Obtaining a Project ID and Name**.

  – **service_id**: service ID, which can be viewed on the service details page.

– **REGION_ENDPOINT**: service endpoint. To obtain it, see **Endpoint**.

```
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)
```

Step 4 Access the service through the IP address and port number.

Log in to the ECS and access the real-time service either by running Linux commands or by creating a Python environment and executing Python code. Obtain the values of **schema**, **ip**, and **port** from **Step 3**.

- Run the following command to access the real-time service:

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

Figure 11-27 Accessing a real-time service

```
[root@ecs-zxy ~]# curl --location --request POST 'http://192.168.205.58:31997' \
> --header 'Content-Type: application/json' \
> --data-raw '{"a":"a"}'
call Post()[root@ecs-zxy ~]#
```

- Create a Python environment and execute Python code to access the real-time service.

```
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)
```

NOTE

High-speed access does not support load balancing. You need to customize load balancing policies when you deploy multiple instances.

----End

11.7 Full-Process Development of WebSocket Real-Time Services

Context

WebSocket is a network transmission protocol that supports full-duplex communication over a single TCP connection. It is located at the application layer in an OSI model. The WebSocket communication protocol was established by IETF in 2011 as standard RFC 6455 and supplemented by RFC 7936. The WebSocket API in the Web IDL is standardized by W3C.

WebSocket simplifies data exchange between the client and the server and allows the server to proactively push data to the client. In the WebSocket API, if the initial handshake between the client and the server is successful, a persistent connection will be established between them and data can be transferred bidirectionally.

Prerequisites

- You are experienced in developing Java and familiar with JAR packaging.
- You have basic knowledge and calling methods of WebSocket.
- You are familiar with the method of creating an image using Docker.

Constraints

- WebSocket supports only the deployment of real-time services.
- It supports only real-time services deployed using models imported from custom images.

Preparations

Before using WebSocket in ModelArts for inference, bring your own custom image. The custom image must be able to provide complete WebSocket services in a standalone environment, for example, completing WebSocket handshakes and exchanging data between the client to the server. The model inference is implemented in the custom image, including downloading the model, loading the model, performing preprocessing, completing inference, and assembling the response body.

Procedure

To develop a WebSocket real-time service, perform the following operations:

- **Uploading the Image to SWR**
- **Creating a Model Using an Image**
- **Deploying the Model as a Real-Time Service**
- **Calling the WebSocket Real-Time Service**

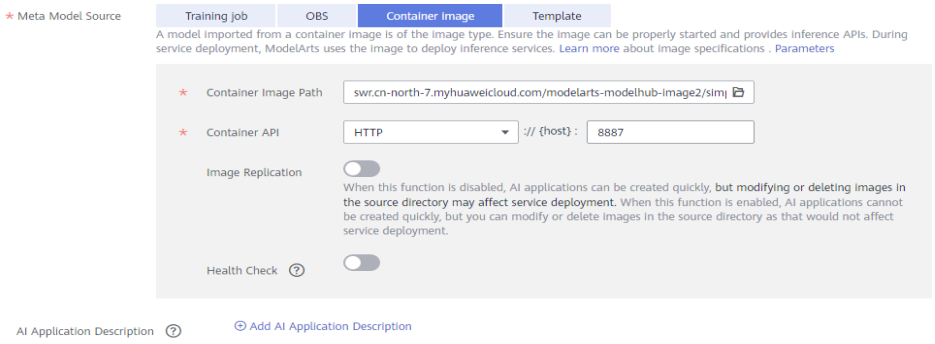
Uploading the Image to SWR

Upload the local image to SWR.

Creating a Model Using an Image

1. Log in to the ModelArts console. In the navigation pane on the left, choose **Model Management (AI Applications)**. On the displayed page, click **Create Model**.
2. Configure the AI application.
   - **Meta Model Source**: Select **Container image**.
   - **Container Image Path**: Select the path specified in **Uploading the Image to SWR**.
   - **Container API**: Configure this parameter based on site requirements.
   - **Health Check**: Retain default settings. If health check has been configured in the image, configure the health check parameters based on those configured in the image.

Figure 11-28 Model configuration parameters

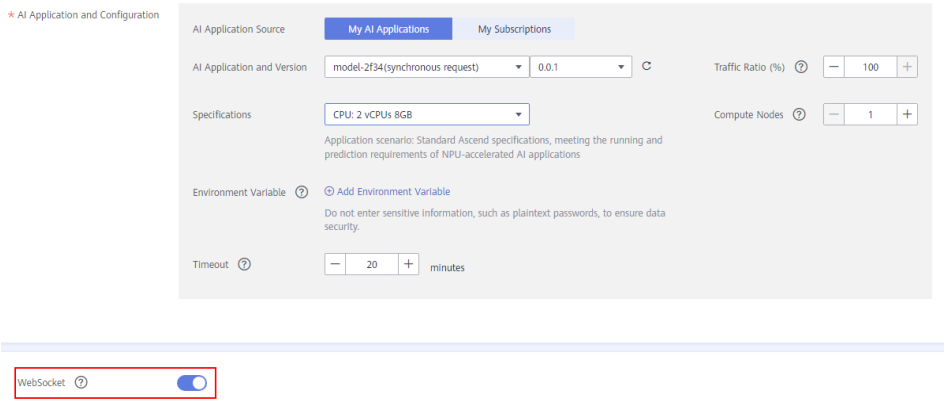


3. Click **Create now**. Wait until the model status changes to **Normal**.

Deploying the Model as a Real-Time Service

1. Log in to the ModelArts management console. In the navigation pane on the left, choose **Model Deployment** > **Real-Time Services**. On the displayed page, click **Deploy**.
2. Configure the service.
   - **Model and Version**: Select the model and version created in **Creating a Model Using an Image**.
   - **WebSocket**: Enable this function.

   Figure 11-29 WebSocket



3. Click **Next**, confirm the configuration, and click **Submit**. In the real-time service list you will be redirected to, check the service status. When it changes to **Running**, the real-time service has been deployed.

Calling a WebSocket Real-Time Service

WebSocket itself does not require additional authentication. ModelArts WebSocket is WebSocket Secure-compliant, regardless of whether WebSocket or WebSocket Secure is enabled in the custom image. WebSocket Secure supports only one-way authentication, from the client to the server.

You can use one of the following authentication methods provided by ModelArts:

- **Token-based Authentication**

- **AK/SK**
- **App Authentication**

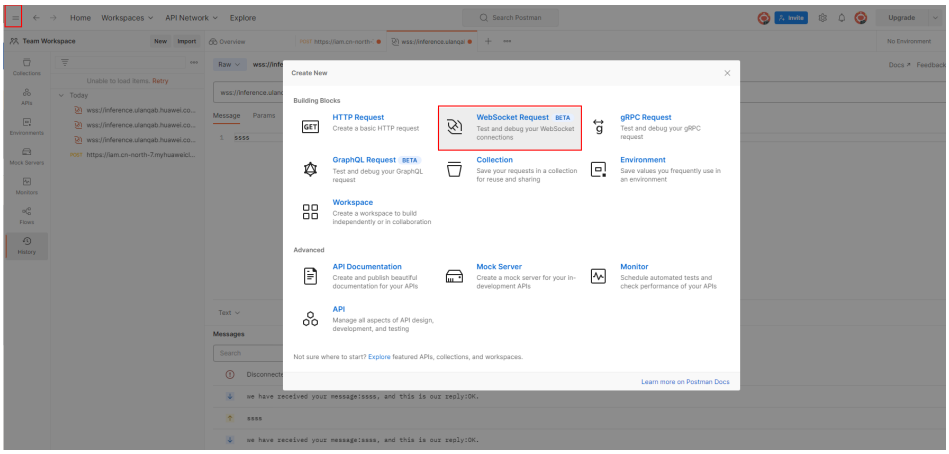
The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call WebSocket.

1. **Establish a WebSocket connection.**
2. **Exchange data between the WebSocket client and the server.**

Step 1 Establish a WebSocket connection.

1. Open Postman of a version later than 8.5, for example, 10.12.0. Click  in the upper left corner and choose **File** > **New**. In the displayed dialog box, select **WebSocket Request** (beta version currently).

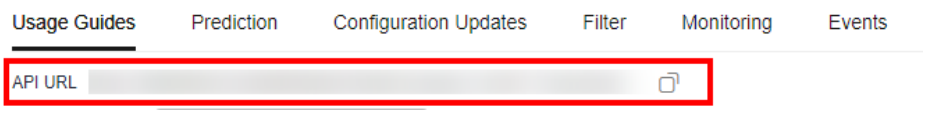
   Figure 11-30 WebSocket Request

2. Configure parameters for the WebSocket connection.

   Select **Raw** in the upper left corner. Do not select **Socket.IO** (a type of WebSocket implementation, which requires that both the client and the server run on **Socket.IO**). In the address box, enter the **API Address** obtained on the **Usage Guides** tab on the service details page. If there is a finer-grained URL in the custom image, add the URL to the end of the address. If **queryString** is available, add this parameter in the **params** column. Add authentication information into the header. The header varies depending on the authentication mode, which is the same as that in the HTTPS-compliant inference service. Click **Connect** in the upper right corner to establish a WebSocket connection.

   Figure 11-31 Obtaining the API address

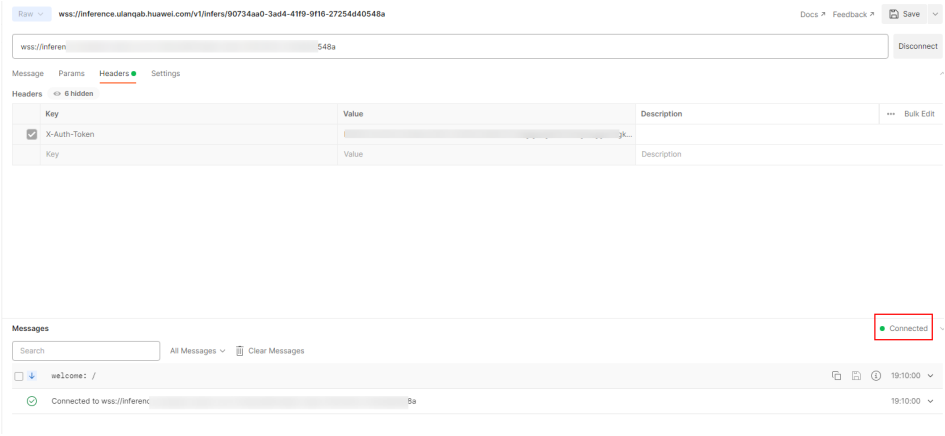
   

NOTE

- If the information is correct, **CONNECTED** will be displayed in the lower right corner.
- If establishing the connection failed and the status code is 401, check the authentication.
- If a keyword such as **WRONG_VERSION_NUMBER** is displayed, check whether the port configured in the custom image is the same as that configured in WebSocket or WebSocket Secure.

The following shows an established WebSocket connection.

Figure 11-32 Connection established



NOTICE

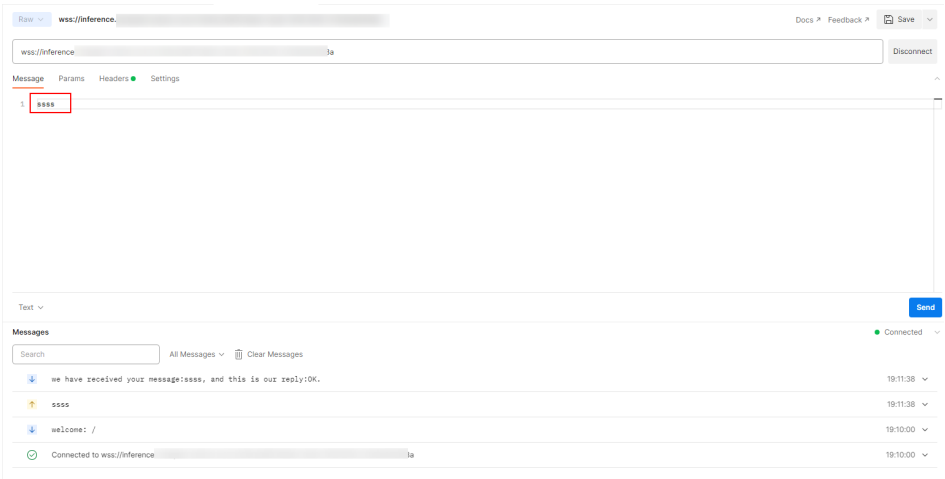
Preferentially check the WebSocket service provided by the custom image. The type of implementing WebSocket varies depending on the tool you used. Possible issues are as follows: A WebSocket connection can be established but cannot be maintained, or the connection is interrupted after one request and needs to be reconnected. ModelArts only ensures that it will not affect the WebSocket status in a custom image (the API address and authentication mode may be changed on ModelArts).

Step 2 Exchange data between the WebSocket client and the server.

After the connection is established, WebSocket uses TCP for full-duplex communication. The WebSocket client sends data to the server. The implementation types vary depending on the client, and the lib package may also be different for the same language. Different implementation types are not considered here.

The format of the data sent by the client is not limited by the protocol. Postman supports text, JSON, XML, HTML, and Binary data. Take text as an example. Enter the text data in the text box and click **Send** on the right to send the request to the server. If the text is oversized, Postman may be suspended.

Figure 11-33 Sending data



----End

11.8 Creating a Custom Image and Using It to Create a Model

If you want to use an AI engine that is not supported by ModelArts, create a custom image for the engine, import the image to ModelArts, and use the image to create models. This section describes how to use a custom image to create a model and deploy it as a real-time service.

The procedure is as follows:

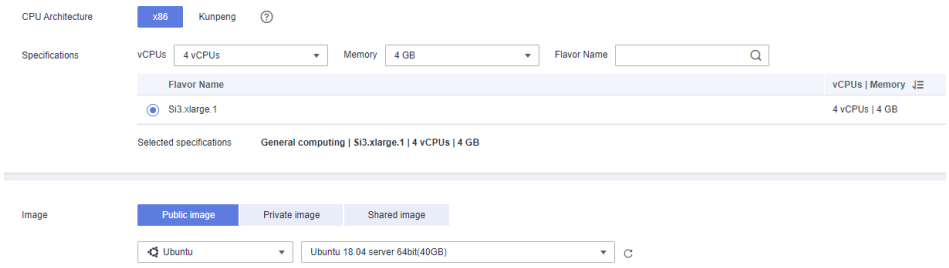
1. **Building an Image Locally**: Create a custom image package locally. For details, see **Specifications for Custom Images**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Creating a Model Using a Custom Image**: Import the image to ModelArts model management.
4. **Deploying the Model as a Real-Time Service**: Deploy the imported model.

Building an Image Locally

This section uses a Linux x86_x64 host as an example. You can purchase an ECS of the same specifications or use an existing local host to create a custom image.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. When creating the ECS, select an Ubuntu 18.04 public image.

Figure 11-34 Creating an ECS using an x86 public image



1. After logging in to the host, install Docker. For details, see **Docker official documents**. Alternatively, run the following commands to install Docker:
   ```
   curl -fsSL get.docker.com -o get-docker.sh
   sh get-docker.sh
   ```
2. Obtain the base image. Ubuntu 18.04 is used in this example.
   ```
   docker pull ubuntu:18.04
   ```
3. Create the **self-define-images** folder, and edit **Dockerfile** and **test_app.py** in the folder for the custom image. In the sample code, the application code runs on the Flask framework.

   The file structure is as follows:
   ```
   self-define-images/
     --Dockerfile
     --test_app.py
   ```

   - **Dockerfile**
     ```
     From ubuntu:18.04
     # Configure the Huawei Cloud source and install Python, Python3-PIP, and 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

     # Copy the application code to the image.
     COPY test_app.py /opt/test_app.py

     # Specify the boot command of the image.
     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'
     ```



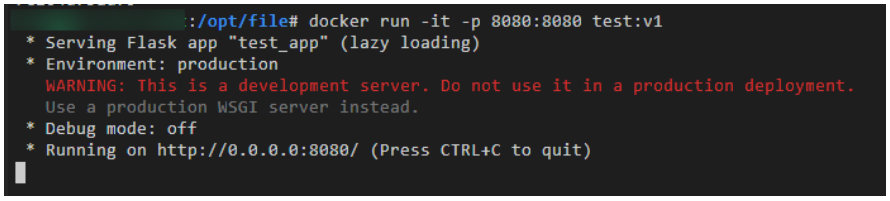
```
@app.route('/', methods=['POST'])
def default_func():
    print("---------- in default func ----------")
    data = json.loads(request.get_data(as_text=True))
    return '\n called default func !\n {} \n'.format(str(data))

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080)
```

4. Switch to the **self-define-images** folder and run the following command to create custom image **test:v1**:
   ```
   docker build -t test:v1 .
   ```
5. Run **docker images** to view the custom image you have created.

Verifying the Image Locally and Uploading It to SWR

1. Run the following command in the local environment to start the custom image:
   ```
   docker run -it -p 8080:8080 test:v1
   ```

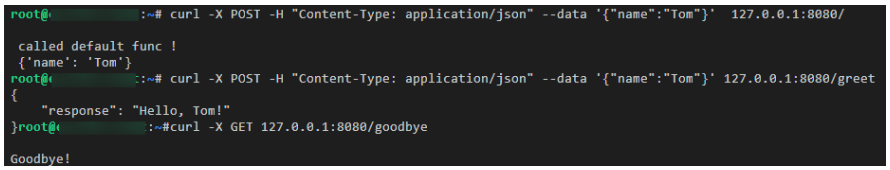
   Figure 11-35 Starting a custom image

2. Open another terminal and run the following commands to test the functions of the three APIs of the custom image:
   ```
   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
   ```

   If information similar to the following is displayed, the function verification is successful.

   Figure 11-36 Testing API functions

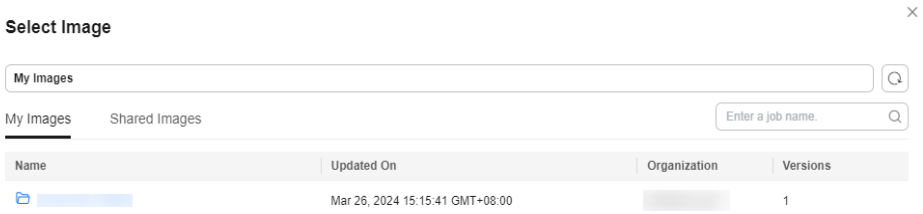
3. Upload the custom image to SWR.
4. After the custom image is uploaded, view the uploaded image on the **My Images** > **Private Images** page of the SWR console.

Creating a Model Using a Custom Image

Import a meta model. For details, see **Importing a Meta Model from a Container Image**. Key parameters are as follows:

- **Meta Model Source**: Select **Container image**.
  - **Container Image Path**: Select the created private image.

Figure 11-37 Created private image



  - **Container API**: Protocol and port number for starting a model. Ensure that the protocol and port number are the same as those provided in the custom image.
  - **Image Replication**: indicates whether to copy the model image in the container image to ModelArts. This parameter is optional.
  - **Health Check**: checks health status of a model. This parameter is optional. This parameter is configurable only when the health check API is configured in the custom image. Otherwise, creating the model will fail.
- **APIs**: APIs of a custom image. This parameter is optional. The model APIs must comply with ModelArts specifications. For details, see **Specifications for Editing a Model Configuration File**.

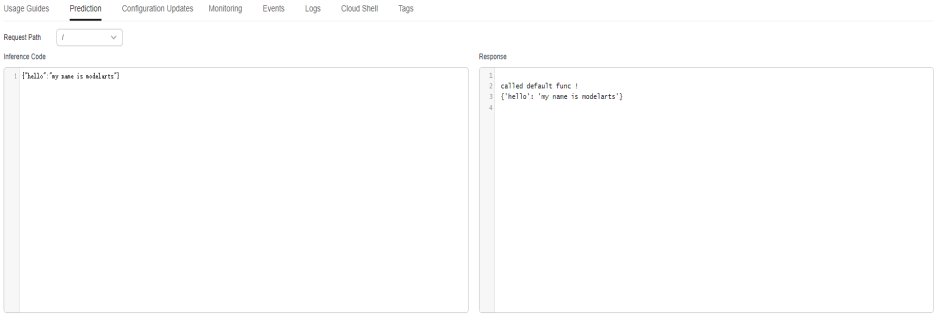
  The configuration file is as follows:

```
[{
      "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"
      }
   }
]
```

Deploying the Model as a Real-Time Service

1. Deploy the model as a real-time service. For details, see **Deploying a Model as a Real-Time Service**.
2. View the details about the real-time service.
3. Access the real-time service in the **Prediction** tab.

Figure 11-38 Accessing the real-time service



12 Best Practices of Security Configuration

Scenario

Security is a shared responsibility between Huawei Cloud and yourself. Huawei Cloud ensures the security of cloud services for a secure cloud. As a tenant, you should properly use the security capabilities provided by cloud services to protect data and securely use the cloud.

This section provides actionable guidance for enhancing the overall security of ModelArts. You can continuously evaluate the security status of your ModelArts resources and enhance their overall security by combining different security capabilities provided by ModelArts. By doing this, data stored in ModelArts can be protected from leakage and tampering both at rest and in transit.

Consider the following aspects for your security configurations:

- **Using an IP Address Whitelist for Access to Notebook**
- **Using a Dedicated Resource Pool in the Production Environment**
- **Running a Custom Image as a Non-root User**
- **Not Using Hard-coded Credentials During Development**
- **Using Independent Agencies for Different IAM Users**

Using an IP Address Whitelist for Access to Notebook

ModelArts Standard notebook instances can be directly connected in SSH mode and authenticated using key pairs. If you have higher security requirements, configure an IP address whitelist to limit access to the instance exclusively to approved endpoints. For details, see the **Whitelist** parameter in **Creating a Notebook Instance**.

Using a Dedicated Resource Pool in the Production Environment

When you use the training, inference, and development environments, you shall use the dedicated resource pool in the production environment, which provides exclusive compute resources and enhanced secure resource isolation capabilities. For details about how to use a dedicated resource pool, see **Creating a Standard Dedicated Resource Pool**.

When using ModelArts for full-process AI development, you can use two different resource pools.

Public resource pools: provide large-scale public compute clusters, which are allocated based on job parameter settings. Resources are isolated by job. You will be billed based on resource specifications, usage duration, and the number of instances used in a public resource pool, regardless of tasks (training, deployment, or development). Public resource pools are provided by ModelArts by default and do not need to be created or configured. You can directly select a public resource pool during AI development.

Dedicated resource pools: provide dedicated compute resources, which can be used for notebook instances, training jobs, and model deployment. The resources provided in a dedicated resource pool are exclusive, featuring higher resource efficiency than a public resource pool.

To use a dedicated resource pool, you need to purchase one and select it during AI development.

Running a Custom Image as a Non-root User

You can create a Dockerfile for a custom image and then push it to SWR. To enhance permission control, you shall explicitly define the default running user as a non-root user when customizing an image. This helps reduce security risks during container runtime.

During the development and runtime of AI services, complex environment dependencies need to be debugged for solidifying configurations. In the best practices of AI development in ModelArts, container images are used to solidify the runtime environment. In this way, dependencies can be managed and the runtime environment can be easily switched. ModelArts offers cloud-based container resources that speed up AI development and streamline model testing cycles.

For details about how to use custom images in ModelArts Standard, see **Application Scenarios of Custom Images**.

Not Using Hard-coded Credentials During Development

If you want to develop an algorithm and publish it to the production environment in ModelArts Standard Notebook, you shall check the password, AK/SK, database connection, OBS connection, and SWR connection information used in the code. Do not use fixed authentication credentials to facilitate subsequent algorithm update and maintenance. You shall encrypt the preceding sensitive information and save it in the program configuration file.

Using Independent Agencies for Different IAM Users

To use ModelArts resources, ensure that you have obtained the agency authorization from the user. For better permission control over IAM users, you shall grant agency permissions to each IAM user individually on the ModelArts global configuration page. The same agency credential shall not be shared by multiple IAM users. For details about agency authorization, see **Creating an IAM User and Granting ModelArts Permissions**.

Using Content Security Services in Generative AI

When deploying generative AI in the production environment, you need to pay attention to the varying regulatory requirements for generative AI models and applications across different regions. For example, AI-generated content might contravene fundamental societal values or touch upon locally sensitive political issues.

You must safeguard AI-generated content by deploying content security services tailored to your needs, which can filter and manage content to minimize outputs from generative AI models that conflict with local values, are harmful, or breach regional laws. Additionally, it is essential to notify local authorities prior to launching operational systems to secure necessary permissions.

To address the handling of AI-generated content, refer to content security services provided by third-party vendors. Enabling APIs allows for processing the generated content effectively. These services offer foundational content recognition and management functionalities, enabling customization according to specific production system requirements.

