

Tencent Cloud TI-ONE Platform Use Cases



Copyright Notice

©2013–2026 Tencent Cloud. All rights reserved.

The complete copyright of this document, including all text, data, images, and other content, is solely and exclusively owned by Tencent Cloud Computing (Beijing) Co., Ltd. ("Tencent Cloud"); Without prior explicit written permission from Tencent Cloud, no entity shall reproduce, modify, use, plagiarize, or disseminate the entire or partial content of this document in any form. Such actions constitute an infringement of Tencent Cloud's copyright, and Tencent Cloud will take legal measures to pursue liability under the applicable laws.

Trademark Notice

 Tencent Cloud

This trademark and its related service trademarks are owned by Tencent Cloud Computing (Beijing) Co., Ltd. and its affiliated companies("Tencent Cloud"). The trademarks of third parties mentioned in this document are the property of their respective owners under the applicable laws. Without the written permission of Tencent Cloud and the relevant trademark rights owners, no entity shall use, reproduce, modify, disseminate, or copy the trademarks as mentioned above in any way. Any such actions will constitute an infringement of Tencent Cloud's and the relevant owners' trademark rights, and Tencent Cloud will take legal measures to pursue liability under the applicable laws.

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Use Cases

LLM Deployment and Inference

- Quickly Deploying and Trying DeepSeek Series Models
- Deploying Custom LLMs (Built-in Inference Images)
- Deploying Custom LLMs (Custom Inference Images)
- Service Grayscale Deploy Practices
- Inference Acceleration Based on TensorRT-LLM
- Use Instructions for Built-in LLM Inference Images
- Guide on LLM Inference Resources
- Guide on LLM Inference Parameters
- Inference Acceleration Based on the Built-in Image Angel-vLLM

LLM Training and Evaluation

- End-to-End Use Cases: Using DeepSeek-R1 to Distill Other Models
- End-to-End Use Cases: Fine-Tuning DeepSeek-R1
- Fine-Tuning a Custom LLM Using Task-based Modeling
- Using Task-based Modeling for GRPO Reinforcement Learning Training
- Guide on Resources Required by LLM Training

LLM and MLLM Data Preparation

- Preprocessing LLM Fine-Tuning Data
- Creating an LLM&MLLM Dataset and Labeling Task
- Detailed Syntax of Schema Configuration

Practices Related to Embodied AI

- Cosmos-Predict2 Inference and Fine-Tuning Guide
- Practices of Running Isaac Sim GUI on a Dev Machine

Multimodal Model Series

- Importing and Deploying the SD-WebUI Service
- Importing and Deploying the SD ComfyUI Service

Built-in Training Image List

Custom Training Image Specifications

Introduction to the Training Acceleration Feature of Angel

Implementing Resource Isolation Between Sub-users Based on Tags

Using the Data Source Feature to Implement Directory-Level Access Control for Storage

Guide on Using CFS Instances in Various Scenarios

Guide on Data Writing Using TI-ONE Dev Machines

Guide on Running DeepSpeed Distributed Training with Task-based Modeling

Use Cases

LLM Deployment and Inference

Quickly Deploying and Trying DeepSeek Series Models

Last updated: 2026-03-03 17:54:13

Overview

DeepSeek is a LLM launched by DeepSeek. The following versions are available:

- DeepSeek-V3 is a powerful Mixture of Experts (MoE) language model pre-trained on 14.8 trillion high-quality tokens, boasting 67.1 billion parameters. As a general-purpose LLM, it demonstrates exceptional performance in domains such as knowledge Q&A, content generation, and AI customer service.
- DeepSeek-R1 is a high-performance inference model trained from DeepSeek-V3-Base, excelling in complex inference tasks such as mathematics, code generation, and logical inference.
- DeepSeek-R1-Distill is a distilled model obtained through supervised fine-tuning (SFT) of open-source models using samples generated from DeepSeek-R1. It features a smaller parameter scale and lower inference costs while still performing well in benchmark tests.

This document introduces how to quickly deploy the DeepSeek series models through **Tencent Cloud TI-ONE Platform** (TI-ONE). After a model is deployed, you can chat with the model or call it via an API for integration into AI applications.

Model List and Resource/Price Reference

The **full series of DeepSeek models** have been deployed on TI-ONE. See the table below for details.

Note:

1. The total list price includes the **CVM computing resource fee** and **TI-ONE software subscription fee**, which makes it slightly higher than just the CVM computing resource fee.
 - For the latest inventory and pricing of each CVM instance type, log in to the **CVM console** and see **Instance Creation Guide** in the CVM documentation.
 - Access to PNV6/HCCPNV6 instance types requires contacting your designated **Tencent Cloud sales representative or pre-sales architect for activation**.

2. In the pay-as-you-go mode, the TI-ONE software subscription fee is settled on a daily basis. The specific bill shall prevail. For more billing rules, see [Billing Overview](#).
3. When deploying the V3 or R1 models, if you only require a brief experience with minimal requirements for concurrency/context windows, a single-node deployment using **Standard Deployment** as the deployment mode is suitable. In other scenarios, it is recommended to perform a multi-node deployment using **Multi-machine Distributed Deployment** as the deployment mode, and ensure that at least 2 nodes are configured. The table lists only certain CVM instance specifications. For more resource recommendations, see [Guide on Resources Required by LLM Inference](#).
 - Single-node deployment: The maximum supported context length is 64K, and the minimum available CVM instance specification is PNV6.32XLARGE1280. If this instance type is out of stock, the alternative option is PNV6.96XLARGE2304.
 - Multi-node deployment: The maximum supported context length is 128K, and the available CVM instance specification is HCCPNV6.96XLARGE2304.
 - Context window: For DeepSeek V3/R1 models, the maximum context window is set to 16K by default. During deployment, this can be extended by modifying the environment variable MAX_MODEL_LEN. For example, MAX_MODEL_LEN=131072 corresponds to a maximum context window of 128K.
4. When DeepSeek distilled models from 1.5B to 70B are deployed, only a single node is required. Set the deployment mode to **Standard Deployment**.
5. The **inference resource configuration** in the table below is slightly smaller than the CVM instance configuration because, in the Select from CVM Instances mode, TI-ONE occupies a small amount of resources when managing CVM instances. For example, a CVM instance specification with 128 cores will have 125.6 cores available after the CVM instance is added to the resource group.

Model Name	Parameter Volume	Maximum Context Window	Recommended Inference Resource (Selecting from CVM Instances)	Total List Price Reference (Including the CVM Computing Resource Fee and TI-ONE Software Subscription Fee)
DeepSeek-R1	671B	Single	<ul style="list-style-type: none"> • Single-node CVM instance specification: PNV6.32XLARGE1280 	<ul style="list-style-type: none"> • Single-node Yearly/monthly subscription:

		no de 64 K	<p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Deployment mode: standard deployment</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>136,809.60 CNY/month</p> <p>Pay-as-you-go: 266.02 CNY/hour</p>
		Mu lti- no de 12 8K	<ul style="list-style-type: none"> • Multi-node <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Deployment mode: Multi-machine Distributed Deployment</p> <p>Number of nodes: 2</p> <p>Inference resource configuration (per node): 380C, 2214 GB, and 8 GPU cards</p>	<ul style="list-style-type: none"> • Multi-node <p>Yearly/monthly subscription: 190,449.60 CNY/node/month</p> <p>Pay-as-you-go: currently not supported</p>
DeepSeek-V3	67 1B	Sin gle - no de 64 K	<ul style="list-style-type: none"> • Single-node <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Deployment mode: standard deployment</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<ul style="list-style-type: none"> • Single-node <p>Yearly/monthly subscription: 136,809.60 CNY/month</p> <p>Pay-as-you-go: 266.02 CNY/hour</p>
		Mu lti- no de 12 8K	<ul style="list-style-type: none"> • Multi-node <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Deployment mode: Multi-machine Distributed Deployment</p> <p>Number of nodes: 2</p>	<ul style="list-style-type: none"> • Multi-node <p>Yearly/monthly subscription: 190,449.60 CNY/node/month</p> <p>Pay-as-you-go: currently not supported</p>

			Inference resource configuration (per node): 380C, 2214 GB, and 8 GPU cards	
DeepSeek-R1-Distill-Qwen-1.5B	1.5B	-	CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Deployment mode: standard deployment Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Yearly/Monthly subscription: 5,286.00 CNY/month Pay-as-you-go: 12.18 CNY/hour
DeepSeek-R1-Distill-Qwen-7B	7B	-	CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Deployment mode: standard deployment Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Yearly/Monthly subscription: 5,286.00 CNY/month Pay-as-you-go: 12.18 CNY/hour
DeepSeek-R1-Distill-Llama-8B	8B	128K	CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Deployment mode: standard deployment Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Yearly/Monthly subscription: 5,286.00 CNY/month Pay-as-you-go: 12.18 CNY/hour
DeepSeek-R1-Distill-Qwen-14B	14B	-	CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Deployment mode: standard deployment Inference resource configuration: 15C, 144 GB,	Yearly/monthly subscription: 17,101.20 CNY/month Pay-as-you-go: 33.25 CNY/hour

			and 1 GPU card	
DeepSeek-R1-Distill-Qwen-32B	32 B	-	CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Deployment mode: standard deployment Inference resource configuration: 15C, 144 GB, and 1 GPU card	Yearly/monthly subscription: 17,101.20 CNY/month Pay-as-you-go: 33.25 CNY/hour
DeepSeek-R1-Distill-Llama-70B	70 B	12 8K	CVM instance specification: PNV6.8XLARGE320 CVM instance configuration: 32C, 320 GB, and 2 GPU cards Deployment mode: standard deployment Inference resource configuration: 31C, 290 GB, and 2 GPU cards	Yearly/Monthly subscription: 34,202.40 CNY/month Pay-as-you-go: 66.50 CNY/hour

Guide to Online Chat Experience

If you want to experience a chat with DeepSeek models directly within a web page, it is recommended to visit the [Tencent Cloud Agent Development Platform](#) to use the [DeepSeek Web Assistant](#).

Note:

TI-ONE's Online Experience activity with limited-time free access has ended on February 24, 2025. Thank you for your support.

API Call Guide

If you want to directly call the DeepSeek API on Tencent Cloud, it is recommended to visit the [Tencent Cloud Agent Development Platform](#) to call the [DeepSeek API](#).

Practices for Model Deployment

If you need to deploy a dedicated DeepSeek LLM inference service, you can see the following instructions. In the following section, the **DeepSeek-R1-Distill-Qwen-1.5B** model, which is

relatively the smallest in size, is selected for deployment practices. The operation process for other models is similar, with the main difference being the configuration of computing resources.

Preliminary Preparations

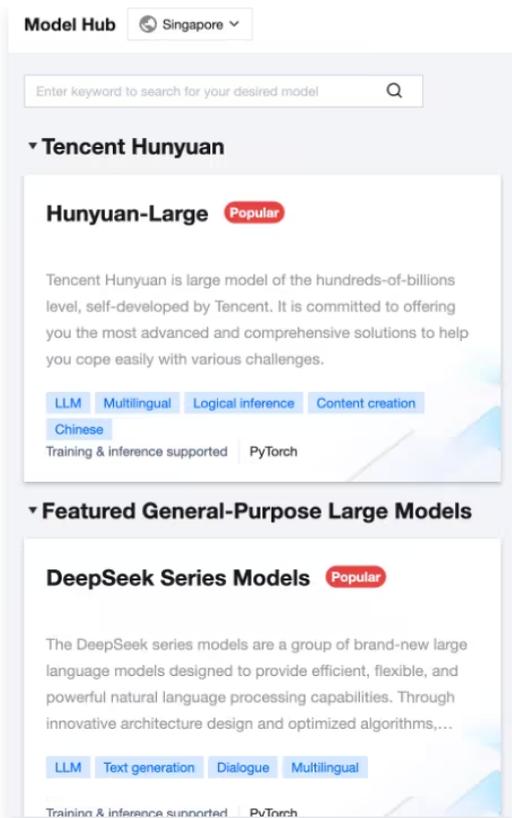
- **Models:** TI-ONE has built-in [DeepSeek](#) models in [Model Hub](#). You can directly select a model and perform one-click deployment.
- **Resources:** The 1.5B DeepSeek model has relatively low requirements for computing power, and a single A10 GPU card is sufficient to support its inference service. You have multiple billing modes to choose from:
 - **Pay-as-you-go:** For users who require a brief experience or have minimal requirements for computing power, it is recommended to set the CVM instance source to Purchase from TI-ONE and select the Pay-As-You-Go billing mode. This mode eliminates the need to prepare computing resources in advance. When you begin the deployment, the platform automatically allocates resources and starts billing. This practice will also use this mode.
 - **Yearly/Monthly subscription:** For users who have purchased CVM instances or require large, stable computing power, it is recommended to set the CVM instance source to Select from CVM Instances and select the corresponding resource group. This mode requires you to purchase CVM instances in advance and add them to the TI-ONE resource group. For detailed operation steps, see [Resource Group Management](#).

Note:

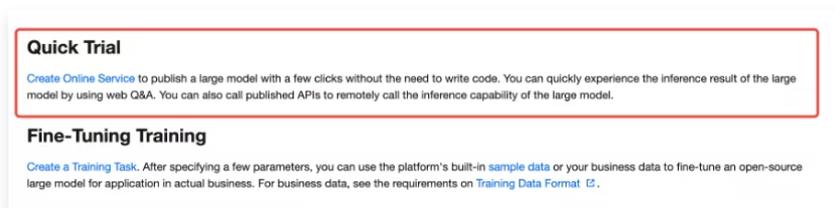
1. **Instance type purchase note:** The HCCPNV6 instance type is required to deploy DeepSeek R1 or V3 models. Contact your Tencent Cloud sales representative or pre-sales architect to enable this instance type.
2. **Model loading note:** Due to the large number of parameters of R1 and V3 models, their model size reaches 641 GB. Loading such a model from platform storage to the CVM instance alone takes considerable time (over 2 hours). Therefore, if the model is not pre-stored on the CVM instance, the overall deployment time will be considerably longer. If you have already purchased CVM instances and want to deploy DeepSeek R1 or V3 models, it is recommended to contact the TI-ONE team through your sales representative or pre-sales architect for manual acceleration support. With this support, the startup time can potentially be reduced to an estimated 10 minutes.

Step 1: Deploying Model Services

1. Log in to [TI-ONE](#). On the [Model Hub](#) page, you can view the built-in open-source LLM cards of TI-ONE.
2. Click to go to the card for the [DeepSeek series models](#) and view detailed model information.



- On the model details page, click **Create Online Service** and choose **Model Services > Online Services > Create Service** to configure deployment parameters.



- Specify the configuration information as prompted on the page as follows:
 - Service name: Enter your custom service name, such as demo-DeepSeek-R1-Distill-Qwen-1_5B.
 - CVM instance source: The Select from CVM Instances and Purchase from the TI-ONE Platform modes are supported. This example uses the **Purchase from the TI-ONE Platform - Pay-As-You-Go** mode. If you have purchased your own CVM instances, you can select the Select from CVM Instances mode. **If the required GPU instance type cannot be selected when you select the Purchase from the TI-ONE Platform option, it indicates that this instance type is currently out of stock on TI-ONE. In this case, you need to select Select from CVM Instances, and then purchase CVM instances and add them to the TI-ONE resource group. For detailed operation steps, see [Resource Group Management](#).**
 - Deployment mode: Select **Standard Deployment**.
 - Service instance:
 - Model source: Select the **Image** type.

- Model and running environment: Select **Built-in LLMs/DeepSeek Series Models/DeepSeek-R1-Distill-Qwen-1.5B**.
- CVM instance specifications: **12C, 44 GB, and 1 A10 GPU**. For recommended configurations for inference resources, see [Guide on Resources Required by LLM Inference](#).

5. Authorize and agree to the [Tencent Cloud TI-ONE Service Agreement](#), and click **Start Service** at the bottom to initiate service deployment.

Step 2: Experiencing the Model Performance

1. After the service is deployed, its status will show as Running in the list on the **Online Services page of the Model Services module**. The deployment time of the DeepSeek-R1-Distill-Qwen-1.5B model is expected to be 1 to 2 minutes.
2. Click **Online Experience** in the list to go to the quick experience page for the model. You can directly ask questions through the frontend page to experience the model's performance.



Step 3: Calling the API for Model Inference

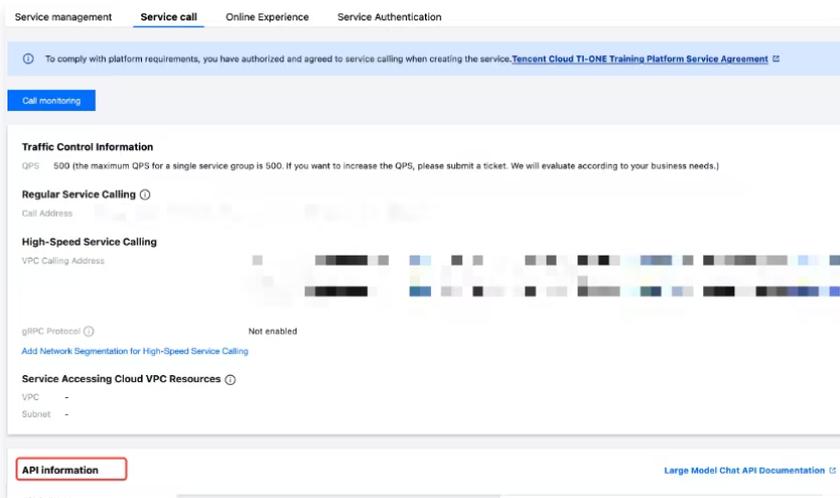
The Online Services module of TI-ONE has a built-in feature for testing API calls. Additionally, you can test API calls using tools such as command-line utilities. After the test is complete, you can integrate the model into AI applications via API calls. The following provides an example of testing and integrating the API for model inference.

Method 1: Using a Built-in Tool of TI-ONE to Test API Calls

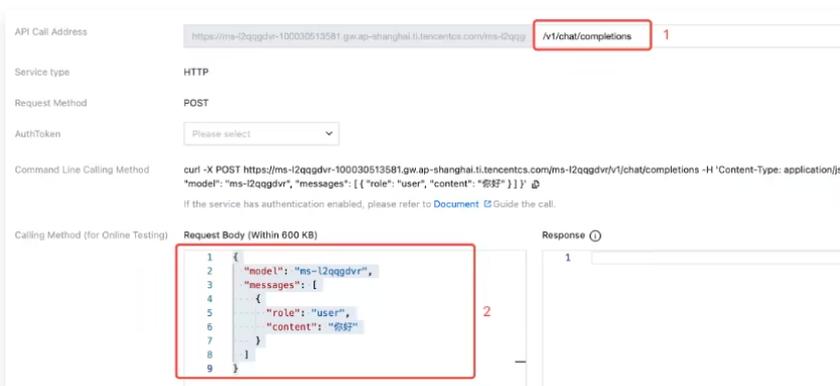
1. In the list on the **Online Services page of the Model Services module**, click the name of the newly deployed service to redirect to the service details page.



2. Go to the **Service Calling** tab on the service details page. At the bottom of the page, you can see the **API Information** section.



3. In the input box of the **API Information** section, enter the API and request information to perform API testing.



- **API name:** Enter the API name at position 1 in the figure above. For the chat API, enter `/v1/chat/completions`.
- **Remarks:** TI-ONE uses `sglang` as the inference framework for the DeepSeek LLM, which is compatible with OpenAI API specifications. For more APIs other than the chat API, see the [sglang official documentation](#).
- **Request body:** Enter the request body at position 2 in the figure above. For the request body format of the Chat Completion API, see the code below. Note that the `model` field value `ms-xxxxxxx` in the code is only an example. **Replace it with a real valid value in your own request body.**

```
{
  "model": "ms-xxxxxxx",
  "messages": [
    {
      "role": "user",
      "content": "Describe your understanding of AI."
    }
  ]
}
```

```
]
}
```

- For the **model** field, enter the **service group ID**, which is the last part of the **call address** at the top of the page. As shown in the figure below, the string marked in the red box is the **service group ID**, which is prefixed with **ms-**.



- For the **content** field, enter the specific questions you want to ask the model.

4. After specifying the information, click **Send Request** and wait a moment. The response result returned by the model will be displayed in the **Request Response** box.

Note:

When testing the API using TI-ONE's built-in tool as described above, note that the request is subject to the unified rules of the Tencent Cloud console, and will be judged as a timeout if the response time exceeds 15 seconds. If you encounter this situation, you can proceed to test the API via the command line as outlined in Method 2.

Method 2: Testing API Calls Using the Command Line Interface

1. In the above **API Information** section, enter the API name at position 1 in the figure below. After entering the information, click the copy icon at position 2 to copy the header of the API call command.



2. Append the parameter **-d'{REQ_BODY}'** to the end of the command header to obtain the complete command, where **{REQ_BODY}** is the request body. Specify the request body according to the format provided in Step 3 of the Method 1: Using a Built-in Tool of TI-ONE to Test API Calls section. The final complete command should be as shown in the code below. If you have not enabled authentication, the command header will not include the Authorization parameter.

```
curl -X POST https://ms-xxxxxxx-xxxxxxx.gw.ap-
shanghai.ti.tencentcs.com/ms-xxxxxxx/v1/chat/completions -H
'Authorization: xxxxxxxx' -H 'Content-Type: application/json' -d'{
  "model": "ms-xxxxxxx",
```

```
"messages":
[
{
"role": "user",
"content": "Describe your understanding of AI."
}
]
```

3. Paste the complete command into the command line interface of a public network-connected computing device and execute it. The output of the model will be returned in the command line.

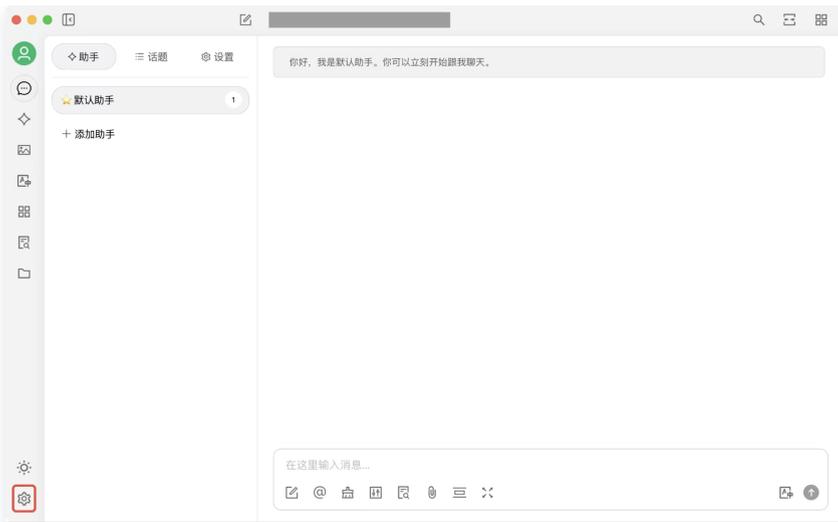
```
ms- ~ % curl -X POST https://ms- .gw.ap-shanghai.ti.tencentcs.com/
/v1/chat/completions -H 'Content-Type: application/json' -d '{
"model": "ms-",
"messages":
[
{
"role": "user",
"content": "描述一下你对人工智能的理解。"
}
]
}'
{"id": "ms-", "object": "chat.completion", "created": , "model":
"ms-", "choices": [{"index": 0, "message": {"role": "assistant", "content": "<think>\n\n嗯，用户让我描述
一下我对人工智能的理解。首先，我得明白用户的需求是什么。可能他们是刚开始学习AI，或者是想了解AI的相关领域
。我需从几个方面来展开，确保信息全面且易于理解。
\n\n首先，定义人工智能。AI是什么，为什么被定义这么重要
？然后，人类和AI WHOSE relationship is important. 这部分应该突出AI作为人类的辅助工具和替代手段。
\n\n接下来，AI的作用领域。科学、教育、医疗、金融这些领域。用户可能有特定的领域需求，所以这些例子能帮助他们更
好地应用AI知识。
\n\n然后，AI的定义。系统智能化方面，设备自适应处理任务模式，像优化系统。这一部分需要技术
术语，但又避免解释清楚。
\n\n技术意义和原理。在数据处理方面，比如无人江，强化学习的概念；用 Animated图
像来展示AI精确捕捉动物行为是什么意思。这部分要具体，让读者容易理解。
\n\n应用前景，KoIASS的应用以及量子计
算。这种方式展示了AI的未来可能，吸引对科技感兴趣的用户。
\n\n探索和挑战。开源社区，如PyTorch和TensorFlow
，是重要的让用户考虑的因素。用户可能想知道AI的发展趋势在哪里。
\n\n总结部分，AI的人机对弈关系的走向和AI伦
理。这些都是我们需要强调的，确保内容全面。
\n\n最后，检查是否有遗漏的领域，比如医疗诊断、金融监控或者农业
优化。验证这些例子是否符合用户的需求，并给出可选解决方案或建议。
\n\n总的来说，我需要结构清晰，涵盖定义、
现状、未来、探索和挑战，同时提供实际的应用例子和技术细节，确保内容全面且易于理解和应用。
\n</think>\n\n人
工智能（AI）是一个强大的技术领域，它不仅模拟了人类思考和社交活动的特点，还具有强大的潜力。从基础到应用，
```

Method 3: Using a Third-Party Tool for Application Development to Call the API

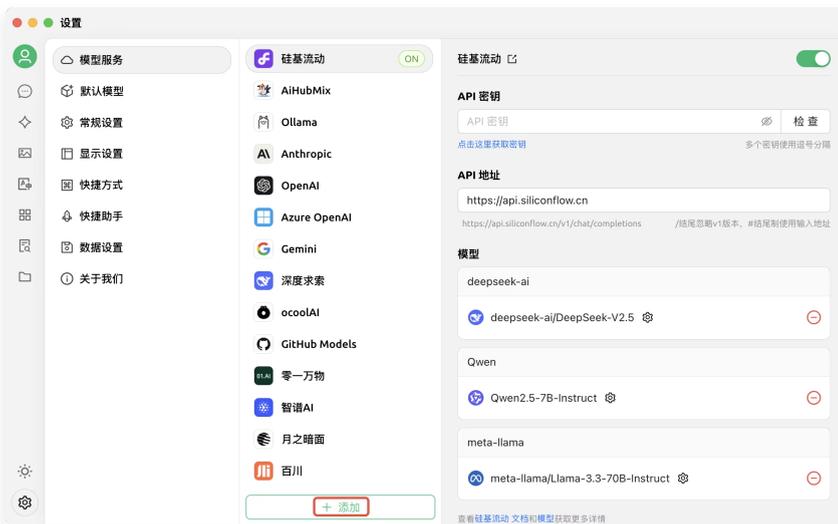
After completing model deployment, if you need to integrate the deployed model service into your AI application, you can configure the information of the service API into the relevant platform or system. The following uses [Cherry Studio](#) as an example to introduce how to integrate the service API into an application.

Cherry Studio is an open-source desktop client that supports multi-model services and can integrate multiple services into a desktop AI chat application. This document uses this client as an example to introduce API calls. If you need to use Cherry Studio for commercial purposes, carefully read its open-source software agreement.

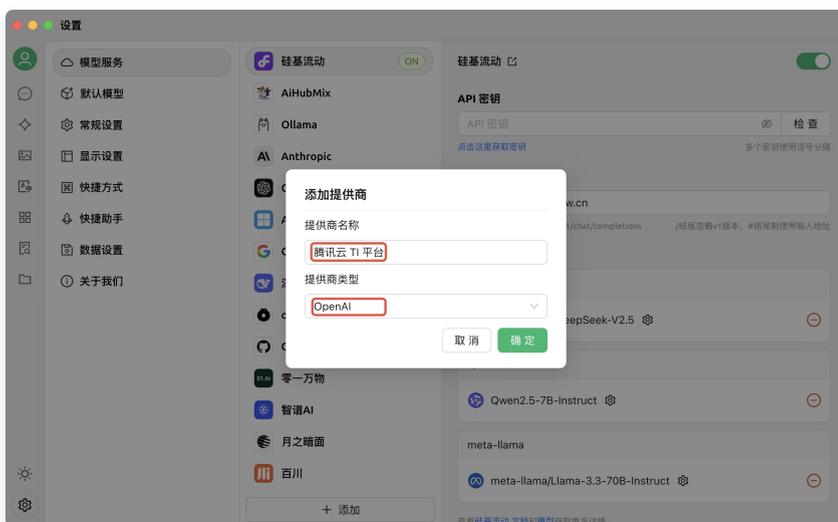
1. Go to the **Service Calling** tab of the service details page for your deployed model service on TI-ONE, find the **Calling Address** field in the upper part of the page, and click the copy icon on the far right to copy the address.
2. Download and install [Cherry Studio](#). After the installation is completed, open Cherry Studio, go to the product homepage, and click the settings icon in the lower-left corner to redirect to the product settings page.



3. After the product settings page is displayed, click **Add** at the middle bottom of the page:

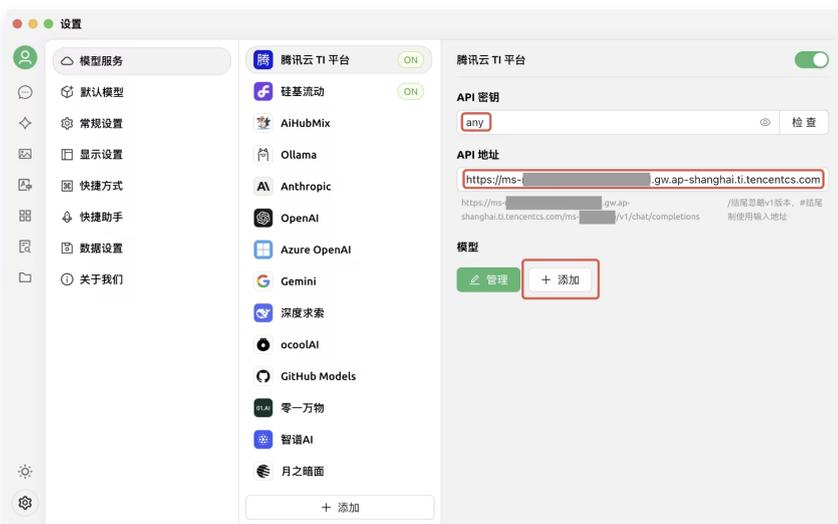


4. After clicking **Add**, you need to enter information in the pop-up Add Provider dialog box. Specify a custom provider name and set the provider type to OpenAI. After specifying the information, click **OK**:

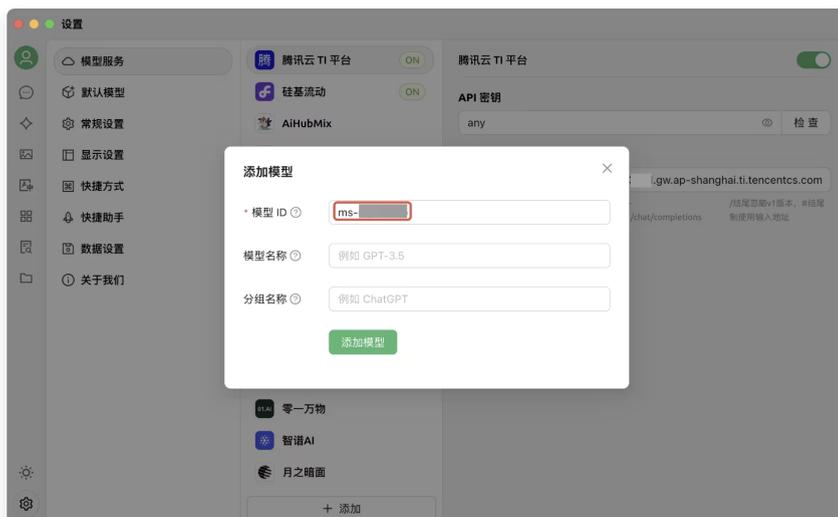


5. After successfully adding a provider as required in Step 4, you will be automatically redirected to the configuration menu of that provider. This document uses TI-ONE as an example. In the menu, perform the following configurations:

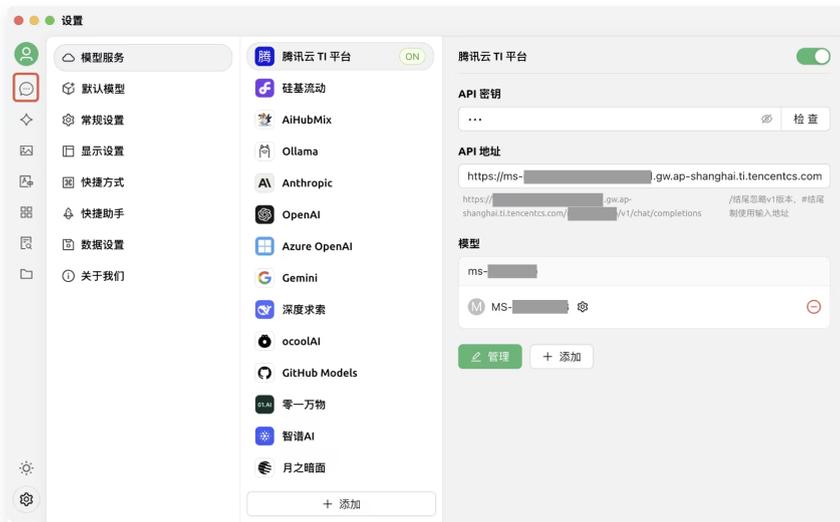
- API key:
 - The deployed service has authentication enabled: Go to the **Service Calling** tab of the service details page, find the **AuthToken** field in the upper part of the page, copy the field value, and paste it here.
 - The deployed service does not have authentication enabled: This field can be specified arbitrarily, but cannot be left empty.
- API address: Paste the **calling address** copied from Step 1.
- After the configuration is completed, click **Add** in the **Models** section below.



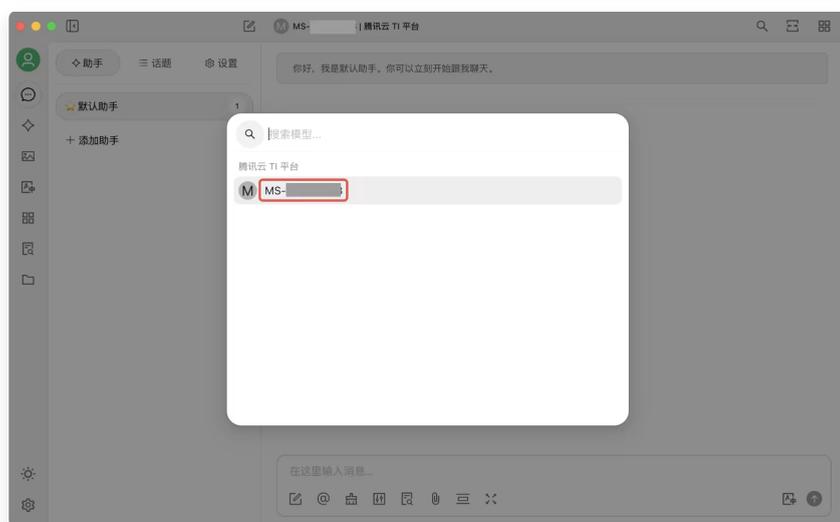
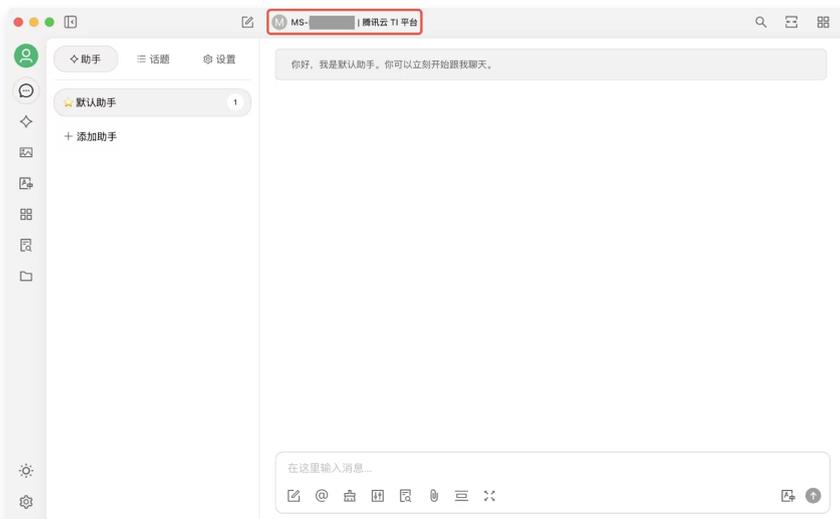
6. After **Add** is clicked, the **Add Model** dialog box will pop up. In the dialog box, enter the required information. The model ID needs to be configured as the **service group ID** of the deployed service on TI-ONE. For information about how to obtain it, see Step 3 of the Method 1: Using a Built-in Tool of TI-ONE to Test API Calls section. This field is prefixed with **ms-**. Then, click **Add Model**:



7. After successfully adding the model as required in Step 6, click the chat icon in the upper-left corner to return to the chat page:



8. Click the model selection button at the top of the chat page. After the button is clicked, a list of models will pop up. Select the model you just added.



9. Now, you can chat through Cherry Studio and the model deployed on TI-ONE:



Step 4: Managing Inference Services

You can choose **Model Services > Online Services** to view and manage an inference service on the Service Details page, including but not limited to stopping/restarting/deleting the service, and viewing the configuration information of the service, instance list, monitoring charts, container events, logs, and update records. For detailed operation guides, see [Operations of Online Services](#).

Must-Knows for Deploying Different Models

For other models of DeepSeek, the deployment process is similar to the aforementioned process, with the main difference being the resource specifications specified during service parameter configuration. See [Guide on Resources Required by LLM Inference](#) for reasonable resource configuration. If you need to deploy models with a large number of parameters, such as DeepSeek R1 or V3, the HCCPNV6 instance type is required. Contact your Tencent Cloud sales representative or pre-sales architect to enable this instance type.

Comparison of Performance Between Large and Small Models

Based on the deployed DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1 model services, we attempt to use the same question to briefly compare the inference performance of large and small models.

- Question input

In my kitchen, there is a table with a cup on it, and inside the cup is a ball. I moved the cup to the bed in the bedroom and turned it upside down. Then, I picked up the cup again and moved it to the living room. Now, where is the ball?

- Comparison of performance between large and small models

Com pare	DeepSeek-R1-Distill-Qwen-1.5B	DeepSeek-R1
-------------	-------------------------------	-------------

Model		
Screenshot example		

From the response results, it is evident that the DeepSeek-R1 model with a larger number of parameters outperforms in inference performance. It correctly deduced that the ball would fall out and remain on the bed when the cup was turned upside down, even though the cup was subsequently moved to the room. In contrast, the smaller DeepSeek-R1-Distill-Qwen-1.5B model still believed that the ball was inside the cup.

On the other hand, compared with the DeepSeek-R1 model, the DeepSeek-R1-Distill-Qwen-1.5B model with a smaller number of parameters offers faster response speed, consumes fewer resources, and has a shorter deployment duration. It remains a good choice for handling relatively simple tasks.

Among them, the deployment time of DeepSeek-R1-Distill-Qwen-1.5B is expected to be 1 to 2 minutes, while the deployment time of DeepSeek-R1 is expected to be 9 to 10 minutes. The model needs to be preloaded to the local data disk of the node.

Deploying Custom LLMs (Built-in Inference Images)

Last updated: 2026-03-03 17:43:08

Overview

This document takes the Qwen2-7B-Instruct model as an example to demonstrate how to import custom LLMs into Tencent Cloud TI-ONE Platform (TI-ONE) and deploy the LLM inference services using the built-in inference images.

Prerequisites

Applying for CFS or GooseFSx

Before deploying a custom LLM, you need to store LLM files in Cloud File Storage (CFS) or Data Accelerator Goose FileSystem extreme (GooseFSx) in advance. Therefore, you need to apply for CFS or GooseFSx in advance.

Operation Steps

1. Uploading LLM Files to CFS or GooseFSx

Log in to TI-ONE, choose **Training Workshop > Dev Machines**, and click **Create**. The instructions for specifying each field are as follows:

- **Image:** Select any **built-in image**.
- **Billing Mode:** Select either pay-as-you-go or yearly/monthly subscription billing mode. For billing rules supported by TI-ONE, see [Billing Overview](#).
- **Storage Path Settings:** Select CFS or GooseFSx. The path is the root directory / by default and is used to specify the storage location for your custom LLM.
- **Other Settings:** Not required by default.

Note:

This dev machine is only used for uploading or downloading LLM files. Therefore, select CPU Computing Power for CVM Instance Specifications.

Name • 10/256

The name can contain up to 256 characters in length, consisting of only Chinese, English letters, digits, underscores (_), and hyphens (-). It should start with a Chinese or English letter or a digit.

Region • Guangzhou

Image •

Source of Machine •

TI-ONE charges for the software and machine. For details, see [Product Pricing](#).

Billing mode • Pay-as-you-go Monthly subscription

Spec •

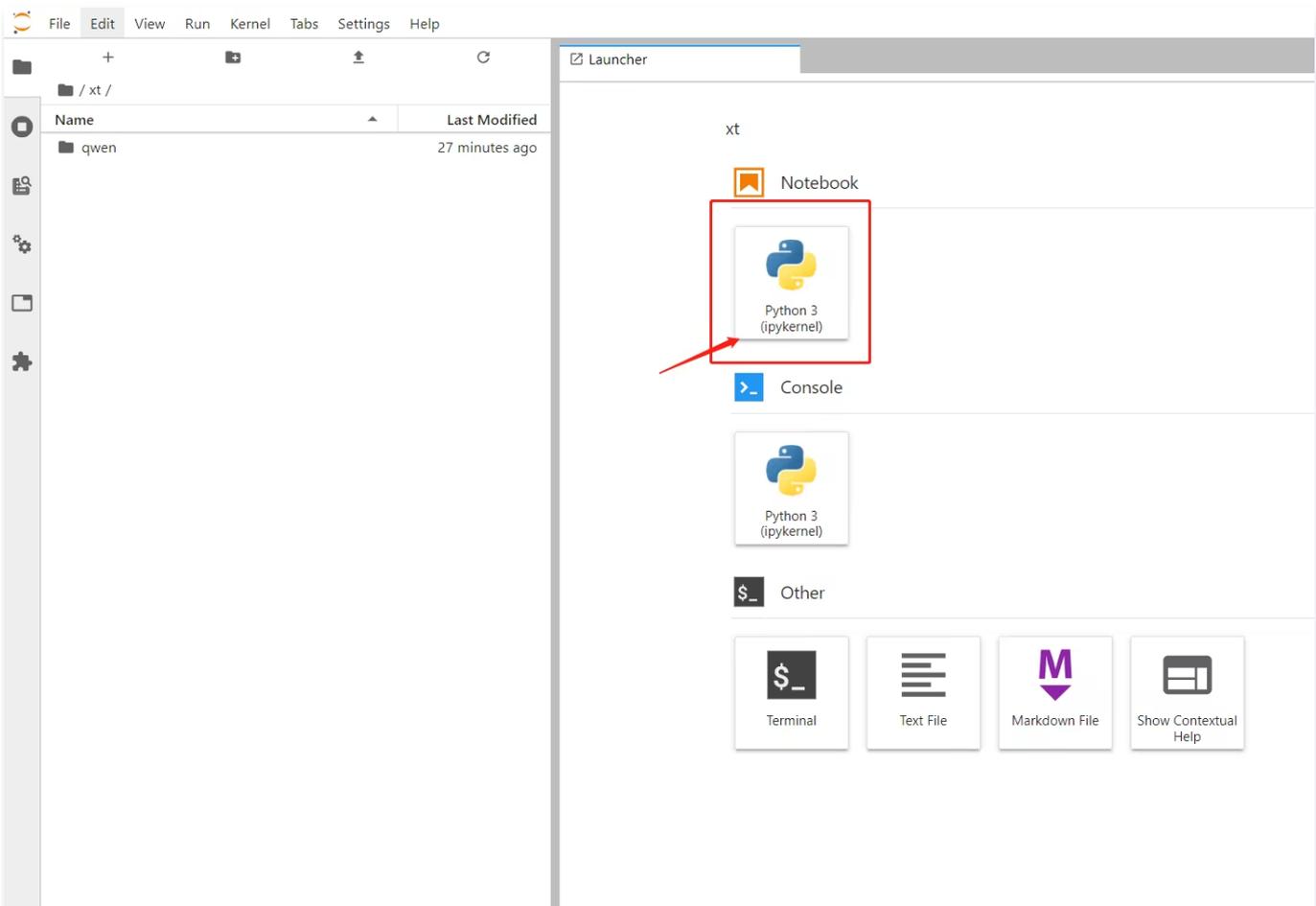
System Disk

Storage Path Settings •

To mount multiple CFS instances, ensure that the selected file systems are under the same VPC or the same CCN. By default, the system connects to the VPC network or CCN network where the first selected file system is located. CBS supports mounting only one instance.
The first configured container mount path will serve as the default working directory.

Storage Type	CFS system	Source path	Container Mounting Path	Operation
⋮ CFS	<input type="text" value="Please select"/>	<input type="text" value="/"/>	<input type="text" value="/home/tione/notebook"/>	Delete

After the creation is completed and the dev machine is in the Running status, choose **Open > Python3(ipkernel)** to download the LLM files you need via scripts.



You can search for the LLMs you need in ModelScope or Hugging Face, download the models through Python scripts in ModelScope, and save them to CFS. This document takes the **Qwen2-7B-Instruct** model as an example, and the code for download is as follows:

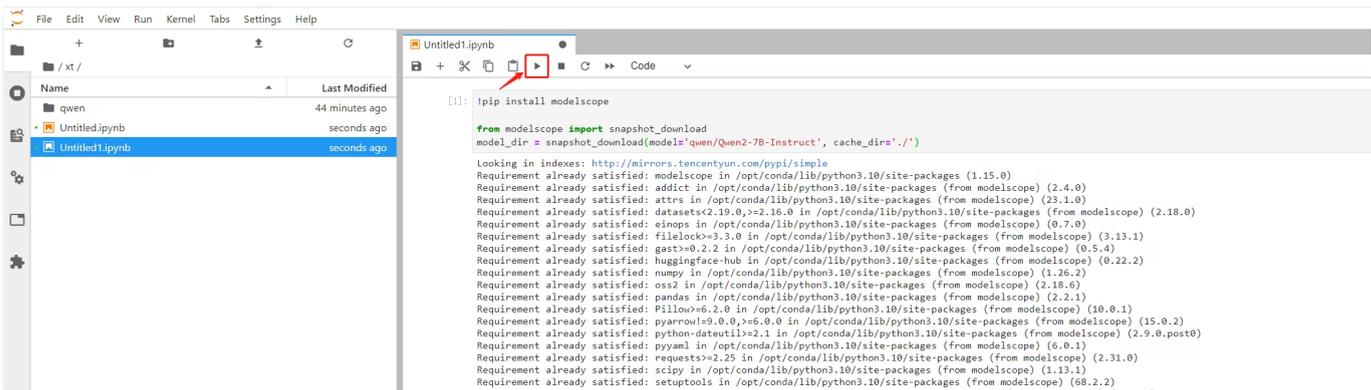
```
!pip install modelscope

from modelscope import snapshot_download
#qwen/Qwen2-7B-Instruct is the name of the model to be downloaded,
cache_dir is the address where the downloaded model will be saved, and ./
indicates saving the downloaded model in the root directory of CFS.
model_dir = snapshot_download('qwen/Qwen2-7B-Instruct', cache_dir='./')
```

Note:

Specify the address `cache_dir` (for example, `path/to/local/dir`) for downloading the model. Then, specify the model address as `/path/to/local/dir/qwen/Qwen2-7B-Instruct` in the CFS instance of Online Services.

Copy the above download script, replace relevant content with the model to be downloaded, paste the modified script into the new `.ipynb` file, and click Run to start downloading the model.



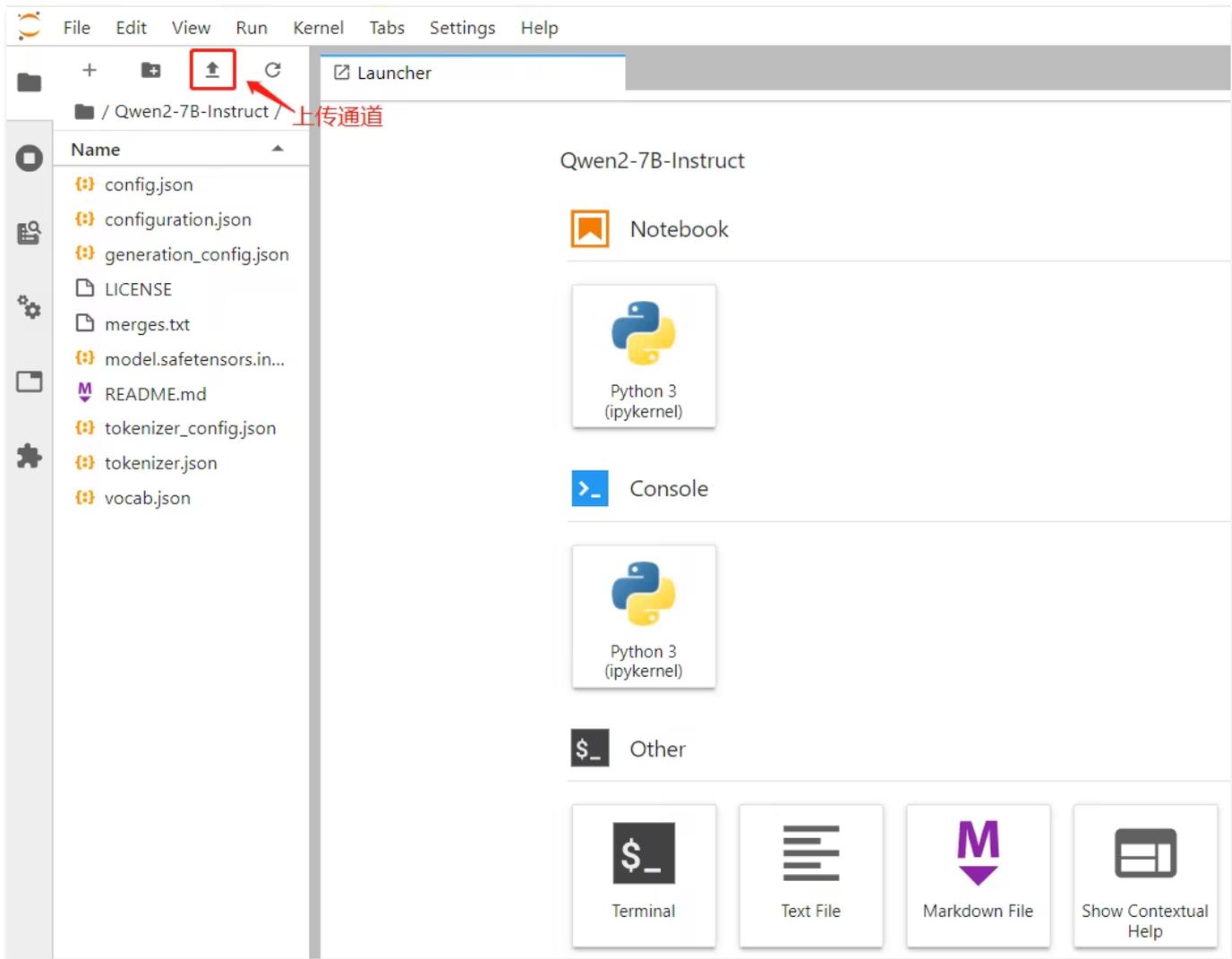
```
File Edit View Run Kernel Tabs Settings Help
Name Last Modified
qwen 44 minutes ago
Untitled.ipynb seconds ago
Untitled1.ipynb seconds ago

Untitled1.ipynb
[]: !pip install modelscope

from modelscope import snapshot_download
model_dir = snapshot_download(model='qwen/Qwen2-7B-Instruct', cache_dir='./')

Looking in indexes: http://mirrors.tencentyun.com/pypi/simple
Requirement already satisfied: modelscope in /opt/conda/lib/python3.10/site-packages (1.15.0)
Requirement already satisfied: addict in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.4.0)
Requirement already satisfied: attrs in /opt/conda/lib/python3.10/site-packages (from modelscope) (23.1.0)
Requirement already satisfied: datasets in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.18.0)
Requirement already satisfied: einops in /opt/conda/lib/python3.10/site-packages (from modelscope) (0.7.0)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (from modelscope) (3.13.1)
Requirement already satisfied: gast in /opt/conda/lib/python3.10/site-packages (from modelscope) (0.5.4)
Requirement already satisfied: huggingface-hub in /opt/conda/lib/python3.10/site-packages (from modelscope) (0.22.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (from modelscope) (1.26.2)
Requirement already satisfied: oss2 in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.18.6)
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.2.1)
Requirement already satisfied: Pillow in /opt/conda/lib/python3.10/site-packages (from modelscope) (10.0.1)
Requirement already satisfied: pyarrow in /opt/conda/lib/python3.10/site-packages (from modelscope) (15.0.2)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.9.0.post0)
Requirement already satisfied: pyyaml in /opt/conda/lib/python3.10/site-packages (from modelscope) (6.0.1)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-packages (from modelscope) (2.31.0)
Requirement already satisfied: scipy in /opt/conda/lib/python3.10/site-packages (from modelscope) (1.13.1)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-packages (from modelscope) (68.2.2)
```

Additionally, you can also download or make slight adjustments to the model locally. Then, save the model file to CFS through the upload channel of the dev machine. The upload API is shown below:



2. Creating an Online Service

Choose **Model Services > Online Services** on TI-ONE, and click **New Service** to start the inference service. The following is the guidance for service instance configuration.

- **Model Source:** Choose **Cloud Storage > CFS**.
- **Model:** Select the model file path stored in the CFS instance, which is `/qwen/Qwen2-7B-Instruct`.
- **Image:** Select `built-in/LLM/Angel-vLLM`.
- **CVM Instance Specifications:** Select resources based on the actual model size or your available resources. The Cloud Virtual Machine (CVM) instance resources required for LLM inference depend on the model parameter scale. It is recommended to configure the resources required for inference services according to the following rules.

Model Parameter Scale	GPU Card Type and Quantity
6 ~ 8B	L20 * 1 / A10 * 1 / A100 * 1 / V100 * 1

12 ~ 14B	L20 * 1 / A10 * 2 / A100 * 1 / V100 * 2
65 ~ 72B	L20 * 8 / A100 * 8

- **Advanced Settings > Environment Variables:** You need to specify the model name **MODEL_ID** (the ID of an open-source model in ModelScope or Hugging Face) and the chat template name **CONV_TEMPLATE** (if **MODEL_ID** is the same as that of the open-source model, the **CONV_TEMPLATE** parameter can be omitted). Common chat template names are listed in the table below. This document uses the qwen-chat series, so set the chat template name to qwen-7b-chat.

Chat Template Name CONV_TEMPLATE	Supported Model Series MODEL_ID
generate	Non-chat models (direct generation, without chat template)
llama-3	llama-3-8b-instruct and llama-3-70b-instruct models
llama-2	llama-2-chat series models
qwen-7b-chat	qwen-chat series models (chatml format)
baichuan2-chat	baichuan2-chat series models
baichuan-chat	baichuan-13b-chat model
chatglm3	chatglm3-6b model
chatglm2	chatglm2-6b model

The configuration for creating an instance in the Online Services module is as follows:

Model source Built-in LLM Model Repository Cloud Storage (CFS/COS) Container Image Data Source

Model *

Select CFS /Qwen2-7B-Instruct

Image *

LLM / angel-vllm

Storage Mount

Enable gRPC

Port 8501

Spec * [A Guide to Resources Required for Large Model Inference](#)

▼ Advanced settings

Startup command

Shell 0/8192

1

Environment variables

MODEL_ID	Qwen2-7B-Instruct	<input type="button" value="x"/>
CONV_TEMPLATE	qwen-7b-chat	<input type="button" value="x"/>

Note:

- If you have high requirements for inference speed, we recommend that you enable quantization acceleration by setting the environment variable **QUANTIZATION**. Valid values: "none", "ifq", "smoothquant", and "auto".
 - **none**: indicates that quantization acceleration is disabled.
 - **ifq**: indicates that online Int8 Weight-Only quantization is enabled, which can accelerate inference with basically no loss in performance and reduce the GPU memory occupied by model weights.
 - **smoothquant**: indicates that LayerwiseSearchSMQ quantization is enabled, which can further accelerate inference with slightly degraded performance (requiring pre-prepared quantized model files, currently only supported for specific models).
 - **auto**: indicates that the quantization mode is automatically determined:

- If the GPU of an instance type does not support quantization, quantization is automatically disabled.
- If the model directory contains the `smoothq_model-8bit-auto.safetensors` file, LayerwiseSearchSMQ quantization acceleration is automatically enabled.
- In other cases, online Int8 Weight-Only quantization acceleration (ifq) is enabled by default.
- If the log reports a **CUDA out of memory** error after the service is enabled, the default value, 32 KB, of the `max-model-len` parameter of the model is too large (the maximum number of context tokens supported by the inference service, which is the context length automatically read from the model configuration information by default). If the model is loaded with a large default context length, it may cause insufficient GPU memory. You can set a smaller value (such as 16 KB or 8 KB) via the `MAX_MODEL_LEN` environment variable or enable quantization acceleration to reduce the GPU memory occupied by model weights.

3. Experiencing Chatting with LLMs on the Frontend Online

Go to the details page of the created online service and chat with the deployed LLM by clicking the **Online Demo** tab.

4. Calling an API Service

You can access a service by clicking the **Service Call** tab and choosing **API Information > Call Method (Online Testing)**. The API calling address is `${SERVER_URL}/v1/chat/completions`, and the format of the request body is as follows:

```
{"messages":[{"role": "user", "content": "Who are you"}]}
```

The content field indicates the specific message content.

The public network access address can be obtained from the **Service Call** tab of the online service instance. An API call example is as follows:

```
# Public network access address.
SERVER_URL = https://ms-gp6rjk2jj-*****.gw.ap-
shanghai.ti.tencentcs.com/ms-gp6rjk2j

# Non-streaming request.
curl -H "content-type: application/json" ${SERVER_URL}/v1/chat/completions
-d '{"messages":[{"role": "user", "content": "Hello"}], "temperature":
```

```
0.0}'
# Streaming request.
curl -H "content-type: application/json" ${SERVER_URL}/v1/chat/completions
-d '{"messages":[{"role": "user", "content": "Hello"}], "temperature":
0.0, "stream": true}'
```

Non-streaming returned results:

```
{"id":"chatcmpl-
4aeRgYwnaYe4RzmmcyKtYs","object":"chat.completion","created":1698291242,"m
odel":"baichuan-13b-chat","choices":[{"index":0,"message":
{"role":"assistant","content":"Hello! How can I assist you
today?","finish_reason":"stop"}],"usage":
{"prompt_tokens":4,"total_tokens":16,"completion_tokens":12}}
```

Streaming returned results:

```
data: {"id": "chatcmpl-hn5mCVt4szVVZBa4fVFZWF", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"role": "assistant"},
"finish_reason": null}]}
```

```
data: {"id": "chatcmpl-hn5mCVt4szVVZBa4fVFZWF", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "You"},
"finish_reason": null}], "usage": {"prompt_tokens": 4, "total_tokens": 5,
"completion_tokens": 1}}
```

```
data: {"id": "chatcmpl-hn5mCVt4szVVZBa4fVFZWF", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "OK"},
"finish_reason": null}], "usage": {"prompt_tokens": 4, "total_tokens": 6,
"completion_tokens": 2}}
```

```
.....Intermediate results are omitted here.....
```

```
data: {"id": "chatcmpl-hn5mCVt4szVVZBa4fVFZWF", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "?"},
"finish_reason": null}], "usage": {"prompt_tokens": 4, "total_tokens": 15,
"completion_tokens": 11}}
```

```
data: {"id": "chatcmpl-hn5mCVt4szVVZBa4fVFZWF", "object":
"chat.completion.chunk", "created": 1714120317, "model": "baichuan-13b-
```

```
chat", "choices": [{"index": 0, "delta": {}, "finish_reason": "stop"}],
"usage": {"prompt_tokens": 4, "total_tokens": 16, "completion_tokens":
12}}

data: [DONE]
```

Alternatively, you can also use the service via the requests library that is commonly used in Python. Below is a demo example of a chat interaction between command lines and the Qwen2-7B-Instruct LLM inference service:

```
import argparse
import requests
import json

def chat(messages):
    data = {
        "messages": messages,
        "temperature": args.temperature,
        "max_tokens": args.max_tokens,
        "top_p": args.top_p,
        "stream": True, # Enable streaming output.
    }
    header = {
        "Content-Type": "application/json",
    }
    if args.token:
        header["Authorization"] = f"Bearer {args.token}"

    response = requests.post(f"{args.server}/v1/chat/completions",
        json=data, headers=header, stream=True) # Set the stream=True parameter
    to obtain real-time data streams.
    if response.status_code != 200:
        print(response.json())
        exit()

    result = ""
    print("Assistant: ", end = "", flush = True)
    for part in response.iter_lines():
        if part:
```

```
        if "content" in part.decode("utf-8"):
            content = json.loads(part.decode("utf-8")[5:])["choices"]
[0]["delta"]["content"] # Filter the string to remove the data: prefix,
convert the string into JSON format, and then extract the text.
            result += content
            print(content, end = "", flush = True)
        print("\n")

    return result

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Chat CLI Demo.")
    # Replace --server with the actual calling address.
    parser.add_argument("--server", type=str, default="https://ms-
gp6rjk2j-*****.gw.ap-shanghai.tenentcs.com/ms-gp6rjk2j")
    parser.add_argument("--max-tokens", type=int, default=512)
    parser.add_argument("--temperature", type=float, default=0.7)
    parser.add_argument("--top_p", type=float, default=1.0)
    parser.add_argument("--system", type=str, default=None)
    parser.add_argument("--token", type=str, default=None)
    args = parser.parse_args()

    messages = []
    if args.system:
        messages.append({"role": "system", "content": args.system})

    while True:
        user_input = input("User: ")
        messages.append({"role": "user", "content": user_input})
        response = chat(messages)
        messages.append({"role": "assistant", "content": response})
```

Deploying Custom LLMs (Custom Inference Images)

Last updated: 2026-03-12 10:45:36

Overview

This document introduces how to deploy custom LLM inference services using open-source inference images (vLLM or SGLang) via the Online Services module of Tencent Cloud TI-ONE Platform (TI-ONE).

Preparing a Model

First, upload the model files that need to be deployed to your Cloud File Storage (CFS) instance. The following two upload methods are recommended:

- For a local model, first upload it to [Cloud Object Storage \(COS\)](#), choose **Training Workshop > Dev Machines** in TI-ONE to mount CFS, and use COSCMD or COSCLI to download the model to CFS.
- For a model to be downloaded from Hugging Face or ModelScope, choose **Training Workshop > Dev Machines** on TI-ONE to mount CFS, install the Software Development Kit (SDK) of Hugging Face or ModelScope, and use the corresponding download command to download the model.

Deploying an Online Service

Creating a Service

Log in to the TI-ONE console, choose **Model Services > Online Services**, and click Create Service to start the inference service. The instructions for filling in each field are as follows:

- **Deployment Mode:** If the model can be deployed on the GPU of a single Cloud Virtual Machine (CVM) instance, select **Standard Deployment**. If the model is very large and requires at least 2 CVM instances for deployment, select **Multi-machine Distributed Deployment**.
- **Source of Machine:** Select resources as needed. It is recommended to click Select from CVMs and select an appropriate resource group.
- **Replica Settings:**
 - **Model Source:** Select **Cloud Storage**. If you want to accelerate service startup and the CVM instance in your resource group has sufficient hard disk space, load the model files to the hard disk of the CVM instance in advance by reference to [File Preloading](#), and then select **Resource Group Preloading**.

- **Model:** If the source is CFS, select the file path of the model on CFS.
- **Image:** Select **Custom**, select **Image Address** in the pop-up dialog box, and then fill in the image address to be used. For specific instructions, see [Selecting an Image](#).
- **Storage Mounting:** This parameter is optional and can be configured if other data paths need to be mounted for your service besides the model. Generally, it does not need to be configured.
- **Enable gRPC:** This parameter is not selected.
- **Port:** The default value 8501 is recommended.
- **Resource:** Configure the required GPU, CPU, and memory resources as needed. For multi-machine distributed deployment, additionally configure the number of resource instances for a single model service, such as 2.
- **Advanced Settings:** Configure the startup command. For specific instructions, see [Startup Command](#).

Source of Machine

Select from CVMs

Purchased on TI-ONE

TI-ONE only charges software fees. The computing power will use the machines you have purchased. For details, see [Product Pricing](#).

Resource group

Select a resource group
▼
↻

Ensure the selected storage instance (CFS or GooseFSx) is interconnected with the node network of the data source instance and resource group.

Model source

Built-in LLM
 Cloud Storage (CFS/GooseFSx/COS)
 Container Image
 Resource Group Preloading
 Data Source

Model Repository

Model

CFS

GooseFSx

COS

Select CFS
▼
↻

CFS Source Path

/data/model

Image

Built-in

Custom

Select Custom Image

Select Image

Storage Mount

+ Add

Enable gRPC

🔌

Port

8501
✎

Resource

Select a gpu type

A Guide to Resources Required for Large Model Inference

-

0.0

+

card(s)

To use GPU, set the number of cards to 0.1 to 1 or a multiple of 1 according to the gpu type. The number of cards cannot be 0 if the runtime environment is the built-in GPU image of the platform.

-

1.0

+

core

-

1.00

+

G

▶ Advanced settings

Selecting an Image

You can search for available open-source image versions from Docker Hub:

©2013–2026 Tencent Cloud. All rights reserved.

Page 31 of 317

- [vLLM](#)
- [SGLang](#)

It is recommended to deploy the model using a stable version image with a version number, such as `vllm/vllm-openai:v0.11.0`, or `lmsysorg/sglang:v0.5.5`.

Certain newer open-source models may need development branches of open-source frameworks for deployment. You may also try to deploy your model using the image committed by the nightly version.

Note:

1. To accelerate pulling the image from Docker Hub, when filling in the image address, you can add `mirror.ccs.tencentyun.com`, which is a Tencent Cloud image pulling proxy prefix. For example, if the open-source image address is `vllm/vllm-openai:v0.11.0`, fill in the image address as `mirror.ccs.tencentyun.com/vllm/vllm-openai:v0.11.0`.
2. To ensure a stable version of the image you use, it is not recommended to use tags such as `latest`, `dev`, `main`, or `nightly`, as images corresponding to these tags without version numbers, commit hashes, or dates may be updated frequently. This can lead to changes in the actual image during service restarts, potentially causing compatibility issues.
3. Whether you use stable versions or development versions of open-source images, they may inevitably contain bugs. It is recommended that you thoroughly test them before integrating your inference service into the production environment.

Startup Command

vLLM Framework

Standard Deployment

The basic startup command is as follows:

```
vllm serve /data/model --port 8501 --host 0.0.0.0 --served-model-name
$SERVED_MODEL_NAME -tp $TP
```

Where:

- **SERVED_MODEL_NAME:** Fill in the model name for the service. By default, vLLM will verify whether the model field in the request matches the `--served-model-name` parameter configured during startup.
- **TP:** If the service uses multi-GPU card TP, fill in the number of GPU cards used by the service.

If the open-source model has other [recommended startup parameters](#) from vLLM, you can try to configure them according to the instructions.

The startup parameters supported by the vLLM framework may change across versions. See the [documentation for the corresponding version](#).

Multi-machine Distributed Deployment

vLLM multi-machine distributed deployment is slightly more complex. It requires initializing the Ray cluster first and then starting up the inference service on the head node of the Ray cluster. The basic startup command is as follows:

```
if [ "$RANK" = "0" ]; then \  
ray start --head --port 6700 --include-dashboard false --disable-usage-\  
stats \  
&& vllm serve /data/model --port 8501 --host 0.0.0.0 --served-model-name \  
$SERVED_MODEL_NAME -tp $TP; \  
else ray start --address $MASTER_ADDR:6700 --block; fi
```

Where:

- **SERVED_MODEL_NAME:** Fill in the model name for the service. By default, vLLM will verify whether the model field in the request matches the `--served-model-name` parameter configured during startup.
- **TP:** If the service uses multi-GPU card TP, fill in the number of GPUs used by the service. For example, for a two-machine, 16-GPU TP setup, fill in 16.
- **MASTER_ADDR and RANK:** They are multi-machine distributed environment variables automatically injected by the platform and do not require modification.

ⓘ Note:

In version v0.11.0, the vLLM framework changed the default value of the environment variable `VLLM_ALLREDUCE_USE_SYMM_MEM` to 1. This causes vLLM to encounter exceptions when obtaining the GPU index if multi-machine distributed deployment does not cover the whole machine. This issue can be temporarily avoided by setting the environment variable `VLLM_ALLREDUCE_USE_SYMM_MEM=0`.

SGLang Framework

Standard Deployment

The basic startup command is as follows:

```
python3 -m sglang.launch_server --model-path /data/model --host 0.0.0.0 --\  
port 8501 --enable-metrics --tp $TP
```

Where:

- `--enable-metrics`: By default, the SGLang framework disables monitoring metrics reporting. It is recommended to enable it so that the platform can collect LLM metrics of the service.
- `TP`: If the service uses multi-GPU card TP, fill in the number of GPU cards used by the service.

The startup parameters supported by the SGLang framework may change across versions. See the [documentation for the corresponding version](#).

Multi-machine Distributed Deployment

The basic startup command is as follows:

```
python3 -m sglang.launch_server --model-path /data/model --host 0.0.0.0 --
port 8501 --enable-metrics --tp $TP \
  --dist-init-addr $MASTER_ADDR:$MASTER_PORT --nnodes $WORLD_SIZE --node-
rank $RANK
```

Where:

- `TP`: If the service uses multi-GPU card TP, fill in the number of GPUs used by the service. For example, for a two-machine, 16-GPU TP setup, fill in 16.
- `MASTER_ADDR`, `MASTER_PORT`, `WORLD_SIZE`, and `RANK`: They are multi-machine distributed environment variables automatically injected by the platform and do not require modification.

FAQs

When the Number of GPU Cards Used Is Less Than 1, Why Is a CUDA Out of Memory (OOM) Error Reported During Startup of the Open-Source vLLM Image?

When the number of GPU cards is less than 1, the qGPU feature is automatically enabled. This feature is currently incompatible with CUDA versions later than 12.2. Therefore, if the CUDA version of the vLLM image is later than 12.2, statistics exceptions will occur during the GPU memory profiling stage, leading to a CUDA OOM error and exit. To use the qGPU capabilities normally, you need to use `cuda-compat-12-2` to downgrade the CUDA runtime version in the container. You can modify the startup command as follows:

```
wget
"https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_6
4/cuda-compat-12-2_535.216.01-0ubuntu1_amd64.deb" \
```

```
&& dpkg -i cuda-compat-12-2_535.216.01-0ubuntu1_amd64.deb \  
&& export LD_LIBRARY_PATH=/usr/local/cuda-12.2/compat:$LD_LIBRARY_PATH \  
&& vllm serve ...
```

If the download of the `cuda-compat` package fails due to network issues, it is recommended to download it in advance, upload it to your COS, and download it from COS during startup. Alternatively, you can upload it to CFS and mount it as additional storage.

GPU Memory Is Sufficient for the Model Weights, But Why Is a CUDA OOM Error Still Reported During Startup?

In addition to the model weights, KV Cache consumes a significant amount of GPU memory during LLM inference. This GPU memory usage is related to the configured context length. Certain open-source models have long default context lengths (such as 256K and 1M). However, in many scenarios, such a long context is not needed. You can reduce the context length during startup. An example startup command is as follows:

```
# vllm  
vllm serve ... --max-model-len 16384  
  
# sglang  
python3 -m sglang.launch_server ... --context-length 16384
```

Process of Loading Model Weights From CFS Is Very Slow. How Can It Be Accelerated?

In the current open-source framework, the default model weight loading logic is optimized for local storage of the model. However, this method is inefficient for network file storage such as NFS and CFS Standard Storage. We can adjust the weight loading mode to optimize speed, but this may increase memory usage. It is recommended that the memory allocated to the container be greater than or equal to the GPU memory.

- The vLLM framework supports the [safetensors-load-strategy parameter](#) starting from version v0.10.2.
- The SGLang framework supports the [weight-loader-disable-mmap parameter](#) starting from version v0.5.1.

Reference for the startup command:

```
# vllm  
vllm serve ... --safetensors-load-strategy eager
```

```
# sglang  
python3 -m sglang.launch_server ... --weight-loader-disable-mmap
```

If the service may be updated or restarted frequently, it is recommended to use the file preloading feature of the platform to further accelerate the loading of model weights. This method is local storage loading and does not require configuration of additional parameters.

Service Grayscale Deploy Practices

Last updated: 2026-03-03 17:43:54

This practice guide details how to leverage the service version and traffic allocation features on Tencent Cloud TI-ONE Platform (TI-ONE) to achieve grayscale release for model services: smoothly transitioning the model of online services from DeepSeek-V3 to the new version DeepSeek-V3.1.

Note: Before performing the upgrade, evaluate the compatibility of the new and old model versions with your production business, and confirm that it is safe to perform the upgrade before proceeding with the steps in this guide.

Prerequisites

- **Current status:** Your online service is using the DeepSeek-V3 model, with service version V1. The service is running stably, handling 100% of online traffic, with 1 instance.
- **Objective:** Smoothly upgrade the service to the latest DeepSeek-V3.1 model, gradually roll it out as the new version V2, and eventually replace the earlier version completely.
- **Resource preparation:** Prepare sufficient Cloud Virtual Machine (CVM) instance resources. At a minimum, you need to launch an additional V2 service instance alongside the existing V1 for verification of the initial traffic switch.

Operation Steps

Model Independent Verification

Before integrating a new model version into production traffic, run independent feature tests to ensure that the inference results and API behavior meet your expectations.

1. Creating a Service

Choose **Model Services > Online Services** on TI-ONE, click **Create Service**, choose **Built-in LLM > DeepSeek - DeepSeek-V3** on the parameter configuration page, configure relevant resources and parameters, and then start the service and wait for it to run.

Model source Built-in LLM Cloud Storage (CFS/GooseFSx/COS) Container Image Resource Group Preloading Data Source Model Repository

Model

Image

Port

Resource [A Guide to Resources Required for Large Model Inference](#)

Currently available gpu type: GPU-PNV4

GPU card(s)

To use GPU, set the number of cards to 0.1 to 1 or a multiple of 1 according to the gpu type. The number of cards cannot be 0 if the runtime environment is the built-in GPU image of the platform.

CPU core

Currently available CPU: 11.70 core ,recommend125.00 core

Memory G

Currently available memory: 38.61 G ,recommend1207.00 G

Advanced settings

2. Testing APIs

After the service is in the **Running** status, you need to use test scripts or tools to call the service API for detailed parameter verification. In this case, you can click the service name to go to the details page and directly initiate service requests on the **Service Call** page.

This practice scenario primarily validates and analyzes a simple comparison between DeepSeek-V3 and DeepSeek-V3.1. The general scenario analysis indicates that the response packets of DeepSeek-V3.1 and DeepSeek-V3 are compatible. The analysis process is shown in the following table.

Scenario	Request Body	DeepSeek-V3 Response	DeepSeek-V3.1 Response
General request			

Use of Function Call			
JSON formatted output			

In addition to the above items analyzed in this practice, we recommend that you additionally use real business requests to test and analyze the service according to your different business scenarios. For example:

- **Token statistics:** Call APIs and check whether the token usage (prompt_tokens and completion_tokens) in the returned results is properly calculated.
- **Chain of thought:** If the model supports the chain of thought, verify whether the returned chain of thought is clear in process and logically self-consistent, and whether the content of the returned fields (content and reasoning_content) is correct.
- **Other parameters:** If other parameters are used in a call, such as temperature, top_p, and max_new_tokens, verify that these parameters are still valid and work as expected in the new version.

In addition, you can also focus on the verification of advanced features of certain services as needed:

- **Function call:** If the function call is supported by your model and the function call feature is applied in your application, simulate a call and check whether the model can correctly understand your intent and generate an expected JSON object for calling the function.
- **JSON object output:** If JSON object output is supported by your model and your application requires JSON-formatted output (such as via response_format: json_object), verify whether the JSON object generated by the new version complies with the preset schema, with complete fields of correct types.

3. Result Comparison

Compare the test results of the new version model with the known stable results of the current model to confirm that the new version model has no functional or behavioral issues. If any issues are identified during verification, immediately stop the grayscale release plan and address these issues at the model or code level for redeployment. After the compatibility of the new version model is confirmed, proceed with the grayscale release operation.

Formal Deployment of the New Version Model

1. In the TI-ONE console, choose **Model Services** > **Online Services**, and find the online service released using your earlier version of the DeepSeek-V3 model.
2. Click the service name to go to the service details page, and then click **Add Version** to prepare for releasing a new service version to deploy a new model.
3. On the parameter configuration page for the new version, select DeepSeek series: DeepSeek-V3.1 in the built-in LLM list, configure corresponding resources and service parameters, and then start the service.

The screenshot shows the parameter configuration page for a new service version. The 'Model source' is set to 'Built-in LLM'. The 'Model' is 'DeepSeek / DeepSeek-V3.1'. The 'Image' is 'tione.tencentcloudcr.com/qcloud-ti-platform/llm-infer:sglang-v0.5.6.post2-'. The 'Port' is '8501'. The 'Resource' section includes:

- GPU Type:** A dropdown menu set to 'Select a gpu type'. A link 'A Guide to Resources Required for Large Model Inference' is provided. Below it, 'Currently available gpu type: GPU-PNV4' is displayed.
- GPU:** A numeric input field set to '8' with minus and plus buttons, followed by 'card(s)'. Below it, a note states: 'To use GPU, set the number of cards to 0.1 to 1 or a multiple of 1 according to the gpu type. The number of cards cannot be 0 if the runtime environment is the built-in GPU image of the platform.'
- CPU:** A numeric input field set to '125.0' with minus and plus buttons, followed by 'core'. Below it, 'Currently available CPU: 11.70 core ,recommend125.00 core' is displayed.
- Memory:** A numeric input field set to '1207.00' with minus and plus buttons, followed by 'G'. Below it, 'Currently available memory: 38.61 G ,recommend1207.00 G' is displayed.

At the bottom left, there is a section for 'Advanced settings'.

4. After the deployment of the new version service is completed, both V1 and V2 are in Running status, and the platform sets the value of Traffic Allocation to 0 for the new version, V2, by default. At this point, no service requests will be allocated to V2.

Grayscale Verification with Low Traffic (First Stage)

After the new version service is successfully deployed, proceed with the first stage of grayscale verification by directing a small portion of traffic to V2 to observe its performance.

1. Allocating Traffic Weights

Click the service name to go to the service details page, and then click **Allocate Traffic** to adjust the traffic weight for V2 from 0 to 10 and adjust the traffic weight for the earlier version, V1, to 90. In this case, 10% of the service request traffic will be allocated to V2.

2. Monitoring and Observing

On the monitoring page, click Request Load to view the load of each version and verify whether the load aligns with the expected traffic weight. For example:

- Observe the load of V2.
- Observe the load of V1.
- Check the logs of V2.

It is recommended that you continuously monitor the load and logs for 30 minutes to 1 hour. You may also extend the observation period to one or more days according to your actual business conditions. The following aspects are suggested for monitoring:

- Alarm monitoring: Check the alarms configured in Tencent Cloud Observability Platform (TCOP) and observe if any alarms are provided during the upgrade period, such as exceptions like instance restarts.
- Core metrics: Observe metrics such as the number of requests, time to first token (TTFT), and output rate of the new version, V2, to ensure its performance meets expectations.
- Log analysis: Check the logs of V2 to confirm whether there are any exceptions.
- Business metrics: Verify from the business side the accuracy of model inference results and whether the API responses meet expectations.

Gradual Increase in Traffic (Second Stage)

If the monitoring results of the first stage are positive, the DeepSeek-V3.1 is stable and reliable. You can gradually increase the grayscale release traffic.

Note: Before executing this step, you can increase the traffic weight for V2 by stages (for example, from 10 to 20 and 30) and continuously observe the system performance. Ensure the number of service instances matches the loaded traffic.

1. Adjusting the Number of Instances

This document takes a single instance as an example and therefore does not demonstrate the actual operations of adjusting the number of instances. However, in actual scenarios, if there are many V1 instances, you need to synchronously adjust the number of V2 instances when adjusting traffic weights to ensure the number of instances matches the allocated traffic weight.

2. Adjusting Traffic Weights

First, increase the traffic weight for V2 from 10 to 50, and then, reduce the traffic weight for V1 to 50. Finally, 50% of the traffic is allocated to each version.

3. Keeping Monitoring

View the monitoring information and logs of V2.

Monitoring and observation suggestions for this stage are as follows:

- Conduct monitoring for a longer period, as this phase handles more online traffic and can expose potential issues more comprehensively.
- Pay special attention to business performance to ensure that the business side can correctly process the results returned by the new version, even when traffic increases.

Rapid Rollback (Any Stage)

At any stage of the grayscale release, if you encounter any issues with the new version, such as unexpected API responses, performance degradation, or degraded output quality, you can immediately execute a rollback operation to switch all traffic back to the original version, V1, to keep the online services unaffected.

1. Switching Traffic Back to V1

On the service management page, use the traffic allocation feature to set the traffic weight for V1 to 100 and that for V2 to 0.

2. Observing Business Recovery and Stopping the New Version

After traffic is switched, observe the business status. After confirming the business is recovered, stop the new version, V2, to release resources. After all issues of the new version are resolved, proceed with the next grayscale release process.

Full Upgrade and Resource Cleanup (Third Stage)

After the new service version, V2, has been running stably at 50% traffic for a period of time, switch full traffic to V2 finally.

Note: Before a full upgrade is performed, it is recommended to continue increasing the traffic weight for V2 by stages (for example, from 50 to 60 and 70) and continuously observe the system performance. Ensure the number of service instances matches the loaded traffic.

1. Full Traffic Switch

Increase the traffic weight for V2 to 100 and reduce the traffic weight for V1 to 0.

2. Service Scale-In and Decommissioning of the Old Version

Check the monitoring information and logs of V1. After confirming that all online traffic has been successfully switched to V2, reduce the number of V1 instances to 0.

By following these grayscale release steps, you can smoothly upgrade your service model from DeepSeek-V3 to DeepSeek-V3.1 while ensuring service stability. If any issues arise during the process, you can adjust traffic weights and the number of instances at any time to promptly resolve these upgrade-related issues.

Inference Acceleration Based on TensorRT-LLM

Last updated: 2026-03-03 16:58:39

Overview

TensorRT-LLM is a Large Language Model (LLM) inference acceleration framework developed by NVIDIA. It provides user-friendly Python APIs and uses the latest optimization techniques to build LLMs into **TensorRT** engine files, enabling efficient inference on NVIDIA GPUs.

TensorRT-LLM also provides a backend that can be integrated with **NVIDIA Triton Inference Server** for deploying models as online inference services and supports the In-Flight Batching technology, which can significantly improve the throughput of inference services and reduce latency.

This document uses the Baichuan2-13B-Chat model as an example to demonstrate how to perform inference acceleration on an LLM using TensorRT-LLM and deploy it.

Operation Steps

Creating a Dev Machine for Model Conversion

You can pull the TensorRT-LLM image provided by Tencent Cloud TI-ONE Platform (TI-ONE) and save it to your image repository instance of **Tencent Container Registry (TCR)** Individual Edition or Enterprise Edition:

```
MY_IMAGE="<Your repository address>"  
docker pull tione-public-hub.tencentcloudcr.com/qcloud-ti-  
platform/tritonserver:23.10-py3-trtllm-0.7.1  
docker tag tione-public-hub.tencentcloudcr.com/qcloud-ti-  
platform/tritonserver:23.10-py3-trtllm-0.7.1 _${MY_IMAGE}  
docker push _${MY_IMAGE}
```

Using the custom image above, launch a dev machine and mount the requested Cloud File Storage (CFS) or Data Accelerator Goose FileSystem (GooseFS) storage, as shown below. This dev machine requires 1 GPU card for inference to build TensorRT engine files.

Resource

GPU Type: Select a gpu type

GPU: 1 card(s)
To use GPU, set the number of cards to a number between 0.1-1 or an integer multiple of 1 according to the gpu type. The number of gpus cannot be 0 if the runtime environment is

CPU: 8.0 core

Memory: 80.00 G

System Disk: CBS, Temporary Storage

Storage Path Settings

Ensure that the selected storage instance of CFS, GooseFS or GooseFSx type is interconnected with the resource group's node network. Local data diskand GooseFSx supports mounting only one instance.
The first configured container mount path will serve as the default working directory.

Storage Type	CFS system	Source path	Container Mounting Path	Operation
CFS		/triton-example	/home/tione/notebook	Delete

+ Add

Building a TensorRT-LLM Model

After the dev machine is accessed, the image has built-in sample code for model conversion in the `/workspace/TensorRT-LLM-examples` directory. You can follow the examples to proceed:

1. Download the Baichuan2-13B-Chat model.

You can download the model yourself and save it to the CFS path. Below is a reference method:

```
apt update && apt install git-lfs
git lfs install
GIT_LFS_SKIP_SMUDGE=1 git clone https://www.modelscope.cn/baichuan-
inc/Baichuan2-13B-Chat.git
cd Baichuan2-13B-Chat
git lfs pull
```

2. Follow the comments to modify the content of the `build_triton_repo_baichuan2_13b.sh` file, and then run the script:

```
#!/bin/bash
set -ex
# Specify the model parallelism number.
TP=1
# Modify it. Specify the local directory of the original Hugging Face
model.
HF_MODEL=/home/tione/notebook/triton-example/hf_model/Baichuan2-13B-Chat
# Modify it. Specify the output directory of the Triton model package.
It is recommended to create a directory in the CFS instance.
TRITON_REPO=/home/tione/notebook/triton-example/triton_model/Baichuan2-
13B-Chat/trt- $\{TP\}$ -gpu
# Specify the path to the TensorRT-LLM Engine build script.
BUILD_SCRIPT=tensorrtllm_backend/tensorrt_llm/examples/baichuan/build.py
```

```
# Create an output directory.
mkdir -p ${TRITON_REPO}
cp -r tensorrtllm_backend/all_models/inflight_batcher_llm/*
${TRITON_REPO}/
# Copy Tokenizer relevant files to the output directory.
cp ${HF_MODEL}/*token* ${MODEL_PATH}/tensorrt_llm/1/

# Build the TensorRT-LLM Engine file. For details of the parameters, see
tensorrt_llm/examples/baichuan/README.md.
# Example 1: Baichuan V2 13B parameter scale model, using FP16 and
enabling in-flight batching.
#python3 $BUILD_SCRIPT --model_version v2_13b \
#           --model_dir ${HF_MODEL} \
#           --output_dir ${TRITON_REPO}/tensorrt_llm/1/ \
#           --world_size ${TP} \
#           --max_batch_size 32 \
#           --dtype float16 \
#           --use_gemm_plugin float16 \
#           --use_gpt_attention_plugin float16 \
#           --remove_input_padding \
#           --paged_kv_cache

# Example 2: Baichuan V2 13B parameter scale model, using INT8 weight-
only quantization and enabling in-flight batching.
python3 $BUILD_SCRIPT --model_version v2_13b \
           --model_dir ${HF_MODEL} \
           --output_dir ${TRITON_REPO}/tensorrt_llm/1/ \
           --world_size ${TP} \
           --max_batch_size 32 \
           --dtype float16 \
           --use_weight_only \
           --use_gemm_plugin float16 \
           --use_gpt_attention_plugin float16 \
           --remove_input_padding \
           --paged_kv_cache

# Modify the Triton config.pbtxt configuration file.
# The options.txt file can be modified as needed, but it is generally
recommended to use the default values.
```

```
OPTIONS=options.txt

python3 tensorrtllm_backend/tools/fill_template.py -i
${TRITON_REPO}/preprocessing/config.pbtxt ${OPTIONS}
python3 tensorrtllm_backend/tools/fill_template.py -i
${TRITON_REPO}/postprocessing/config.pbtxt ${OPTIONS}
python3 tensorrtllm_backend/tools/fill_template.py -i
${TRITON_REPO}/tensorrt_llm_bls/config.pbtxt ${OPTIONS}
python3 tensorrtllm_backend/tools/fill_template.py -i
${TRITON_REPO}/ensemble/config.pbtxt ${OPTIONS}
python3 tensorrtllm_backend/tools/fill_template.py -i
${TRITON_REPO}/tensorrt_llm/config.pbtxt ${OPTIONS}

# Create a symbolic link to /data/model. In TI-ONE Online Services,
models are mounted to this path by default.
mkdir -p /data
ln -s ${TRITON_REPO} /data/model

# Start the Triton inference service locally for debugging.
# launch_triton_server
```

The directory structure of the converted model is as follows:

```
# tree
.
├── ensemble
│   ├── 1
│   └── config.pbtxt
├── postprocessing
│   ├── 1
│   │   └── model.py
│   └── config.pbtxt
├── preprocessing
│   ├── 1
│   │   └── model.py
│   └── config.pbtxt
├── tensorrt_llm
│   ├── 1
│   │   ├── baichuan_float16_tp1_rank0.engine
│   │   ├── config.json
│   │   └── model.cache
```

```

| | |— special_tokens_map.json
| | |— tokenization_baichuan.py
| | |— tokenizer_config.json
| | |— tokenizer.model
| | |— config.pbtxt
| | |— tensorrt_llm_bls
| | |— 1
| | |— model.py
| | |— config.pbtxt

```

You can directly run the `launch_triton_server` command on the dev machine to start the Triton Inference Server and refer to `api_test.sh` for local calls. If you want to release a formal inference service that can be called from the public network or within the Virtual Private Cloud (VPC) network, see the section below.

Creating an Online Service

When a service is created, set the **model source** to CFS or GooseFS, and **select a model** from the path on the CFS or GooseFS instance where the converted Triton model package is stored. For the running environment, select the previous custom image or the built-in image **Built-in/TRION(1.0.0)/23.10-py3-trtllm-0.7.1**.

Select computing resources in accordance with actual resource availability, with at least 8 CPU cores, 40 GB of memory, and preferably, an A100 or A800 GPU.

Logs similar to the following indicate that the service has started up successfully:

```

[2024-01-31 16:51:52] 1031 00:51:51.374852 100 tritonserver.cc:2458]
[2024-01-31 16:51:52] +-----+-----+
[2024-01-31 16:51:52] | Option | Value |
[2024-01-31 16:51:52] +-----+-----+
[2024-01-31 16:51:52] | server_id | triton |
[2024-01-31 16:51:52] | server_version | 2.39.0 |
[2024-01-31 16:51:52] | server_extensions | classification sequence model_repository model_repository(unload_dependencies) schedule_policy model_configuration system_shared |
[2024-01-31 16:51:52] | model_repository_path[0] | /data/model |
[2024-01-31 16:51:52] | model_control_mode | MODE_NONE |
[2024-01-31 16:51:52] | strict_model_config | 1 |
[2024-01-31 16:51:52] | rate_limit | 0 |
[2024-01-31 16:51:52] | pinned_memory_pool_byte_size | 268435456 |
[2024-01-31 16:51:52] | cuda_memory_pool_byte_size[0] | 67188864 |
[2024-01-31 16:51:52] | min_supported_compute_capability | 6.0 |
[2024-01-31 16:51:52] | strict_readiness | 1 |
[2024-01-31 16:51:52] | exit_timeout | 30 |
[2024-01-31 16:51:52] | cache_enabled | 0 |
[2024-01-31 16:51:52] +-----+-----+
[2024-01-31 16:51:52] 1031 00:51:51.376420 100 grpc_server.cc:2513] Started GRPCInferenceService at 0.0.0.0:8001
[2024-01-31 16:51:52] 1031 00:51:51.376618 100 http_server.cc:4497] Started HTTPService at 0.0.0.0:8501
[2024-01-31 16:51:52] 1031 00:51:51.418100 100 http_server.cc:270] Started Metrics Service at 0.0.0.0:8002

```

Calling an API

For the text generation API, see Triton's [documentation](#), with an example as follows:

```

# The public access address can be obtained from the Service Call tab on
the frontend page of the online service instance.
SERVER_URL=https://service-*****.sh.tencentapigw.com:443/tione

# Non-streaming request.
curl -X POST ${SERVER_URL}/v2/models/tensorrt_llm_bls/generate -d
'{"text_input": "<reserved_106>Who are you?<reserved_107>", "max_tokens":

```


Use Instructions for Built-in LLM Inference Images

Last updated: 2026-03-12 10:12:19

Inference Image Introduction

Tencent Cloud TI-ONE Platform (TI-ONE) features a built-in general-purpose inference image for LLMs, supports running inference on Hugging Face-format LLMs, provides HTTP services compatible with OpenAI API-format text chat, and incorporates Tencent's self-developed LLM inference acceleration capabilities.

Currently, two LLM inference images are preset:

- **[Recommended] Angel-vLLM:** A version optimized by the TI-ONE team based on the open-source vLLM, supporting various quantization acceleration methods, such as online INT8, NF4, and FP8 weight-only quantization and LayerwiseSearchSMQ. This image is recommended for production environments with high concurrency requirements.
- **Angel-DeepSpeed:** A version optimized by the TI-ONE team based on open-source DeepSpeed, supporting multiple optimization techniques such as operators, communication, quantization, and so on.

This document will introduce how to use the TI-ONE built-in LLM inference image to start an inference service that provides an LLM chat API.

Model Directory Preparations

First, you need to place the model for inference in your own Cloud File Storage (CFS) or Data Accelerator Goose FileSystem (GooseFS) instances. The model directory structure should be the same as that of Text Generation models on Hugging Face. The model format supports `.bin` and `.safetensors` formats.

Note:

The Angel-vLLM image also supports inference for models quantized using the LayerwiseSearchSMQ method via the `tilearn-llm` library. Confirm that the model directory contains the `smoothq_model-8bit-auto.safetensors` file, and it is recommended to delete the original non-quantized model files.

If you want to use the LayerwiseSearchSMQ quantization acceleration technology provided by TI-ONE, submit a ticket for feedback.

Model Configuration Files

To help the inference image better identify the model used for inference, it is recommended to save an additional `ti_model_config.json` configuration file in the model directory, with the format as follows:

```
{"model_id": "Baichuan2-13B-Chat", "conv_template_name": "baichuan2-chat"}
```

Field description:

model_id

- **Meaning:** model name.
- **Recommended value:** If you want to run an open-source model from Hugging Face, it is recommended to directly specify the model ID on Hugging Face, for example, Baichuan2-13B-Chat or Qwen1.5-14B-Chat. The inclusion of a forward slash (/) in the model name or the case of the text does not affect actual usage.
- **Alternative specification method:** If you do not want to add a `ti_model_config.json` file to the model directory, you can also specify the model name by setting the **MODEL_ID** environment variable when starting up the service.

conv_template_name

- **Meaning:** chat template name.
- **Recommended value:** If your `model_id` is the same as the open-source model, and you have not modified the model's chat template during the supervised fine-tuning (SFT) stage, you do not need to add the `conv_template_name` parameter. The inference image will attempt automatic matching. If you know which chat template the model uses, it is recommended to specify it manually.
- **Alternative specification method:** If you do not want to add a `ti_model_config.json` file to the model directory, you can also specify the chat template by setting the **CONV_TEMPLATE** environment variable when starting up the service.

The currently commonly used chat template names are listed in the table below. For the full list of chat templates currently supported by the image, see [conversation.py](#).

Chat Template Name	Supported Model Series
generate	Non-chat models (direct generation, without chat template)
shennong_chat	Tencent industry LLM (an allowlist needs to be

	activated for industry LLMs)
llama-3	llama-3-8b-instruct and llama-3-70b-instruct models
llama-2	llama-2-chat series models
qwen-7b-chat	qwen-chat series models (chatml format)
baichuan2-chat	baichuan2-chat series models
baichuan-chat	baichuan-13b-chat model
chatglm3	chatglm3-6b model
chatglm2	chatglm2-6b model

Starting the Inference Service

You can choose **Model Services > Online Services > New Service** on TI-ONE to start the inference service. The following is the guidance for service instance configuration.

Model Sources and Running Environments

For the model source of the LLM inference service, select CFS or GooseFS. For the path, enter the CFS or GooseFS path that you prepared for placing the model in the [Model Directory Preparations](#) section.

For the running environment, select **Built-in/LLM/Angel-vLLM** or **Built-in/LLM/Angel-DeepSpeed**. It is recommended to select Angel-vLLM.

Resource Application Recommendations

The CVM instance resources required for LLM inference are related to the model parameter scale. We recommend configuring inference service resources according to the following rules. (CPU and memory should be allocated proportionally based on the actual available resources of the node and the number of occupied GPU cards. It is recommended that the allocated memory be larger than the GPU memory to avoid OOMKilled exceptions.)

Model Parameter Scale	GPU Card Type and Quantity
6 ~ 8B	PNV5b * 1 / A10 * 1 / A100 * 1 / V100 * 1
12 ~ 14B	PNV5b * 1 / A10 * 2 / A100 * 1 / V100 * 2
65 ~ 72B	PNV5b * 8 / A100 * 8

Note:

Certain models support long context, such as 32K and 128K, which require significant GPU memory for KV cache. If you do not need such a long context during inference, you can manually reduce the context length.

Note:

If the service fails to start due to insufficient GPU memory, try the following solutions:

1. Allocate more GPU cards to the service or switch to an instance type with larger GPU memory.
2. **[Recommended]** Try enabling INT8 weight-only quantization, which can halve the GPU memory required by model weights, such as `QUANTIZATION=ifq`.
3. **[Recommended]** Try reducing the context length, which can decrease the GPU memory required by KV cache, such as `MAX_MODEL_LEN=4096`.
4. Try adjusting the GPU memory reserved allocation ratio, such as `GPU_MEMORY_UTILIZATION=0.95`.
5. Try using eager mode by adding the startup parameter `--enforce-eager`.

For specific parameter definitions, see [Advanced Settings of the Angel-vLLM Image](#).

Advanced Settings of the Angel-vLLM Image

In most cases, you can directly start the image without any modifications. If you have further advanced settings requirements, you can adjust the container's startup parameters via the **startup command** or **environment variables**.

The default startup command of the image:

```
run
```

The currently supported startup parameters and environment variables are as follows (the startup parameters and environment variables in the same row are equivalent in function; if both are set, the parameters of the startup command have higher priority):

Startup Parameter	Environment Variable	Default Value	Meaning
None	MODEL_ID	None	Model name. For details, see the description of model_id in Model Configuration Files .

None	CONV_TEMPLATE	None	Chat template name. For details, see the description of conv_template_name in Model Configuration Files .
--model-path	MODEL_PATH	/data/model	Model loading path. The CFS path specified when starting the service will be automatically mounted to the /data/model path in the container by default. You usually do not need to modify this.
--num-gpus	TP	Number of GPUs allocated to the container	<p>Number of GPUs for model parallel inference, which is used when a single GPU's memory is insufficient and multiple GPUs are needed to load the model. This defaults to the number of GPUs allocated to a single container.</p> <div style="border: 1px solid #add8e6; padding: 10px; margin-top: 10px;"> <p>Note: The Angel-vLLM(2.0) image supports distributed inference. When multi-machine distributed deployment is selected, this will be automatically set to the total number of GPU cards in the distributed inference cluster.</p> </div>
--worker-num	WORKER_NUM	1	Number of inference workers, which affects the number of processes running inference services in a single container. The default value is 1. Generally, the number of GPU cards allocated to the container equals TP x WORKER_NUM.
--limit-worker-concurrency	MAX_CONCURRENCY	128	Maximum concurrency supported by each inference worker. When the number of requests being processed exceeds this value, new requests will be queued.
--dtype	DTYPE	float16	Model precision type. Valid values: auto, float16, bfloat16, and float32. If it is set to auto, the precision type will be automatically set to the one used during model training.

<code>--seed</code>	SEED	None	Random seed for each generation, not specified by default.
<code>--max-model-len</code>	MAX_MODEL_LEN	Model context length	The maximum number of context tokens supported by the inference service. The context length in the model's configuration is read by default. If a long-context model encounters insufficient GPU memory when loaded with the default configuration, you can manually set this parameter to reduce the value.
<code>--quantization</code>	QUANTIZATION	none	<p>Quantization acceleration mode. Valid values: "none", "ifq", "ifq_nf4", "fp8", "smoothquant", and "auto".</p> <ul style="list-style-type: none"> "none": indicates that quantization acceleration is disabled. "ifq": indicates that online Int8 Weight-Only quantization is enabled, which can accelerate inference with basically no loss in performance and reduce the GPU memory occupied by model weights. "ifq_nf4": indicates that online NF4 Weight-Only quantization is enabled. 4-bit quantization can further accelerate inference and reduce the GPU memory occupied by model weights. This setting is supported starting from the Angel-vLLM(2.0) image. "fp8": indicates that online FP8 Weight-Only quantization is enabled, which can further accelerate inference and reduce the GPU memory occupied by model weights. This setting is supported starting from the Angel-vLLM(2.0) image and is only supported on NVIDIA Hopper series instance types. "smoothquant": indicates that LayerwiseSearchSMQ quantization is enabled, which can further accelerate inference with slightly degraded performance (requiring pre-prepared quantized model files, currently only supported for certain models). "auto": indicates that the quantization mode is automatically determined:

			<ul style="list-style-type: none"> ○ If the GPU of the instance type does not support quantization, quantization is automatically disabled. ○ If the model directory contains the <code>smoothq_model-8bit-auto.safetensors</code> file, LayerwiseSearchSMQ quantization acceleration is automatically enabled. ○ In other cases, online Int8 Weight-Only quantization acceleration (ifq) is enabled by default. <p>If you have high requirements for inference speed or limited GPU memory resources, we recommend that you enable quantization acceleration.</p>
<code>--use-lookahead</code>	USE_LOOKAHEAD	0	<p>Lookahead parallel decoding acceleration. Valid values: 0, 1, true, and false. Setting it to 1 or true enables Lookahead parallel decoding acceleration. Lookahead parallel decoding is effective for accelerating generation in scenarios with long inputs, particularly in retrieval-augmented generation (RAG) scenarios where generated content is included in the input text.</p> <p>This setting is supported starting from the Angel-vLLM(2.0) image.</p>
<code>--num-speculative-tokens</code>	NUM_SPECULATIVE_TOKENS	8	<p>Number of tokens for parallel decoding. It takes effect when parallel decoding is enabled. It indicates the number of tokens decoded at a time during parallel decoding. For single concurrency, it is recommended to set the value to 12. When BatchSize is large, you can appropriately reduce the value, for example, to 6 or 4.</p> <p>This setting is supported starting from the Angel-vLLM(2.0) image.</p>
<code>--api-keys</code>	OPENAI_API_KEYS	None	<p>API_KEY list. This value is not set by default. The supported format is a comma-separated string, such as <code>sk-xxxx,sk-yyyy,sk-zzzz</code>. Once the value is set, Bearer Token authentication is enabled for the</p>

			<p>service's API. You need to include a valid <code>API_KEY</code> as an access token in requests. For example, use <code>-H "Authorization: Bearer sk-xxxx"</code> in a curl request, or set <code>OPENAI_API_KEY</code> when using the OpenAI SDK. After authentication is enabled, requests that fail authentication will trigger an HTTP 401 error.</p> <p>Note: After <code>API_KEY</code> authentication is enabled, online experience is not supported. Only service calls are supported.</p>
<code>--max-num-batched-tokens</code>	None	max(model context length, 2048)	Maximum number of tokens processed per iteration. The default value is the greater of 2048 and the model's supported context length. If GPU memory resources are sufficient and input texts are long, you can increase this value to optimize throughput performance.
<code>--max-num-seqs</code>	None	256	Maximum number of generated sequences processed per iteration.
<code>--gpu-memory-utilization</code>	<code>GPU_MEMORY_UTILIZATION</code>	0.9	The proportion of GPU memory reserved for model weights, activations, and KV cache. A larger value allows for a larger KV cache but increases the risk of GPU out-of-memory.
<code>--enforce-eager</code>	<code>ENFORCE_EAGER</code>	1	Whether to forcibly enable PyTorch's eager mode. By default, it is disabled. In this case, CUDA graphs are used for additional acceleration, which occupies extra GPU memory and increases service startup time. If you encounter GPU out-of-memory errors during startup, you can add the <code>--enforce-eager</code> parameter to the startup command to save GPU memory, though this may slightly decrease inference performance. This setting is enabled by default in the Angel-vLLM(2.0) image.

In addition, the image also supports the startup mode of vLLM's native entrypoint inference service. You can modify the startup command to begin with `python3 -m vllm.entrypoints.openai.api_server`, which supports all parameters natively available in the open-source vLLM 0.4.2. For details, see [vLLM Official Documentation](#).

The platform image version further supports the following parameters:

- `--use-lookahead` : Enables lookahead parallel decoding. This is particularly effective for accelerating generation in scenarios with long inputs, particularly in RAG scenarios where generated content is included in the input text. This parameter requires `--use-v2-block-manager` to be enabled simultaneously.
- `--num-speculative-tokens` : Specifies the number of tokens decoded at a time during parallel decoding. For single concurrency, it is recommended to set the value to 12. When BatchSize is large, you can appropriately reduce the value, for example, to 6 or 4.
- `--quantization` : Additionally supports the `ifq` and `smoothquant` quantization methods. For details, see the quantization acceleration mode description above.

Example:

```
python3 -m vllm.entrypoints.openai.api_server --model /data/model --
served-model-name model --trust-remote-code --quantization ifq --use-v2-
block-manager --use-lookahead --num-speculative-tokens 6
```

Advanced Settings for the Angel-DeepSpeed Image

In most cases, you can directly start the image without any modifications. If you have further advanced settings requirements, you can adjust the container's startup parameters via the **startup command** or **environment variables**.

The default startup command of the image:

```
run
```

The currently supported startup parameters and environment variables are as follows (the startup parameters and environment variables in the same row are equivalent in function; if both are set, the parameters of the startup command have higher priority):

Startup Parameter	Environment Variable	Default Value	Meaning
None	MODEL_ID	None	Model name. For details, see the description of model_id in Model Configuration Files .
None	CONV_TEMPLATE	None	Chat template name. For details, see the description of conv_template_name in

			Model Configuration Files .
<code>--model-path</code>	MODEL_PATH	/data/ model	Model loading path. The CFS path specified when starting the service will be automatically mounted to the /data/model path in the container by default. You usually do not need to modify this.
<code>--num-gpus</code>	TP	1	Number of GPUs for model parallel inference. The default value is 1.
<code>--worker-num</code>	WORKER_NUM	1	Number of inference workers, which affects the number of processes running inference services in a single container. The value can be modified when the number of GPU cards allocated to the container is greater than the value of the <code>--num-gpus</code> parameter.
<code>--limit-worker-concurrency</code>	MAX_CONCURRENCY	32	Maximum concurrency supported by each inference worker. When the number of requests being processed exceeds this value, new requests will be queued.
<code>--dtype</code>	DTYPE	float16	Model precision type. Valid values: "float16", "bfloat16", and "float32"; if the value is set to auto, the precision type will be automatically set to the one used during model training.
<code>--seed</code>	SEED	None	Random seed for each generation, not specified by default.
<code>--ds-dtype</code>	DS_DTYPE	float16	Precision type that is used during Angel-Deepspeed acceleration. Valid values: float16, bfloat16, and int8.
<code>--max-batch-size</code>	MAX_BATCH_SIZE	16	Maximum number of requests per batch when dynamic batching is enabled.
<code>--batch-wait-timeout</code>	BATCH_WAIT_TIMEOUT	0.1	Maximum waiting time per batch (in seconds) when dynamic batching is enabled.

Chat API Documentation

After the inference image container starts up, it will listen to port 8501 by default. You can directly enter the container for debugging by choosing **Instance List** > **Enter Container** in Online

Services.

If you need to access the model's chat API externally from the service, the following methods are available:

1. **Online experience:** Directly access the service on the frontend online experience page or by calling the API in [Quick Tryout of Open-Source LLM – Step 3: Experiencing Model Effects Online](#). This method requires content moderation and supports both built-in LLMs and custom LLMs.
2. **Service call:** On the service's Service Call tab, obtain the service's calling address in the public network or Virtual Private Cloud (VPC) network. The inference services of the platform's built-in LLM do not support service calls; To call services, you use the inference services of your own model or the built-in LLM image. You can perform secondary development of your LLM application based on this API. If content moderation is required, you need to integrate it yourself. If you have high-performance requirements for service calls, it is recommended to use the VPC private network call method.

1. API Description

POST `/v1/chat/completions`

The chat API is compatible with OpenAI's Chat API format in most cases.

2. Input Parameters

HTTP Header parameters

Name	Value
Content-Type	application/json

HTTP request body parameters

Parameter Name	Required	Type	Description
messages	Yes	Array of Message	Chat messages arranged in chronological order.
max_tokens	No	Integer	The default value is the context length supported by the model. It indicates the maximum number of tokens the model can generate. This value cannot exceed the context length supported by the model.

stream	No	Boolean	<p>The default value is false. It indicates a non-streaming response; when the value is set to true, it enables a streaming response. If you are sensitive to first-token latency, streaming is recommended for a better experience.</p>
temperature	No	Float	<p>The default value is 0.7, with a value range of [0.0, 2.0]. This parameter indicates the sampling temperature, which controls the level of randomness during sampling from the generation model.</p> <p>Higher values increase output randomness, while lower values make outputs more focused and deterministic. When the value is set to 0, a greedy sampling policy is applied, making the generation results deterministic. This parameter affects the quality and diversity of the model's generated responses.</p>
top_p	No	Float	<p>The default value is 1.0, with a value range of (0.0, 1.0]. It indicates that tokens with a cumulative probability not exceeding the value of top_p are added to the candidate list.</p> <p>It affects the diversity of the output text. Larger values increase the diversity of the generated text. This parameter affects the quality and diversity of the model's generated responses.</p>
top_k	No	Integer	<p>The default value is -1. It indicates that top-k filtering sampling is disabled. When the value is greater than 0, the model first filters the top-k tokens with the highest probabilities and then applies top-p sampling. This parameter affects the quality and diversity of the model's generated responses.</p>
presence_penalty	No	Float	<p>The default value is 0.0, with a value range of [-2.0, 2.0]. It indicates that a penalty exists. When the value is positive, a penalty is applied to a new token based on whether it appears in the current text, increasing the probability of the model introducing new topics. This parameter affects the quality and diversity of the model's generated responses.</p>
frequency_penalty	No	Float	<p>The default value is 0.0, with a value range of [-2.0, 2.0]. It indicates the frequency penalty. When the value is positive, a penalty is applied to the occurrence frequency of new tokens in the current text, reducing the probability of the model repeating the same text.</p>

			This parameter affects the quality and diversity of the model's generated responses.
repetition_penalty	No	Float	The default value is 1.0, with a value range of [1.0, 2.0]. It indicates the repetition penalty coefficient. A value greater than 1 suppresses the repeated generation of tokens and reduces consecutive repetitions. This parameter affects the quality and diversity of the model's generated responses.
stop	No	Array of String	The default value is None. It indicates no additional stop tokens; you can set one or more stop tokens based on actual business needs. This way, generation automatically stops when the stop token is encountered.
ignore_eos	No	Boolean	The default value is false. It indicates that stop tokens are not ignored; when the value is set to true, the model is forced to generate tokens until max_tokens is reached. This is typically used only during debugging. This parameter is supported only by the Angel-vLLM image.
seed	No	Integer	The default value is None. It indicates that no random seed is set; when a specific value is set, this random seed will be used for each generation, which is typically used to reproduce specific generation results. This parameter is supported only by the Angel-vLLM image.

Message

Parameter Name	Required	Type	Description
role	Yes	String	Roles that correspond to chat messages. The following values are currently supported: <ul style="list-style-type: none"> system: indicates a system prompt (only supported once at the start). user: indicates the user. assistant: indicates the chat assistant. Messages must alternate between the user and assistant in a question-and-answer format.
content	Yes	String	Chat message content.

3. Output Parameters

In non-streaming scenarios, `content-type: application/json` data will be returned in the following format:

Parameter Name	Type	Description
id	String	Unique identifier for a chat.
choices	Array of Choice	Generation result.
created	Integer	UNIX timestamp (in seconds) for the start of the chat.
model	String	Name of the model used in the chat.
object	String	Object type, which is fixed as chat.completion.
usage	Usage	Token quantity statistics include: <ul style="list-style-type: none"> completion_tokens: number of tokens included in the generation result. prompt_tokens: number of tokens in the prompt. total_tokens: total number of tokens in the request (prompt and generation result).

In streaming output, data will be returned through SSE with `content-type: text/event-stream; charset=utf-8`

In streaming output, each row of returned data is in the `data: shard` format, and the shard data format is as follows:

Parameter Name	Type	Description
id	String	Unique identifier for a chat.
choices	Array of StreamChoice	Generation result.
created	Integer	UNIX timestamp (in seconds) for the start of the chat.
model	String	Name of the model used in the chat.
object	String	Object type, which is fixed as chat.completion.chunk.

The streaming output ends with `data: [DONE]`. For details, see [Examples](#).

4. Examples

Example 1 Non-streaming Request

Input example

```
curl -H "content-type: application/json"
http://localhost:8501/v1/chat/completions -d '{"messages":[{"role":
"user", "content": "Hello"}], "temperature": 0.0}'
```

Output example

```
{"id":"chatcpl-
4aeRgYwnaYe4RzmmcyKtYs","object":"chat.completion","created":1698291242,"m
odel":"baichuan-13b-chat","choices":[{"index":0,"message":
{"role":"assistant","content":"Hello! How can I assist you
today?},"finish_reason":"stop"}],"usage":
{"prompt_tokens":4,"total_tokens":16,"completion_tokens":12}}
```

Example 2: Streaming Request

Input example

```
curl -H "content-type: application/json"
http://localhost:8501/v1/chat/completions -d '{"messages":[{"role":
"user", "content": "What's up"}], "temperature": 0.0, "stream": true}'
```

Output example

```
data: {"id": "chatcpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"role": "assistant"},
"finish_reason": null}}

data: {"id": "chatcpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "What's"},
"finish_reason": null}}

data: {"id": "chatcpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "up!"},
"finish_reason": null}}
```

```
data: {"id": "chatcmpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "How can I"},
"finish_reason": null}}}]

data: {"id": "chatcmpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "assist"},
"finish_reason": null}}}]

data: {"id": "chatcmpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "you"},
"finish_reason": null}}}]

data: {"id": "chatcmpl-3fYW8fqN3YYMJiebiZgpzZ", "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {"content": "today?"},
"finish_reason": null}}}]

data: {"id": "chatcmpl-3fYW8fqN3YYMJiebiZgpzZ", "object":
"chat.completion.chunk", "created": 1698290857, "model": "baichuan-13b-
chat", "choices": [{"index": 0, "delta": {}, "finish_reason": "stop"}]}

data: [DONE]
```

5. Developer Resources

Python SDK

You can directly use the OpenAI [SDK](#) to call this inference service. Examples are as follows:

```
import os
from openai import OpenAI

# Example 1: local access address of the container.
os.environ["OPENAI_BASE_URL"] = "http://127.0.0.1:8501/v1"
# Example 2: public network access address, which can be obtained from the
service call page. When using the OpenAI SDK, add v1 at the end of the
address.
os.environ["OPENAI_BASE_URL"] = "https://service-
*****.sh.tencentapigw.com:443/tione/v1"
# If authentication is enabled by setting the --api-keys parameter or the
OPENAI_API_KEYS environment variable when the service is started,
```

```
OPENAI_API_KEY needs to be set to an available api_key; otherwise, it can
be set to any value.
```

```
os.environ["OPENAI_API_KEY"] = "EMPTY"

client = OpenAI()

# Non-streaming request example.
print("----- standard request -----")
completion = client.chat.completions.create(
    model="model",
    messages=[
        {
            "role": "user",
            "content": "Hello!"
        },
    ],
    temperature=0.7,
    top_p=1.0,
    max_tokens=128,
)
print(completion.choices[0].message.content)

# Streaming request example.
print("----- streaming request -----")
stream = client.chat.completions.create(
    model="model",
    messages=[
        {
            "role": "user",
            "content": "Hello!"
        },
    ],
    temperature=0.7,
    top_p=1.0,
    max_tokens=128,
    stream=True,
)
for chunk in stream:
    if not chunk.choices or not chunk.choices[0].delta.content:
        continue
    print(chunk.choices[0].delta.content, end="")
```

```
print()
```

Command Line Chat Demo

You can also call the chat API through the requests library that is commonly used in Python. Below is a demo example of a chat interaction between command lines and the LLM inference service:

```
import argparse
import requests
import json

def chat(messages):
    data = {
        "messages": messages,
        "temperature": args.temperature,
        "max_tokens": args.max_tokens,
        "top_p": args.top_p,
        "stream": True, # Enable streaming output.
    }
    header = {
        "Content-Type": "application/json",
    }
    if args.token:
        header["Authorization"] = f"Bearer {args.token}"

    response = requests.post(f"{args.server}/v1/chat/completions",
        json=data, headers=header, stream=True) # Set the stream=True parameter
    to obtain real-time data streams.
    if response.status_code != 200:
        print(response.json())
        exit()

    result = ""
    print("Assistant: ", end = "", flush = True)
    for part in response.iter_lines():
        if part:
            if "content" in part.decode("utf-8"):
```

```
        content = json.loads(part.decode("utf-8")[5:])[0]["choices"]
[0]["delta"]["content"] # Filter the string to remove the data: prefix,
convert the string into JSON format, and then extract the text.
        result += content
        print(content, end = "", flush = True)

print("\n")

return result

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Chat CLI Demo.")
    parser.add_argument("--server", type=str,
default="http://127.0.0.1:8501")
    parser.add_argument("--max-tokens", type=int, default=512)
    parser.add_argument("--temperature", type=float, default=0.7)
    parser.add_argument("--top_p", type=float, default=1.0)
    parser.add_argument("--system", type=str, default=None)
    parser.add_argument("--token", type=str, default=None)
    args = parser.parse_args()

messages = []
if args.system:
    messages.append({"role": "system", "content": args.system})

while True:
    user_input = input("User: ")
    messages.append({"role": "user", "content": user_input})
    response = chat(messages)
    messages.append({"role": "assistant", "content": response})
```

Guide on LLM Inference Resources

Last updated: 2026-03-03 17:44:43

This document aims to introduce the configuration resources required to ensure the normal running of LLMs during LLM inference on Tencent Cloud TI-ONE Platform (TI-ONE). It is provided for your reference only.

Guide on Inference Resources for Built-in LLMs

Note:

1. For the inventory and pricing of each instance type, go to the [Cloud Virtual Machine \(CVM\) console](#) and see [Instance Creation Guide](#). For the PNV6/HCCPNV6 instance types, you need to contact your designated Tencent Cloud sales representative for activation and purchase.
2. When deploying the DeepSeek V3 or R1 model, if you only need a low concurrency experience, you can opt for a single-node deployment. If you have high requirements for inference performance and context length, and you have sufficient computing resources, it is recommended to use a deployment of at least 2 nodes.
3. The **inference resource configuration** in the table below is slightly smaller than the CVM instance configuration because TI-ONE occupies a small amount of resources when managing CVM instances. For example, a CVM instance specification with 128 cores will have 125.6 cores available after the CVM instance is added to the resource group.

Built-in LLM	Model List	Recommended Inference Resource	
		CVM instance source: Select from CVM Instances (yearly/monthly subscription)	CVM instance source: Purchase from the TI-ONE Platform (pay-as-you-go)
Hunyuan-Large	Hunyuan-Large-chat	Deployment mode: standard deployment (with nf4 quantization enabled) <ul style="list-style-type: none"> • [Recommended configuration 1] CVM instance specifications: PNV6.96XLARGE2304	-

		<p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> [Recommended configuration 2] <p>CVM instance specifications: HCCPNV4h.48XLARGE1024</p> <p>CVM instance configuration: 192C, 1024 GB, and 8 A100 GPU cards</p> <p>Inference resource configuration: 189C, 980 GB, and 8 A100 GPU cards</p>	
Hunyuan-A13B	Hunyuan-A13B-Instruct	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: PNV6.16XLARGE640</p> <p>CVM instance configuration: 64C, 640 GB, and 4 GPU cards</p> <p>Inference resource configuration: 60C, 600 GB, and 4 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)</p>
DeepSeek series models	DeepSeek-V3.1-Terminus	<p>[Minimum configuration]</p> <p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

		<p>[Recommended configuration] Deployment mode: Multi-machine Distributed Deployment CVM instance specification: HCCPNV6.96XLARGE2304 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	
DeepSeek-V3.1		<p>[Minimum configuration] Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <p>[Recommended configuration] Deployment mode: Multi-machine Distributed Deployment CVM instance specification: HCCPNV6.96XLARGE2304 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	<p>Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

DeepSeek-R1-0528-AngelACC	<p>Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
DeepSeek-R1-0528-AngelACC-PD	<p>[Minimum configuration] Deployment mode: multi- machine distributed deployment CVM instance specification: HCCPNV6.96XLARGE230 4 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances) Note: This model is deployed using a PD disaggregation architecture, with 2 instances configured as 1p1d, 3 instances as 2p1d, 4 instances as 2p2d, 5 instances as 3p2d, and so on.</p>	-
DeepSeek-R1-0528	<ul style="list-style-type: none"> Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards 	<p>Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

		<p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> • Deployment mode: Multi-machine Distributed Deployment <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	
	DeepSeek-R1-AngelACC	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
	DeepSeek-R1-AngelACC-PD	<p>[Minimum configuration]</p> <p>Deployment mode: multi-machine distributed deployment</p> <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	-

		<p>Note: This model is deployed using a PD disaggregation architecture, with 2 instances configured as 1p1d, 3 instances as 2p1d, 4 instances as 2p2d, 5 instances as 3p2d, and so on.</p>	
	<p>DeepSeek-R1</p>	<ul style="list-style-type: none"> • Deployment mode: standard deployment <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> • Deployment mode: Multi-machine Distributed Deployment <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Deployment mode: Multi-machine Distributed Deployment</p> <p>Number of nodes: 2</p> <p>Inference resource configuration (per node): 380C, 2214 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

<p>DeepSeek-V3-0324-AngelACC</p>	<p>Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
<p>DeepSeek-V3-0324-AngelACC-PD</p>	<p>[Minimum configuration] Deployment mode: multi- machine distributed deployment CVM instance specification: HCCPNV6.96XLARGE230 4 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances) Note: This model is deployed using a PD disaggregation architecture, with 2 instances configured as 1p1d, 3 instances as 2p1d, 4 instances as 2p2d, 5 instances as 3p2d, and so on.</p>	<p>-</p>
<p>DeepSeek-V3-0324</p>	<ul style="list-style-type: none"> Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards 	<p>Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

	<p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> [Recommended configuration] Deployment mode: Multi-machine distributed deployment <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards * (2 instances)</p>	
DeepSeek-V3-AngelACC	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
DeepSeek-V3	<ul style="list-style-type: none"> Deployment mode: standard deployment <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> Deployment mode: Multi-machine 	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>

		<p>Distributed Deployment</p> <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Number of nodes: 2</p> <p>Inference resource configuration (per node): 380C, 2214 GB, and 8 GPU cards</p>	
DeepSeek-Prover-V2-7B	<ul style="list-style-type: none"> • Deployment mode: standard deployment <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> • [Recommended configuration] <p>Deployment mode: Multi-machine distributed deployment</p> <p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>	
DeepSeek-Prover-V2-671B	<ul style="list-style-type: none"> • Deployment mode: standard deployment 	<p>Deployment mode: standard deployment</p>	

		<p>CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p> <ul style="list-style-type: none"> [Recommended configuration] Deployment mode: Multi-machine distributed deployment <p>CVM instance specification: HCCPNV6.96XLARGE2304 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	<p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
	<p>DeepSeek-R1-Distill-Qwen-1.5B</p>	<p>Deployment mode: standard deployment CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card</p>	<p>Deployment mode: standard deployment CVM instance specifications: 12C, 44 GB, and 1 A10 GPU card</p>
	<p>DeepSeek-R1-Distill-Qwen-7B</p>	<p>Deployment mode: standard deployment CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card</p>	<p>Deployment mode: standard deployment CVM instance specifications: 12C, 44 GB, and 1 A10 GPU card</p>

		Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	
DeepSeek-R1-Distill-Qwen-14B	<ul style="list-style-type: none"> [Recommended configuration 1] Deployment mode: standard deployment <p>CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card</p> <ul style="list-style-type: none"> [Recommended configuration 2] Deployment mode: Standard deployment <p>CVM instance specifications: PNV5b.8XLARGE96 CVM instance configuration: 32C, 96 GB, and 1 PNV5b GPU card Inference resource configuration: 30C, 80 GB, and 1 PNV5b GPU card</p>	<ul style="list-style-type: none"> [Recommended configuration 1] Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160) [Recommended configuration 2] Deployment mode: standard deployment CVM instance specifications: 32C, 96 GB, and 1 PNV5b GPU card 	
DeepSeek-R1-Distill-Qwen-32B-AngelACC	<p>Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card</p>	<p>Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)</p>	

	<p>DeepSeek-R1-Distill-Qwen-32B</p>	<ul style="list-style-type: none"> [Recommended configuration 1] Deployment mode: standard deployment <p>CVM instance specification: PNV6.4XLARGE160</p> <p>CVM instance configuration: 16C, 160 GB, and 1 GPU card</p> <p>Inference resource configuration: 15C, 144 GB, and 1 GPU card</p> <ul style="list-style-type: none"> [Recommended configuration 2] Deployment mode: Standard deployment <p>CVM instance specifications: PNV5b.16XLARGE192</p> <p>CVM instance configuration: 64C, 192 GB, and 2 PNV5b GPU cards</p> <p>Inference resource configuration: 62C, 172 GB, and 2 PNV5b GPU cards</p>	<ul style="list-style-type: none"> [Recommended configuration 1] Deployment mode: standard deployment <p>CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)</p> <ul style="list-style-type: none"> [Recommended configuration 2] Deployment mode: standard deployment <p>CVM instance specifications: 64C, 192 GB, and 2 PNV5b GPU cards</p>
<p>gpt-oss</p>	<p>gpt-oss-120b</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.8XLARGE320</p> <p>CVM instance configuration: 32C, 320 GB, and 2 GPU cards</p> <p>Inference resource configuration: 30C, 288 GB, and 2 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 32C, 320 GB, and 2 GPUs (PNV6.8XLARGE320)</p>
	<p>gpt-oss-20b</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specification:</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 16C, 160</p>

		<p>PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card</p>	<p>GB, and 1 GPU (PNV6.4XLARGE160)</p>
Kimi-K2	Kimi-K2- Instruct	<p>Deployment mode: Multi- machine Distributed Deployment CVM instance specification: HCCPNV6.96XLARGE230 4 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	-
	Kimi-K2- Instruct- 0905	<p>Deployment mode: Multi- machine Distributed Deployment CVM instance specification: HCCPNV6.96XLARGE230 4 CVM instance configuration: 384C, 2304 GB, and 8 GPU cards Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	-
Cosmos- Reason1	Cosmos- Reason1	<p>Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card</p>	<p>Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)</p>

		Inference resource configuration: 15C, 144 GB, and 1 GPU card	
Qwen3 series models	Qwen3-0.6B	Deployment mode: standard deployment CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Deployment mode: standard deployment CVM instance specifications: 12C, 44 GB, and 1 A10 GPU card
	Qwen3-1.7B	Deployment mode: standard deployment CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Deployment mode: standard deployment CVM instance specifications: 12C, 44 GB, and 1 A10 GPU card
	Qwen3-4B	Deployment mode: standard deployment CVM instance specification: GNV4.3XLARGE44 CVM instance configuration: 12C, 44 GB, and 1 A10 GPU card Inference resource configuration: 11C, 35 GB, and 1 A10 GPU card	Deployment mode: standard deployment CVM instance specifications: 12C, 44 GB, and 1 A10 GPU card
	Qwen3-8B	Deployment mode: standard deployment CVM instance specifications: PNV6/HCCPNV6 series Inference resource configuration: 15C, 144	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)

	GB, and 1 GPU card	
Qwen3-14B	Deployment mode: standard deployment CVM instance specifications: PNV6/HCCPNV6 series Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
Qwen3-30B-A3B	Deployment mode: standard deployment CVM instance specifications: PNV6/HCCPNV6 series Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
Qwen3-32B	Deployment mode: standard deployment CVM instance specifications: PNV6/HCCPNV6 series Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
Qwen3-235B-A22B	Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards	Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)
Qwen3-235B-A22B-Instruct-2507	Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280	Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)

		GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards	
	Qwen3-235B-A22B-Thinking-2507	Deployment mode: standard deployment CVM instance specification: PNV6.32XLARGE1280 CVM instance configuration: 128C, 1280 GB, and 8 GPU cards Inference resource configuration: 125C, 1207 GB, and 8 GPU cards	Deployment mode: standard deployment CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)
	Qwen3-0.6B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3-1.7B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3-4B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)

		CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	
	Qwen3-8B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3-14B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3-30B-A3B-FP8	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3-32B-FP8	Deployment mode: standard deployment CVM instance specification:	Deployment mode: standard deployment CVM instance specifications: 16C, 160

		PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	GB, and 1 GPU (PNV6.4XLARGE160)
	Qwen3- 235B- A22B-FP8	Deployment mode: standard deployment CVM instance specifications: PNV6.16XLARGE640 CVM instance configuration: 64C, 640 GB, and 4 GPU cards Inference resource configuration: 60C, 600 GB, and 4 GPU cards	Deployment mode: standard deployment CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)
	Qwen3- 235B- A22B- Instruct- 2507-FP8	Deployment mode: standard deployment CVM instance specifications: PNV6.16XLARGE640 CVM instance configuration: 64C, 640 GB, and 4 GPU cards Inference resource configuration: 60C, 600 GB, and 4 GPU cards	Deployment mode: standard deployment CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)
	Qwen3- 235B- A22B- Thinking- 2507-FP8	Deployment mode: standard deployment CVM instance specifications: PNV6.16XLARGE640 CVM instance configuration: 64C, 640 GB, and 4 GPU cards Inference resource configuration: 60C, 600 GB, and 4 GPU cards	Deployment mode: standard deployment CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)
Qwen3- Coder series models	Qwen3- Coder- 480B-	Deployment mode: Multi- machine Distributed Deployment	-

	A35B-Instruct	<p>CVM instance specification: HCCPNV6.96XLARGE2304</p> <p>CVM instance configuration: 384C, 2304 GB, and 8 GPU cards</p> <p>Inference resource configuration: 380C, 2214 GB, and 8 GPU cards (2 instances)</p>	
	Qwen3-Coder-480B-A35B-Instruct-FP8	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
QwQ series models	qwq_32b	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.4XLARGE160</p> <p>CVM instance configuration: 16C, 160 GB, and 1 GPU card</p> <p>Inference resource configuration: 15C, 144 GB, and 1 GPU card</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)</p>
Qwen series models	qwen-14b-base	<p>CVM instance specifications: PNV5b.8XLARGE96</p> <p>CVM instance configuration: 32C, 96 GB, and 1 PNV5b GPU card</p> <p>Inference resource configuration: 30C, 80 GB, and 1 PNV5b GPU card</p> <p>Deployment mode: standard deployment</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 32C, 96 GB, and 1 PNV5b GPU card</p>

	qwen-14b-chat	<p>CVM instance specifications: PNV5b.8XLARGE96</p> <p>CVM instance configuration: 32C, 96 GB, and 1 PNV5b GPU card</p> <p>Inference resource configuration: 30C, 80 GB, and 1 PNV5b GPU card</p> <p>Deployment mode: standard deployment</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 32C, 96 GB, and 1 PNV5b GPU card</p>
Qwen3-VL series models	Qwen3-VL-235B-A22B-Instruct	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.32XLARGE1280</p> <p>CVM instance configuration: 128C, 1280 GB, and 8 GPU cards</p> <p>Inference resource configuration: 125C, 1207 GB, and 8 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 128C, 1280 GB, and 8 GPUs (PNV6.32XLARGE1280)</p>
Qwen2.5-VL series models	Qwen2.5-VL-32B-Instruct	<p>Deployment mode: standard deployment</p> <p>CVM instance specification: PNV6.8XLARGE320</p> <p>CVM instance configuration: 32C, 320 GB, and 2 GPU cards</p> <p>Inference resource configuration: 30C, 288 GB, and 2 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 32C, 320 GB, and 2 GPUs (PNV6.8XLARGE320)</p>
	Qwen2.5-VL-72B-Instruct	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: PNV6.16XLARGE640</p> <p>CVM instance configuration: 64C, 640 GB, and 4 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)</p>

		Inference resource configuration: 60C, 600 GB, and 4 GPU cards	
	Qwen2.5-VL-72B-Instruct-AWQ	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
Qwen3-Embedding series models	Qwen3-Embedding-8B	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
Baichuan2 series models	baichuan2-7b-base	CVM instance specifications: PNV4.7XLARGE116 CVM instance configuration: 28C, 116 GB, and 1 A10 GPU card Deployment mode: standard deployment Inference resource configuration: 24C, 96 GB, and 1 A10 GPU card	Deployment mode: standard deployment CVM instance specifications: 28C, 116 GB, and 1 A10 GPU card
	baichuan2-7b-chat	CVM instance specifications: PNV4.7XLARGE116 CVM instance configuration: 28C, 116 GB, and 1 A10 GPU card	Deployment mode: standard deployment CVM instance specifications: 28C, 116 GB, and 1 A10 GPU card

		Deployment mode: standard deployment Inference resource configuration: 24C, 96 GB, and 1 A10 GPU card	
	baichuan2-13b-base	CVM instance specifications: PNV5b.8XLARGE96 CVM instance configuration: 32C, 96 GB, and 1 PNV5b GPU card Inference resource configuration: 30C, 80 GB, and 1 PNV5b GPU card Deployment mode: standard deployment	Deployment mode: standard deployment CVM instance specifications: 32C, 96 GB, and 1 PNV5b GPU card
	baichuan2-13b-chat	CVM instance specifications: PNV5b.8XLARGE96 CVM instance configuration: 32C, 96 GB, and 1 PNV5b GPU card Inference resource configuration: 30C, 80 GB, and 1 PNV5b GPU card Deployment mode: standard deployment	Deployment mode: standard deployment CVM instance specifications: 32C, 96 GB, and 1 PNV5b GPU card
Gemma 3 series models	gemma-3-27b-it	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160 CVM instance configuration: 16C, 160 GB, and 1 GPU card Inference resource configuration: 15C, 144 GB, and 1 GPU card	Deployment mode: standard deployment CVM instance specifications: 16C, 160 GB, and 1 GPU (PNV6.4XLARGE160)
	gemma-3-12b-it	Deployment mode: standard deployment CVM instance specification: PNV6.4XLARGE160	Deployment mode: standard deployment CVM instance specifications: 16C, 160

		<p>CVM instance configuration: 16C, 160 GB, and 1 GPU card</p> <p>Inference resource configuration: 15C, 144 GB, and 1 GPU card</p>	<p>GB, and 1 GPU (PNV6.4XLARGE160)</p>
GLM-4.5V series models	GLM-4.5V	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: PNV6.16XLARGE640</p> <p>CVM instance configuration: 64C, 640 GB, and 4 GPU cards</p> <p>Inference resource configuration: 60C, 600 GB, and 4 GPU cards</p>	<p>Deployment mode: standard deployment</p> <p>CVM instance specifications: 64C, 640 GB, and 4 GPUs (PNV6.16XLARGE640)</p>

Guide on LLM Inference Parameters

Last updated: 2026-01-15 13:37:47

Module Series	Model List	Inference Image	General Environment Variable for Inference	Non-General Environment Variable for Inference
Hunyuan series models	Hunyuan- Large- Instruct	angel- vllm- 0.6.2- ti- 2.1.6- dev202 50627	<ul style="list-style-type: none"> • QUANTIZATION: quantization method, which is not specified by default. • MAX_MODEL_LEN: context length, which is automatically populated. • GPU_MEMORY_UTILIZATION: proportion of GPU memory reserved. The default value is 0.9. 	<ul style="list-style-type: none"> • USE_LOOKAHEAD: The default value is 0. A value of 1 indicates that Lookahead parallel decoding is enabled, which can significantly accelerate decoding in many scenarios. • NUM_SPECULATIVE_TOKENS: The default value is 6. This parameter defines the number of tokens decoded at a time during Lookahead parallel decoding. If the required concurrency is high, this value can be decreased. If the concurrency is low, it can be increased. • ENFORCE_EAGER: whether to forcibly enable PyTorch's eager mode. The default value is false, in which case CUDA Graph is additionally used for further acceleration, but this consumes extra GPU memory and increases service startup time. • DTYPE: The default value is float16. If you

				<p>want to perform inference with bfloat16 precision, manually change the value to bfloat16.</p> <ul style="list-style-type: none"> • ENABLE_PREFIX_CACHING: --enable-prefix-caching. The default value is true.
Hunyuan A13B series models	Hunyuan-A13B-Instruct	vllm-0.10.0-ti-20250827	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. The default value is 32768. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 256. • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.9. • SERVED_MODEL_NAME: model name. The default value is the service ID. 	-

			<ul style="list-style-type: none"> • QUANTIZATION: quantization method, which is not configured by default. 	
DeepSeek series models	<p>DeepSeek-V3.1-Terminus</p> <p>DeepSeek-V3.1-Terminus</p>	angel-vllm-angelacc-v19-ds31-dev20250821	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. The default value is 32768. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 32. • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.85 for tp16 and 0.95 for tp8. • SERVED_MODEL_NAME: model name. The default value is the service ID. 	<ul style="list-style-type: none"> • NUM_MTP_TOKENS: number of tokens that the MTP speculative sampling mechanism attempts to predict. The default value is 1. Setting this parameter to 0 disables MTP, while setting it to 2 allows the model to predict 2 tokens at a time, which may lead to a degradation in output quality. • VLLM_MTP_REJECT_SAMP_METHOD: policy for MTP speculative sampling. The default value is strict, which indicates the strict mode. If improved speed is required, the value can be changed to prob, which enhances performance but may result in less consistent output alignment. • GPU_MEMORY_UTILIZATION: The default value is 0.95 for stand-alone deployment and 0.85 for distributed deployment. • ENABLE_THINKING: whether to enable the thinking model. The default value is 0, which indicates that the model is disabled. Changing the

			<ul style="list-style-type: none"> • DISABLE_LOG_REQUESTS: whether to disable log details. The default value is 1, which indicates that detailed prompts for each request are not printed. If log printing is required, change the value to 0. 	<p>value to 1 will enable the model.</p>
	<p>DeepSeek-R1-0528-AngelaCC</p>	<p>angel-vllm-angelacc-v18-dev20250710</p>	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. The default value is 65536. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 128. • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.85. 	<ul style="list-style-type: none"> • SEPARATE_REASONING: whether to return the chain of thought separately in the reasoning_content field. The default value is 1. • VLLM_MTP_REJECT_SAMPLE_METHOD: policy for MTP speculative sampling. The default value is strict, which indicates the strict mode. If improved speed is required, the value can be changed to prob, which enhances performance but may result in less consistent output alignment. • VLLM_CPU_GPU_OVERLAP: whether to enable CPU-GPU overlap scheduling for acceleration. The default value is 1, which indicates that this feature is disabled. When enabled, this feature is incompatible
	<p>DeepSeek-R1-AngelaCC</p>			

<p>DeepSeek-V3-0324-AngelACC</p>		<ul style="list-style-type: none"> • SERVED_MODEL_NAME: model name. The default value is the service ID. • DISABLE_LOG_REQUESTS: whether to disable log details. The default value is 1, which indicates that detailed prompts for each request are not printed. If log printing is required, change the value to 0. 	<p>with constrained decoding (such as enforcing JSON output format), with a slight performance improvement.</p> <ul style="list-style-type: none"> • TOOL_CHAT_VERSION: This parameter can be set to v2 to update the tool calling chat template to the v2 version. The default version of the tool calling chat template may not perform well. In this case, enabling this setting can improve the success rate of tool calls in certain scenarios. By default, the v2 version of the chat template is not used.
<p>DeepSeek-R1-0528-AngelACC-PD</p>	<p>angel-vllm-angelacc-pd-v19-dev20250812</p>		<ul style="list-style-type: none"> • VLLM_PREFILL_MAX_NUM_SEQS: maximum number of concurrent sequences for the prefill node. The default value is 8. • VLLM_DECODE_MAX_NUM_SEQS: maximum number of concurrent sequences for the decode node. The default value is 32. • VLLM_PREFILL_GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved for the prefill node. The default value is 0.85. • VLLM_DECODE_GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved for the decode node. The default value is 0.85.

DeepSeek-R1-AngelA-CC-PD

- **VLLM_PREFILL_MAX_MODEL_LEN:** maximum context length of the model on the prefill node. The default value is 32768.
- **VLLM_DECODE_MAX_MODEL_LEN:** maximum context length of the model on the decode node. The default value is 32768.
- **VLLM_PREFILL_MAX_PROMPT_LEN:** maximum input length of the model on the prefill node. The default value is 16384.
- **VLLM_DECODE_MAX_PROMPT_LEN:** maximum input length of the model on the decode node. The default value is 16384.
- **VLLM_PREFILL_MAX_NUM_BATCHED_TOKENS:** maximum number of tokens that can be processed in a single batch on the prefill node. The default value is 32768.
- **VLLM_DECODE_MAX_NUM_BATCHED_TOKENS:** maximum number of tokens that can be processed in a single batch on the decode node. The default value is 64.
- **VLLM_NUM_PREFILLS:** number of prefill nodes. By default, it is set to half the total number of nodes, rounded up.
- **VLLM_NUM_DECODES:** number of decode

DeepSe
ek-V3-
0324-
Angela
CC-PD

nodes. By default, it is set to half the total number of nodes, rounded down.

- **TOOL_CHAT_VERSION:** This parameter can be set to v2 to update the tool calling chat template to the v2 version. The default version of the tool calling chat template may not perform well. In this case, enabling this setting can improve the success rate of tool calls in certain scenarios. By default, the v2 version of the chat template is not used.
- **EXTRA_PARAMS:** extra parameters for service startup, which will be passed as-is to the vllm startup command.
- **SEPARATE_REASONING:** whether to return the chain of thought separately in the reasoning_content field. The default value is 1.
- **VLLM_CPU_GPU_OVERLAP:** whether to enable CPU-GPU overlap scheduling for acceleration. The default value is 0, which indicates that this feature is disabled. When enabled, this feature is incompatible with constrained decoding (such as enforcing JSON output format), with a slight performance improvement.

DeepSeek-V3-AngelA-CC

sglang-v0.4.4-mtp-dev20250327

- **MAX_MODEL_LEN:** context length. The default value is 65536.
- **MAX_RUNNING_REQUESTS:** maximum number of concurrent requests. The default value is 64.
- **MAX_CUDA_GRAPH_BS:** upper limit for CUDA graph size. The default value is 64.
- **CHUNKED_PREFILL_SIZE:** --chunked-prefill-size. The default value is 24000.
- **MEM_FRACTION_STATIC:** --mem-fraction-static. The default value is 0.7.
- **SEPARATE_REASONING:** whether to return the chain of thought via reasoning_content. The default value is 0.
- **ENABLE_MTP:** whether to
- **ENABLE_DP:** whether to enable DP acceleration. The default value is 0.
- **DP_SIZE:** the --dp-size parameter used when DP acceleration is enabled. The default value is the number of GPU cards.

enable MTP acceleration for parallel decoding. The default value is 1.

- **AUTO_TEMPLATE:** whether to automatically set the chat template, tool parser, and reasoning parser. The default value is 1.

DeepSeek-R1-0528

sglang-v0.4.6.post4-dev20250716

DeepSeek-V3-0324

DeepSeek-V3

DeepSeek-R1

- **MAX_MODEL_LEN:** context length, which is automatically populated. For full-featured DeepSeek, the default value is 64k.
- **MAX_RUNNING_REQUESTS:** maximum number of concurrent requests, which is automatically populated. For full-featured DeepSeek, the value is adjusted.
- **MAX_CUDA_GRAPH_BS:** upper limit for

- **ENABLE_EP_MOE:** whether to enable EP MOE acceleration. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek.
- **QUANTIZATION:** quantization method, which is not configured by default.
- **DP_SIZE:** the --dp-size parameter used when DP acceleration is enabled, which is not configured by default.
- **ENABLE_MULTIMODAL:** whether to enable multimodal support. By default, this parameter is not set.

DeepSeek-Prover-V2-671B

DeepSeek-Prover-V2-7B

DeepSeek-R1-Distill-Qwen-1.5B

DeepSeek-R1-Distill-Qwen-7B

CUDA graph size, which is automatically populated. For full-featured DeepSeek, the value is adjusted.

- **CHUNKED_PREFILL_SIZE:** -chunked-prefill-size. It is automatically populated.
- **MEM_FRACTION_STATIC:** -mem-fraction-static. It is automatically populated. For full-featured DeepSeek, the value is adjusted.
- **ENABLE_MTP** : whether to enable MTP acceleration for parallel decoding. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek with 16 GPU cards.
- **SEPARATE_REASONING:**

<p>DeepSeek-R1-Distill-Qwen-14B</p>		<p>whether to return the chain of thought via reasoning_content. The default value is 0.</p> <ul style="list-style-type: none"> • AUTO_TEMPLATE: whether to automatically set the chat template, the tool calling template, and the chain-of-thought parsing template (DeepSeek, Qwen3, and QwQ). The default value is 1. 	
<p>DeepSeek-R1-Distill-Qwen-32B</p>			
<p>DeepSeek-R1-Distill-Qwen-32B-AngelA CC</p>	<p>hunyuan-H-5.1.1-v1-trtllm-ti-20250814</p>	<p>-</p>	<ul style="list-style-type: none"> • SERVER_TIMEOUT: timeout period for service startup. The default value is 1800 seconds. • MEM_FRACTION: The default value is 0.5. • REASONING_PARSE: reasoning parser. Optional value: deepseek_r1. • TOOL_CALL_PARSER: tool parser. Optional value: deepseek_v3. • MAX_BATCH_SIZE: maximum batch size. The default value is 128. • MAX_INPUT_TOKEN_LENGTH: maximum input

				<p>length. The default value is 4000.</p> <ul style="list-style-type: none"> • MAX_SEQ_LEN: maximum context length (including output). The default value is 4500.
GPT-OSS series models	GPT-OSS-20B	vllm-0.10.1-gptoss-ti-20250811	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. The default value is 65536. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 128. • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.85. • SERVED_MODEL_NAME: model name. The default value is the service ID. • QUANTIZATION: quantization 	-
	GPT-OSS-120B			

			method, which is not configured by default.	
--	--	--	--	--

<p>Cosmos-Reason1 series models</p>	<p>Cosmos-Reason1-7B</p>	<p>vllm-0.9.1-ti-20250703</p>	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. The default value is 65536. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 128. • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.85. • SERVED_MODEL_NAME: model name. The default value is the service ID. • QUANTIZATION: quantization method, which is not configured by default. 	<p>-</p>
<p>Kimi K2 series</p>	<p>Kimi-K2-</p>	<p>sglang-v0.4.9.p</p>	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context 	

models	Instruct	ost2-dev20250715	<p>length, which is automatically populated.</p> <ul style="list-style-type: none"> • MAX_RUNNING_REQUESTS: maximum number of concurrent requests, which is automatically populated. • MAX_CUDA_GRAPH_BS: upper limit for CUDA graph size. • CHUNKED_PREFILL_SIZE: --chunked-prefill-size. It is automatically populated. • MEM_FRACTION_STATIC: --mem-fraction-static. It is automatically populated. For full-featured DeepSeek, the value is adjusted. • ENABLE_MTP: whether to enable MTP acceleration for parallel decoding. By default, this parameter is not set. This 	<ul style="list-style-type: none"> • MAX_CUDA_GRAPH_BS: The default value is 128. • ENABLE_EP_MOE: whether to enable EP MOE acceleration. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek. • QUANTIZATION: quantization method, which is not configured by default. • DP_SIZE: the --dp-size parameter used when DP acceleration is enabled, which is not configured by default. • ENABLE_MULTIMODAL: whether to enable multimodal support. By default, this parameter is not set.
	Kimi-K2-Instruct-0905			

			<p>feature is enabled by default for full-featured DeepSeek with 16 GPU cards.</p> <ul style="list-style-type: none"> • SEPARATE_REASONING: whether to return the chain of thought via reasoning_content. The default value is 0. • AUTO_TEMPLATE: whether to automatically set the chat template, the tool calling template, and the chain-of-thought parsing template. The default value is 1. 	
Qwen series models	qwen_14b	sglang-v0.4.6.post4-dev20250716	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length, which is automatically populated. For full-featured DeepSeek, the default value is 64k. • MAX_RUNNING_REQUESTS: maximum number of 	<ul style="list-style-type: none"> • ENABLE_EP_MOE: whether to enable EP MOE acceleration. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek. • QUANTIZATION: quantization method, which is not configured by default. • DP_SIZE: the --dp-size parameter used when DP acceleration is
	qwen_14b_chat			
Qwen3 series models	Qwen3-0.6B			

concurrent requests, which is automatically populated. For full-featured DeepSeek, the value is adjusted.

- **MAX_CUDA_GRAPH_BS:** upper limit for CUDA graph size, which is automatically populated. For full-featured DeepSeek, the value is adjusted.
- **CHUNKED_PREFILL_SIZE:** -chunked-prefill-size. It is automatically populated.
- **MEM_FRACTION_STATIC:** -mem-fraction-static. It is automatically populated. For full-featured DeepSeek, the value is adjusted.
- **ENABLE_MTP** : whether to enable MTP acceleration for parallel

enabled, which is not configured by default.

- **ENABLE_MULTIMODAL:** whether to enable multimodal support. By default, this parameter is not set.

Qwen3
-1.7B

Qwen3
-4B

Qwen3
-8B

Qwen3
-14B

Qwen3

-32B	<p>decoding. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek with 16 GPU cards.</p> <ul style="list-style-type: none"> • SEPARATE_REASONING: whether to return the chain of thought via reasoning_content. • reasoning_content: The default value is 0. • AUTO_TEMPLATE: whether to automatically set the chat template, the tool calling template, and the chain-of-thought parsing template (DeepSeek, Qwen3, and QwQ). The default value is 1.
Qwen3-0.6B-FP8	
Qwen3-1.7B-FP8	
Qwen3-4B-FP8	
Qwen3-8B-FP8	
Qwen3-14B-FP8	
Qwen3-32B-FP8	
Qwen3-30B-A3B	
Qwen3-30B-A3B-FP8	
Qwen3-235B-A22B	
Qwen3-235B-A22B-FP8	

Qwen3-30B-Instruct-2507	vllm-0.9.1-ti-20250725	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length. • MAX_BATCH_SIZE: maximum number of concurrent requests. The default value is 128. 	<p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set to <code>--enable-auto-tool-choice --tool-call-parser hermes</code>, which indicates that tool calling parsing is enabled with the default tool parser set to hermes.</p>
Qwen3-235B-A22B-Instruct-2507		<ul style="list-style-type: none"> • MAX_NUM_BATCHED_TOKENS: maximum number of concurrent tokens. The default value is 8000. 	<p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set to <code>--enable-expert-parallel --enable-auto-tool-choice --tool-call-parser hermes</code>, which indicates that expert parallelism and tool calling parsing are enabled with the default tool parser set to hermes.</p>
Qwen3-235B-A22B-Instruct-2507-FP8		<ul style="list-style-type: none"> • GPU_MEMORY_UTILIZATION: ratio of GPU memory reserved. The default value is 0.85. • SERVED_MODEL_NAME: model name. The default value is the service ID. 	<p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set to <code>--enable-reasoning --</code></p>
Qwen3-235B-A22B-Thinking-2507		<ul style="list-style-type: none"> • QUANTIZATION: quantization method, which is not configured by default. 	<p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set to <code>--enable-reasoning --</code></p>

	<p>Qwen3-235B-A22B-Thinkin-g-2507-FP8</p>		<p>reasoning-parser deepseek_r1 --enable-auto-tool-choice --tool-call-parser hermes, which indicates that chain-of-thought parsing is enabled with the chain-of-thought parser set to deepseek_r1 and tool calling parsing is enabled with the default tool parser set to hermes.</p> <p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set to --enable-expert-parallel --enable-reasoning --reasoning-parser deepseek_r1 --enable-auto-tool-choice --tool-call-parser hermes, which indicates that expert parallelism and chain-of-thought parsing are enabled with the chain-of-thought parser set to deepseek_r1 and tool calling parsing is enabled with the default tool parser set to hermes.</p>
<p>Qwen3 Coder series models</p>	<p>Qwen3 - Coder-480B-A35B-Instruct</p>		<p>EXTRA_PARAMS: extra parameters directly passed to vllm as startup commands. For parameter meanings, see the vllm documentation. Any parameter that needs to be enabled can be passed through this environment variable. By default, it is set</p>

	<p>Qwen3 - Coder- 480B- A35B- Instruct -FP8</p>		<p>to <code>--enable-expert-parallel</code> <code>--enable-auto-tool-choice</code> <code>--tool-call-parser</code> <code>qwen3_xml</code> <code>--tool-parser-plugin</code> <code>examples/qwen3coder_tool_parser.py</code>, which indicates that expert parallelism and tool calling parsing are enabled with the default tool parser set to <code>qwen3_xml</code>. It also specifies a custom tool calling parsing plugin.</p>
<p>Qwen3-VL series</p>	<p>Qwen3 -VL- 235B- A22B- Instruct</p>	<p>vllm- 0.11.0- ti- dev202 51009</p>	<ul style="list-style-type: none"> • MAX_MODEL_LEN: 128000 • GPU_MEMORY_UTILIZATION: 0.85 • EXTRA_PARAMS: The default value is <code>--limit-mm-per-prompt.video 0</code> <code>--limit-mm-per-prompt.image 4</code> <code>--enable-log-requests</code>, which enables logging and limits the number of image and video files per request.
<p>Qwen2.5-VL series models</p>	<p>Qwen2.5-VL-32B-Instruct</p>	<p>vllm- 0.10.0- ti- 202508 21</p>	<ul style="list-style-type: none"> • MAX_MODEL_LEN The default value is 64000. • LIMIT_MM_PER_PROMPT: Limit on the prompt content. The default value is <code>{"images":2}</code>, which indicates that each prompt can include a maximum of 2 images. • MM_PROCESSOR_KWARGS: used to modify model-specific parameters. The default value is <code>{"max_pixels":3686400}</code>.
	<p>Qwen2.5-VL-72B-Instruct</p>		
	<p>Qwen2.5-VL-72B-Instruct-AWQ</p>		

Qwen3 Embedding series models	Qwen3 - Embedding-8B	vllm-0.9.1-ti-proxy-20250827		<ul style="list-style-type: none"> • HF_OVERRIDES: used to specify parameters that override the model's config.json configuration file. The default value for this model is {"is_matryoshka":true}, which indicates support for Matryoshka embedding. • PROXY: whether to enable proxy. The default value is False.
QwQ series models	QwQ-32B	sglang-v0.4.6.post4-dev20250716	<ul style="list-style-type: none"> • MAX_MODEL_LEN: context length, which is automatically populated. • MAX_RUNNING_REQUESTS: maximum number of concurrent requests, which is automatically populated. • MAX_CUDA_GRAPH_BS: upper limit for CUDA graph size. • CHUNKED_PREFILL_SIZE: --chunked-prefill-size. It is automatically populated. • MEM_FRACTION_STATIC: --mem-fraction-static. It is 	<ul style="list-style-type: none"> • MAX_CUDA_GRAPH_BS: This parameter is automatically populated. • ENABLE_EP_MOE: whether to enable EP MOE acceleration. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek. • QUANTIZATION: quantization method, which is not configured by default. • DP_SIZE: the --dp-size parameter used when DP acceleration is enabled, which is not configured by default. • ENABLE_MULTIMODAL: whether to enable multimodal support. By default, this parameter is not set.

automatically populated. For full-featured DeepSeek, the value is adjusted.

- **ENABLE_MTP**: whether to enable MTP acceleration for parallel decoding. By default, this parameter is not set. This feature is enabled by default for full-featured DeepSeek with 16 GPU cards.
- **SEPARATE_REASONING**: whether to return the chain of thought via reasoning_content. The default value is 0.
- **AUTO_TEMPLATE**: whether to automatically set the chat template, the tool calling template, and the chain-of-thought parsing template. The

			default value is 1.	
Gemma 3 series models	gemma-3-27b-it	vllm-0.10.0-ti-20250821	-	<ul style="list-style-type: none"> • MAX_MODEL_LEN: The default value is 64000. LIMIT_MM_PER_PROMPT: Limit on the prompt content. The default value is {"images":2}, which indicates that each prompt can include a maximum of 2 images. • MM_PROCESSOR_KWARGS: used to modify model-specific parameters. The default value is {"max_pixels":3686400}.
	gemma-3-12b-it		-	
GLM-4.5V series models	GLM-4.5V	vllm-0.10.2-ti-dev20250918	-	EXTRA_PARAMS: -- enable-auto-tool-choice -- -tool-call-parser glm45 -- reasoning-parser glm45 -- media-io-kwarg {"video": {"num_frames":-1}}

Inference Acceleration Based on the Built-in Image Angel-vLLM

Last updated: 2026-03-12 09:41:51

Overview

This document takes qwen2-7b-instruct as an example to demonstrate how to use Tencent Cloud TI-ONE Platform (TI-ONE) to deploy and perform inference acceleration on custom LLMs. It also provides usage instructions for accelerated quantization performance testing through test scripts. You may also use the scripts provided in this document to test the inference acceleration effects achieved through TI-ONE's built-in inference image Angel-vLLM.

Angel-vLLM is an LLM inference acceleration framework deeply optimized by the Tencent Cloud AI acceleration team based on open-source vLLM. While using the same API as the community vLLM and maintaining full compatibility with the community vLLM features, it has the following features:

- 1. Richer features.** Compared to the community open-source edition of vLLM, Angel-vLLM provides INT8, NF4, and FP8 online quantization, lookahead parallel decoding, and other features.
- 2. More powerful performance.** Compared to the community open-source edition of the vLLM, Angel-vLLM quantization not only saves GPU memory but also reduces latency and increases throughput. Lookahead parallel decoding has been refined through practical business applications, showing significant improvement in RAG scenarios.
- 3. Better aligned precision.** Angel-vLLM generation results have been extensively tested in online business operations, achieving complete alignment with Hugging Face generation results or keeping the precision basically unchanged.

TI-ONE has a built-in inference image integrated with the Angel-vLLM inference acceleration framework, enabling one-click deployment of LLM inference services compatible with the OpenAI API protocol.

This document will introduce how to use the built-in image Angel-vLLM on TI-ONE to deploy LLM inference services, enable Angel-vLLM acceleration capabilities, and call APIs.

Prerequisites

Before deploying and using your own model via TI-ONE and its built-in inference acceleration image Angel-vLLM, you need to prepare the following resources (including Cloud File Storage (CFS) for storing model files and GPU computing resources for the inference model).

Cloud File Storage

Apply for CFS: During the deployment and inference of custom LLMs, CFS is used for model file storage. Therefore, you need to apply for CFS first. For differences between CFS instances, see [CFS > Storage Types and Specifications](#). Ensure network connectivity between the purchased CFS instance and the computing resource Cloud Virtual Machine (CVM) instances mentioned above. For details about CFS usage, see [CFS > Creating File Systems and Mount Points](#).

Note: If your underlying storage service is Data Accelerator Goose FileSystem extreme (GooseFSx), the operations are similar to CFS described in this document, so you can still use this document as a reference.

Computing Resources

- Self-purchased computing resources: If you are using TI-ONE for the first time, see Resource Group Management to purchase appropriate computing resources based on the inference model parameters.
- Select resources based on the actual model size or your available resources. The CVM instance resources required for LLM inference depend on the model parameter scale and context length. We recommend referring to the following table to quickly estimate the resources required for inference services. For detailed GPU memory usage, see [Inference Resource Requirements](#).

Model Parameter Scale	GPU Card Type and Quantity
6 ~ 8B	PNV5b * 1 / A10 * 1 / A100 * 1 / V100 * 1
12 ~ 14B	PNV5b * 1 / A10 * 2 / A100 * 1 / V100 * 2
65 ~ 72B	PNV5b * 8 / A100 * 8

Operation Steps

Model Preparations

Choose [Training Workshop > Dev Machines > Create](#) to create a dev machine:

- Image: You may choose any built-in image, as this dev machine is only used for downloading model files.
- Source of Machine: If you have not purchased any computing resources previously, you can select Purchase from the TI-ONE platform and pay-as-you-go billing mode. To select from CVM instances or select the yearly/monthly subscription billing mode, see the computing resource purchase instructions in the Prerequisites section.
- Billing Mode: You can select either pay-as-you-go or yearly/monthly subscription billing mode.

- **Storage Path Configuration:** Select CFS. The name should be the CFS instance you applied for and configured in the Prerequisites section. The path is the root directory by default and is used to specify the storage location for your custom LLM.
- **Other Settings:** Not required by default.

Note: We download or upload the required model files through the dev machine. The resources used in the example are 8 CPU cores and 16 G of memory (they are used only for the dev machine and are unrelated to subsequent inference services; resources can be appropriately reduced).

The final configuration is as follows:

Create Dev Machine

Name: test-cfs (8/256)

Region: Shanghai

Image: Built-in common image / tlearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu

Source of Machine: Purchased on TI-ONE

Billing mode: Pay-as-you-go

Spec: 8C16G

System Disk: Temporary Storage

Storage Path Settings:

Storage Type	CFS system	Source path	Container Mounting Path	Operation
CFS	Please select	/	/home/tione/notebook	Delete

Model Files

After the dev machine is successfully created, start it. Then, click the Python3 kernel under the dev machine to create an ipynb page for downloading the required models via scripts. You can search for the LLM you need in [ModelScope](#) or [Hugging Face](#), and use the Python scripts provided in these communities to download the model and save it to CFS. This document takes the **Qwen2-7b-Instruct** model as an example, with the download code as follows:

```
!pip install modelscope

from modelscope import snapshot_download
#qwen/Qwen2-7B-Instruct is the name of the model to be downloaded, and
cache_dir is the address where the downloaded model will be saved. Here,
/home/tione/notebook indicates saving the downloaded model in the root
directory of the CFS mount directory.
```

```
model_dir = snapshot_download('qwen/Qwen2-7b-Instruct',
cache_dir='/home/tione/notebook')
```

Note: The path mapping relationship is as follows: Taking the container mount path `/home/tione/notebook` when CFS is mounted as an example, if the CFS source path is `/` and `cache_dir` is `/home/tione/notebook`, the model is actually saved at the root path of CFS; if the CFS path is `/dir` and `cache_dir` is `/home/tione/notebook/model`, the model is actually saved in the `/dir/model` directory of CFS.

Copy the above download script, replace `qwen/Qwen2-7b-Instruct` with the model you need to download, paste it into the new ipynb file, and click the **run** icon (or click the cell and press **Enter+Shift**) to start downloading the model.

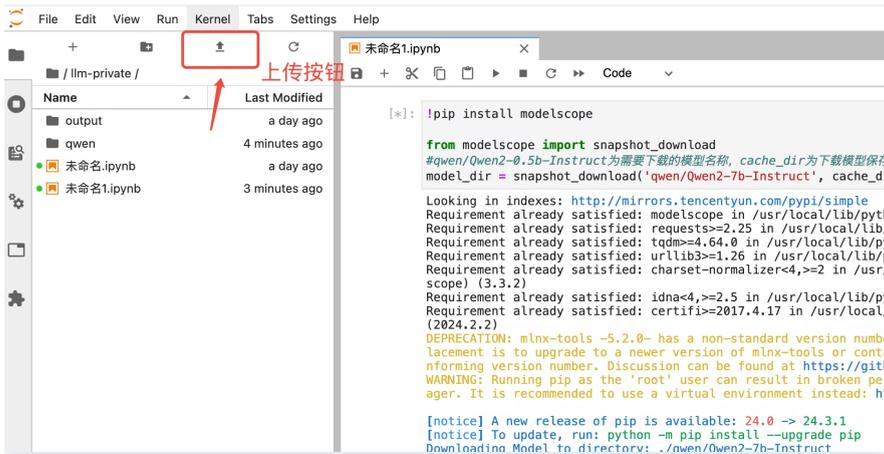
```
[!]: !pip install modelscope
from modelscope import snapshot_download
qwen/Qwen2-7b-Instruct为需要下载模型的名称, cache_dir为下载模型保存的路径, 这里 './' 表示将下载模型保存在CFS挂载目录的根目录中
model_dir = snapshot_download('qwen/Qwen2-7b-Instruct', cache_dir='./')

Looking in indexes: http://mirrors.tencentyun.com/pypi/simple
Requirement already satisfied: modelscope in /usr/local/lib/python3.10/dist-packages (1.22.2)
Requirement already satisfied: requests>=2.25 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.32.3)
Requirement already satisfied: tqdm>=4.64.8 in /usr/local/lib/python3.10/dist-packages (from modelscope) (4.66.2)
Requirement already satisfied: urllib3>=1.26 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.2.2)
Requirement already satisfied: charset-normalizer>=4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->modelscope) (3.3.2)
Requirement already satisfied: idna>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->modelscope) (3.6)
Requirement already satisfied: certifi>=2023.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->modelscope) (2024.2.2)
DEPRECATION: mlox-tools<5.2.0 has a non-standard version number. pip 24.1 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of mlox-tools or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at https://github.com/pypa/pip/issues/12063 class=ansi-yellow-fg>
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv class=ansi-yellow-fg>

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] to update, run: python -m pip install --upgrade pip
Downloading Model to directory: ./qwen/Qwen2-7b-Instruct
2025-01-22 16:11:04.852 - modelscope - INFO - Got 14 files, start to download ...
Processing 14 items: 0% | 0.00/14.0 [00:00<, 715/s]
Downloading [generation_config.json]: 0% | 0.00/243 [00:00<, 78/s]
Downloading [model-00001-of-00004.safetensors]: 0% | 0.00/3.63G [00:00<, 78/s]
Downloading [config.json]: 0% | 0.00/48.8 [00:00<, 78/s]
Downloading [model-00003-of-00004.safetensors]: 0% | 0.00/3.68G [00:00<, 78/s]
Downloading [config.json]: 0% | 0.00/663 [00:00<, 78/s]
Downloading [LICENSE]: 0% | 0.00/11.1k [00:00<, 78/s]
Downloading [merges.txt]: 0% | 0.00/1.59M [00:00<, 78/s]
Downloading [model-00002-of-00004.safetensors]: 0% | 0.00/3.68G [00:00<, 78/s]
Processing 14 items: 7% | 1.00/14.0 [00:00:00.04, 2.741/s]
Downloading [model-00004-of-00004.safetensors]: 0% | 0.00/3.31G [00:00<, 78/s]
Downloading [model.safetensors.index.json]: 0% | 0.00/27.1k [00:00<, 78/s]
Downloading [tokenizer.json]: 0% | 0.00/6.41k [00:00<, 78/s]
Downloading [tokenizer.json]: 0% | 0.00/6.79M [00:00<, 78/s]
Processing 14 items: 36% | 5.00/14.0 [00:00:00.01, 6.741/s]
```

Additionally, you can upload your existing LLM files locally, as shown in the figure below:

Note: The bandwidth for this upload method is limited. It is recommended to be used only for files smaller than 10 MB. For larger models, it is recommended to upload via Tencent Cloud Object Storage (COS) (preferably to a COS bucket in the same region as CFS). For desktop COS usage, see [COS > User Instructions for Desktop Version](#). For command-line COS usage, see [COS > COSCMD](#) or [COS > COSCLI](#). Then, install the `coscmd` or `coscli` tool in the dev machine to download files to CFS.



Deployment Process

Go to the **Model Services > Online Services** page in the TI-ONE console, and click **Create Service**:

- **Service name:** Defined by you.
- **Source of Machine:** Choose either **Select from CVM instances** or **Purchase from the TI-ONE platform**, and then select the corresponding resource group or use **pay-as-you-go** mode to select CVM instance specifications on demand.
- **Deployment mode:** If the model can be deployed using the resources of a single machine, standard deployment is recommended. If you need to deploy very large models where a single machine's resources are insufficient, you can select the multi-machine distributed deployment mode.
- **Replica settings:**
 - **Model Source:** Choose **Cloud Storage > CFS**.
 - **Model:** Select the CFS instance where the model is located, then enter the source path of the model file on CFS. If you need to select a specific checkpoint from the training output, enter the actual training output path and ensure to specify up to the checkpoint-level directory.
 - **Image:** Choose **Built-in/LLM/Angel-vLLM(2.1)**.
 - **Resource:** Select the required computing resources for the model on demand. For resource requirements, see the appendix **Inference Resource Requirements**.
 - **Advanced Settings:** For the starting command and environment variables, see **Service Deployment Parameters**.

The screenshot displays the configuration page for a service in the Tencent Cloud TI-ONE Platform. The settings are as follows:

- Model source:** Radio buttons for Built-in LLM (selected), Cloud Storage (CFS/GooseFSx/COS), Container Image, Resource Group Preloading, and Data Source. A Model Repository checkbox is also present.
- Model:** Tabs for CFS, GooseFSx, and COS. A 'Select CFS' dropdown and a 'CFS Source Path' input field (containing '/data/model') are visible.
- Image:** Radio buttons for Built-in (selected) and Custom. A dropdown menu shows 'LLM / angel-vllm-cu128(0.10.2)'.
- Storage Mount:** A '+ Add' button.
- Enable gRPC:** A toggle switch.
- Port:** A text input field containing '8501'.
- Resource:**
 - GPU Type:** A dropdown menu with a link to 'A Guide to Resources Required for Large Model Inference'.
 - GPU:** A numeric input field set to '0.0' with a unit of 'card(s)'. A note below states: 'To use GPU, set the number of cards to 0.1 to 1 or a multiple of 1 according to the gpu type. The number of cards cannot be 0 if the runtime environment is the built-in GPU image of the platform.'
 - CPU:** A numeric input field set to '1.0' with a unit of 'core'.
 - Memory:** A numeric input field set to '1.00' with a unit of 'G'.
- Advanced settings:** A link to expand more options.

Click **Start Service**, and the service will enter the creation process. The service status may remain in the **Preparing** status for a period, indicating that the service startup is not yet completed. You can view the startup progress by clicking the **Events** or **Logs** tabs of a specific service.

Service Deployment Parameters

Startup Command Description

The image's default startup command (that is, the default value when no startup command is specified) is `run`. Its overall feature is essentially equivalent to the startup command `vllm serve` or `python3 -m vllm.entrypoints.openai.api_server` provided by vLLM.

The main differences between the default startup command and the startup command provided by vLLM are as follows:

- Added the feature for setting certain parameters via environment variables. For details, see [Description of Startup Command Parameters and Environment Variables](#) below.
- Adjusted the default values of certain parameters and added support for automatic configuration of certain parameters. For details, see [Description of Startup Command Parameters and Environment Variables](#) below.
- When the multi-machine distributed deployment mode is used, the multi-machine Ray communication environment is automatically established.
- Added support for quick startup of Low-Rank Adaptation (LoRA) models from built-in LLM training on TI-ONE (automatically enabling multi-LoRA or performing LoRA weight merging).
- Optimized the memory pre-reading loading speed of models on CFS Turbo storage media.

All the following startup command parameters are compatible with the three startup commands: `run`, `vllm serve`, and `python3 -m vllm.entrypoints.openai.api_server`, while environment variables are only supported by the default `run` startup command.

Description of Startup Command Parameters and Environment Variables

Parameter Details

Angel-vLLM core acceleration feature parameters:

Startup Command Parameter	Environment Variable	Meaning
<code>--quantization</code>	QUANTIZATION	<p>Quantization method, not set by default. Compared to the open-source vLLM 0.6.2 version, the following quantization methods are added:</p> <ul style="list-style-type: none"> • <code>FP8</code> : W8A8 FP8 precision quantization, supported on NVIDIA GPU series with a compute capability of 8.9 or higher (Ada Lovelace and Hopper); reduces GPU memory usage for model weights by 50%, improves inference speed, and maintains near-lossless precision. • <code>ifq_nf4</code> : W4A16 NF4 precision quantization, supported on NVIDIA GPU series with a compute capability of 8.0 or higher (Ampere, Ada Lovelace, and Hopper); reduces GPU memory usage for model weights by 75%, improves inference speed, and offers better precision compared to INT4 quantization (this quantization method only supports quantization based on float16 precision, that is, <code>--dtype float16</code>). • <code>ifq</code> : W8A16 INT8 precision quantization, supported on NVIDIA GPU series with a compute capability of 7.0 or higher (Volta, Turing, Ampere, Ada Lovelace, and Hopper); reduces GPU memory usage for model weights by 50%, improves inference speed, and maintains near-lossless precision. <p>All these 3 quantization methods support one-click online quantization, without the need for prior model conversion or offline calibration.</p> <p>GPU memory usage: <code>ifq_nf4</code> < <code>FP8</code> = <code>ifq</code> < non-quantized</p> <p>Inference speed: <code>FP8</code> >= <code>ifq_nf4</code> > <code>ifq</code> > non-quantized</p>

		Model precision: non-quantized >= FP8 >= ifq > ifq_nf4
<code>--use-lookahead</code>	USE_LOOKAHEAD	The default value is 0. Setting it to 1 enables lookahead parallel decoding, which can significantly speed up decoding in many scenarios. Compared to the open-source implementation, lookahead parallel decoding does not require additional small models or model heads and can achieve results consistent with non-parallel decoding. It shows noticeable acceleration effects when a large portion of the output text has appeared in the input text (such as RAG scenarios), or when there are similar requests or answers in a large number of requests (for acceleration effects, see Appendix).
<code>--num-speculative-tokens</code>	NUM_SPECULATIVE_TOKENS	The default value is 6. This parameter defines the number of tokens decoded at a time during lookahead parallel decoding. If the required concurrency is high, this value can be decreased. If the concurrency is low, it can be increased.

To be compatible with more scenarios, the platform's built-in image has adjusted the default values of certain startup parameters. The following startup parameters **differ** from vLLM's default startup parameters:

Startup Command Parameter	Environment Variable	Meaning
<code>--max-model-len</code>	MAX_MODEL_LENGTH	Maximum context length of the model. To be compatible with instance types with smaller GPU memory, the platform modifies the parameter value to <code>8192</code> for models with a context length greater than 8K by default. If you need to support a longer context, you can manually configure this parameter.
<code>--dtype</code>	DTYPE	The default value is <code>float16</code> . If you want to perform inference with bfloat16 precision, manually change the value to bfloat16.
<code>--enable-prefix-caching</code>	ENABLE_PREFIX_CACHING	The default value is <code>true</code> , which means the prefix caching feature is enabled. This significantly accelerates processing for long-text inputs with

		repeated prefixes and reduces Time to First Token (TTFT) in continuous multi-turn chat sessions. You can manually disable it by setting the environment variable <code>ENABLE_PREFIX_CACHING=false</code> . (For V100 GPUs, as this feature is not yet supported, the default value is false).
<code>--tensor-parallel-size</code>	TP	The default value is the number of GPU cards specified when the online service is created, indicating the model parallelism size.
<code>--trust-remote-code</code>	TRUST_REMOTE_CODE	In order to be compatible with models that require this option to be enabled, the default value is <code>true</code> . You can set the environment variable <code>TRUST_REMOTE_CODE</code> to false to manually disable it.
<code>--model</code>	MODEL	Model name or path. The default value is <code>/data/model</code> , which corresponds to the default model mount path within the platform's container.
<code>--port</code>	-	Service port. The default value is <code>8501</code> , which corresponds to the default inference service port on the platform.
<code>--use-v2-block-manager</code>	-	The default value is <code>true</code> , indicating that the v2 block manager is used.
<code>--chat-template</code>	-	Hugging Face chat template. It can be a <code>.jinja</code> chat template file path or template string. (If you do not set this parameter but have specified the <code>MODEL_ID</code> or <code>CONV_TEMPLATE</code> environment variables, or if the model directory contains a <code>ti_model_config.json</code> file, the platform will attempt to perform automatic matching. For details, see Chat Templates).
<code>--tool-call-parser</code>	-	Tool call parser. Valid values: <code>llama3_json</code> , <code>hermes</code> , <code>mistral</code> , and <code>internlm</code> . (If you do not set this parameter but have specified the <code>MODEL_ID</code> or <code>CONV_TEMPLATE</code> environment variables, or if the model directory contains a <code>ti_model_config.json</code> file, the platform will attempt to perform automatic matching. For details, see Chat Templates).

<code>--enable-auto-tool-choice</code>	-	<p>Enables the automatic tool call capability, which must be used in conjunction with <code>--tool-call-parser</code>.</p> <p>(If you do not set this parameter but have specified the <code>MODEL_ID</code> or <code>CONV_TEMPLATE</code> environment variables, or if the model directory contains a <code>ti_model_config.json</code> file, the platform will attempt to perform automatic matching. For details, see Chat Templates).</p>
--	---	---

Other parameters that can be configured via environment variables supported by the platform image:

Startup Command Parameter	Environment Variable	Meaning
<code>--gpu-memory-utilization</code>	<code>GPU_MEMORY_UTILIZATION</code>	GPU memory utilization. The default value is <code>0.9</code> .
<code>--enforce-eager</code>	<code>ENFORCE_EAGER</code>	Whether to forcibly enable PyTorch's eager mode. The default value is <code>false</code> , in which case CUDA Graph is additionally used for further acceleration, but this consumes extra GPU memory and increases service startup time.
-	<code>MODEL_ID</code>	Model name. The default value is <code>model</code> . It is recommended to set it to the model's name on Hugging Face, which will trigger automatic matching for the chat template .
-	<code>CONV_TEMPLATE</code>	Chat template name. It is not set by default and primarily used for automatic matching of the chat template .
-	<code>DISABLE_MEMORY_CACHE</code>	Disables memory cache pre-reading. The default value is <code>false</code> .

For the remaining services-supported startup parameters, see vLLM's official documentation [OpenAI Compatible Server](#).

Use Instructions

To specify the startup command: **Advanced Settings > Startup Command**. For example, to use `ifq_nf4` quantization, the configuration is as follows (to adjust other parameters, see the service

deployment parameters below):



To specify environment variables: **Advanced Settings > Environment Variables**. For example, to use ifq_nf4 quantization, the configuration is as follows (the method for using other parameters via environment variables is the same; **note that environment variable names must be in uppercase**):



Recommended Parameters for Different Scenarios

In most cases, you can start the inference service without entering any startup commands or environment variables. For specific scenarios, we provide configuration guidance for certain startup parameters here.

Case – Model Experience – Minimum Resource Requirements

- **Scenario:** The computing resources for inference are limited. Insufficient GPU memory is reported when you deploy the model with default parameters. The goal is only for a quick trial of the model, with no specific requirements for model precision or inference speed.
- **Recommended startup command:** `run --quantization ifq_nf4 --enforce-eager --gpu-memory-utilization 0.95 --max-model-len 2048`
- **Description:**
 - `--quantization ifq_nf4` : enables 4-bit quantization, reducing GPU memory occupied by model weights by 75%. (This feature is supported on Ampere, Ada Lovelace, or Hopper series GPUs. For V100 or T4 GPUs, enable the `ifq` quantization mode.)
 - `--enforce-eager` : disables CUDA Graph, reducing additional GPU memory usage and allowing for a higher value for `--gpu-memory-utilization`.
 - `--gpu-memory-utilization 0.95` : increases GPU memory utilization to 95%, enhancing GPU memory usage efficiency.
 - `--max-model-len 2048` : reduces the context length to 2048, decreasing the GPU memory required for KV cache.

Case – Production Environment Deployment – Resume Parsing

- **Scenario:** There are higher requirements for both model precision and inference speed. In resume parsing scenarios, the input text is long, and the generated content is highly likely to overlap with the input, making it suitable to enable parallel decoding acceleration.

- **Recommended startup command:** `run --dtype auto --quantization fp8 --use-lookahead --num-speculative-tokens 4 --max-model-len 32768 --enable-prefix-caching --disable-log-requests --api-keys mock_api_key --served-model-name model_123 .`
- **Description:**
 - `--dtype auto` : automatically uses the precision configured in the model's config.json file. (For certain models, the effects of fp16 and bf16 may be different; adjust based on actual testing conditions.)
 - `--quantization fp8` : enables FP8 W8A8 quantization, significantly accelerating inference while maintaining precision and reducing GPU memory occupied by model weights by 50%. (This feature is supported on Ada Lovelace and Hopper series GPUs. For other GPU models, enable the `ifq` quantization mode.)
 - `--use-lookahead` : enables parallel decoding to accelerate generation in this scenario and increase throughput.
 - `--num-speculative-tokens 4` : specifies the number of tokens for parallel decoding. When support for high concurrency is required, it is recommended to set a smaller value. This value can be adjusted based on actual testing to evaluate throughput.
 - `--max-model-len 32768` : sets the maximum supported context length as needed.
 - `--enable-prefix-caching` : enables the prefix caching feature. This option is enabled by default in the run startup command and can be omitted.
 - `--disable-log-requests` : disables the logging of request details to prevent excessive and overly lengthy request information in logs.
 - `--api-keys mock_api_key` : sets an authentication token compatible with OpenAI API calls to prevent unauthorized access to public network APIs.
 - `--served-model-name model_123` : specifies the model name, making it easier for clients to identify which model returns the inference result.

Case – Using the vLLM Native Startup Command – Throughput Optimization

- **Scenario:** You want to deploy the service using vLLM's native startup command and optimize service throughput as much as possible.
- **Recommended startup command:** `vllm serve /data/model --port 8501 -tp 2 --quantization fp8 --max-model-len 32768 --enable-prefix-caching --disable-log-requests --served-model-name model_123 --num-scheduler-steps 8`
- **Description:**
 - `/data/model` : sets the model path to the default container mount path.
 - `--port 8501` : sets the service port to the default platform port 8501.
 - `-tp 2` : sets the model tensor parallelism size to 2. This value should be adjusted based on the actual number of GPU cards used by the service, and can be omitted for single-GPU card inference.

- `--quantization FP8` : enables FP8 W8A8 quantization, significantly accelerating inference while maintaining precision and reducing GPU memory occupied by model weights by 50%. (This feature is supported on Ada Lovelace and Hopper series GPUs. For other GPU models, enable the `ifq` quantization mode.)
- `--max-model-len 32768` : sets the maximum supported context length as needed.
- `--enable-prefix-caching` : enables the prefix caching feature to reduce TTFT.
- `--disable-log-requests` : disables the logging of request details to prevent excessive and overly lengthy request information in logs.
- `--served-model-name model_123` : specifies the model name, making it easier for clients to identify which model returns the inference result.
- `--num-scheduler-steps 8` : enables multi-step scheduling to optimize GPU utilization and improve service throughput (this option conflicts with parallel decoding; it is recommended to choose either this or parallel decoding based on actual business needs).

Chat Experience

After the service is ready, you can see the Chat Experience tab. Click the tab to go to the page for an online chat experience.

Chat API Call

The service also supports direct requests via HTTP/HTTPS protocol. Click the Service Call page, and enter the chat API endpoint `/v1/chat/completions` in the API calling address field. For the API format, see the format of the OpenAI Chat Completions API.

Below are several common call scenarios:

Plain Text Chat

Example request:

```
{"messages":[{"role":"user","content":"Hello"}],"max_tokens":128}
```

Tool Call

This is supported by the API only when the tool call capability is enabled during service deployment. For details, see [FAQs](#).

Example request:

```
{
  "tools": [
    {
      "type": "function",
```

```
    "function": {
      "name": "get_current_weather",
      "description": "Get the current weather in a given
location",
      "parameters": {
        "type": "object",
        "properties": {
          "city": {
            "type": "string",
            "description": "The city to find the weather
for, e.g. 'Beijing'"
          }
        },
        "required": [
          "city"
        ]
      }
    },
  ],
  "messages": [
    {"role": "user", "content": "what's the weather of Shanghai
today?"}
  ],
  "max_tokens": 512
}
```

Multimodal

This is supported only when the model supports multimodal capabilities, such as the [Qwen/Qwen2-VL-7B-Instruct](#) and [meta-llama/Llama-3.2-11B-Vision-Instruct](#) models.

Example request:

```
{
  "messages": [
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "What's in this image?"
        }
      ]
    }
  ]
}
```

```
    },
    {
      "type": "image_url",
      "image_url": {
        "url": "https://tione-public-cos-1308945662.cos.ap-shanghai.myqcloud.com/test/cat.jpg"
      }
    }
  ]
}
],
"max_tokens": 512
}
```

Third-Party LLM Application Integration

Taking Dify as an example, you can add a model of type OpenAI-API-compatible, and then enter the service's online calling address in the API endpoint URL field, appending `/v1` to the end.

Server Performance Testing

Note: The following test uses random data. If you need to test the parallel decoding acceleration capability, it is recommended to use real business data for testing.

Start up the dev machine you prepared in the model preparation step above (ensure that the CFS instance is mounted with the LLM). Download the test scripts from the official vLLM project at [vllm/benchmarks at main · vllm-project/vllm · GitHub](#) (including `benchmark_serving.py` and `backend_request_func.py`). Create a working directory on the dev machine. Create a startup script as follows (you can modify the parameters according to your needs).

```
#!/bin/bash
RESULTS_FOLDER="results"
HOST="https://ms-cxxxxxxx.ti.tencentcs.com/ms-czzfp5c9/"
MODEL="../Qwen2-7B-Instruct"

QPS="inf"
INPUT_LEN=128
OUTPUT_LEN=128
NUM_PROMPTS=20
CONCURRENCY=16
```

```

mkdir -p $RESULTS_FOLDER

client_command="python3 benchmark_serving.py \
--port 8501 \
--base-url $HOST \
--endpoint /v1/chat/completions \
--backend openai-chat \
--model $MODEL \
--dataset-name random \
--random-input-len $INPUT_LEN \
--random-output-len $OUTPUT_LEN \
--ignore-eos \
--request-rate $QPS \
--num-prompts $NUM_PROMPTS \
--save-result \
--result-dir $RESULTS_FOLDER \
--metadata input_len=$INPUT_LEN output_len=$OUTPUT_LEN qps=$QPS
concurrency=$CONCURRENCY"

eval $client_command

```

The meaning of each parameter is described as follows:

Parameter Name	Parameter Meaning
MODEL	Path of the model mounted in CFS
RESULTS_FOLDER	Path to save the final result
INPUT_LEN	Number of tokens input into the model
OUTPUT_LEN	Maximum number of tokens output by the model
NUM_PROMPTS	Total number of requests
CONCURRENCY	Number of concurrent requests
HOST	Calling address

Note: Conventional service calling addresses are not recommended because the time it takes to call linkages may fluctuate due to potential impacts from Web Application Firewall (WAF) and other factors. It is recommended that you use high-speed service

calling addresses to stress-test service performance. That is, configure HOST as a high-speed service calling address. You can add the VPC network where the dev machine resides to the high-speed service call IP range of the online service. The operation guide is as follows:

First, on the service call page, click **Add High-Speed Service Call IP Range**:

The screenshot shows the 'Service call' configuration page. At the top, there are tabs for 'Service management', 'Service call', and 'Service Authentication'. A blue banner at the top states: 'To comply with platform requirements, you have authorized and agreed to service calling when creating the service.' Below this is a 'Call monitoring' button. The main content area is divided into sections: 'Traffic Control Information' (QPS: 500), 'Regular Service Calling' (Call Address: https://ms-q61...), and 'High-Speed Service Calling'. Under 'High-Speed Service Calling', there is a table of VPC Calling Addresses with columns for the address, a delete icon, and a status 'VPCs Allowed for'. A red box highlights the link 'Add Network Segmentation for High-Speed Service Calling'. At the bottom, there is a 'gRPC Protocol' toggle (Not enabled) and a 'CLB Call' toggle.

Next, copy the dev machine's network information (the network it belongs to and the subnet it resides in) as follows:

The screenshot shows the 'Basic information' tab of an instance console. The instance is named 'an...' and is in a 'Submitting' state. The 'Network information' section is highlighted with a red box, showing the 'Network' as 'Defa' (vpc-1c9h5es) and the 'Subnet' as 'Defa' (Subnet 3). Other sections include 'Access entrance', 'Lifecycle script', 'Remote Connection', and 'Custom port'.

Fill in the corresponding information and confirm the addition:

Add Network Segmentation for High-Speed Service Calling

✕

1. Enabling the configuration will apply to all services in all spaces within the region.
2. After configuration, 1 terminal nodes will be created in the VPC and subnet of your account based on the number of services and billed independently. [Terminal Node Billing](#)

VPC • Do not use VPC

Subnet • Please select

OK

Cancel

The final calling address is as follows:

服务管理 服务调用 在线体验

遵守平台要求，服务调用已在创建服务时，授权并同意了《腾讯云 TI-ONE 训练平台服务协议》。

调用监控

常规服务调用

调用地址 `https://tencent.com/ms-czfp5c9`

是否生成鉴权 已关闭

QPS 500 (单服务组的QPS上限为500，如需提高QPS请提交工单，我们将根据您的业务情况进行评估)

高速服务调用

VPC调用地址

<code>http://zzfp5c9</code>	删除 仅限以下VPC调用:
<code>http://czfp5c9</code>	删除 仅限以下VPC调用:
<code>http://p5c9</code>	删除 仅限以下VPC调用:

新增高速服务调用网络

服务访问云上VPC资源

The final test file directory structure is as follows:

File Edit View Run Kernel Tabs Settings

+ + ↑ ↻

/ llm-benchmark /

Name	Last Modified
backend_request_func...	16 hours ago
benchmark_serving.py	17 hours ago
run.sh	6 minutes ago

Create a terminal and run `bash run.sh` to start the test script. You can view the performance test results from the output on the terminal:

```

Starting initial single prompt test run...
Initial test run completed. Starting main benchmark run...
Traffic request rate: inf
Burstiness factor: 1.0 (Poisson process)
Maximum request concurrency: None
100% ██████████ 20/20 [00:27:00:00, 1.37s/it]
===== Serving Benchmark Result =====
Successful requests:          20
Benchmark duration (s):      27.37
Total input tokens:          2500
Total generated tokens:      7910
Request throughput (req/s):   0.73
Output token throughput (tok/s): 289.03
Total Token throughput (tok/s): 382.57
-----Time to First Token-----
Mean TTFT (ms):              248.81
Median TTFT (ms):            244.84
P99 TTFT (ms):               261.74
-----Time per Output Token (excl. 1st token)-----
Mean TPOT (ms):              15.45
Median TPOT (ms):            15.56
P99 TPOT (ms):               15.94
-----Inter-token Latency-----
Mean ITL (ms):               15.05
Median ITL (ms):             15.14
P99 ITL (ms):                67.92
=====

```

The meanings of the metrics are as follows:

Metric Name	Metric Meaning
Successful requests	Total number of completed requests
Benchmark duration (s)	Total time taken to complete all requests
Total input tokens	Number of input tokens in all requests
Total generated tokens	Number of output tokens in all requests
Request throughput (req/s)	Request throughput
Output token throughput (tok/s)	Generated result throughput
Time to First Token (ms)	Time for generating the first token
Time per Output Token (ms)	Time between each token generation
Inter-token Latency (ms)	Time between each token generation

For a comparison of inference acceleration results on actual data sets, see [Inference Acceleration Performance Test Results](#).

FAQs

Question: What are the limitations on using Angel-vLLM's parallel decoding capability?

Answer: Parallel decoding conflicts with vLLM's `--enable-chunked-prefill` feature. By default, vLLM automatically enables `--enable-chunked-prefill` when `--max-model-len` exceeds 32 KB. To use lookahead parallel decoding acceleration in this scenario, manually disable it by adding `--enable-chunked-prefill false` to the startup command. Additionally, parallel decoding is also incompatible if vLLM is started with the `--num-scheduler-steps` parameter set to a value greater than 1.

Question: What are the differences between Angel-vLLM's quantization capability and open-source GPTQ and AWQ quantization?

Answer: Angel-vLLM's INT8, NF4, and FP8 quantization all support one-click online quantization, without requiring offline calibration with a data set. Compared to other open-

source quantization, it offers faster implementations. Among these, INT8 and FP8 achieve near-lossless precision, while NF4 quantization yields better precision than INT4 in experimental results. Open-source GPTQ and AWQ are offline calibration algorithms, primarily used for INT4 and INT8 quantization. Angel-vLLM does not support using ifq, ifq_nf4, or FP8 quantization methods to deploy models that have been calibrated by GPTQ or AWQ. For such models, use vLLM's native quantization methods, such as gptq, gptq_marlin, awq, and awq_marlin.

Note:

To run inference on the GPTQ or AWQ quantized model of qwen2 or qwen2.5, add the environment variable `VLLM_ALLOW_QUANT_LM_HEAD=0`.

Question: Why does the GPU memory utilization reach 90% when I deploy a 7B model on a GPU with 40 G of memory?

Answer: The vLLM framework pre-allocates GPU memory. Specifically, it uses `GPU memory x gpu_memory_utilization - peak GPU memory consumption for a single inference (including model weights + KV cache)`. The remaining GPU memory is allocated to GPU KV cache blocks to increase the batch processing size supported by the service. To estimate the minimum GPU memory required for model weights and KV cache, see [Inference Resource Requirements](#).

Question: How to enable the function call capability?

Answer: To enable tool call, add the parameters `--enable-auto-tool-choice` and `--tool-call-parser <parser_name>`. For usage, see [vLLM official documentation](#). Alternatively, you can set `MODEL_ID` to the model's name on Hugging Face. For models supporting tool calls, these parameters are enabled by default. For details, see [Chat Templates](#).

Appendix

Inference Acceleration Performance Test Results

FP8 Quantization Acceleration Test

Test environment:

Configuration Item	Configuration Details
GPU type	PNV5b
Test model name	Qwen2-7B-Instruct
Test data set	ShareGPT_V3_unfiltered_cleaned_split.json
Test code	vllm/benchmark
Test item	Comparison between no acceleration and

FP8 acceleration

Test script:

```
python benchmark_serving.py \
  --port xxxx \
  --base-url http://xxxxxxxx/ms-rpbpr92c/ \
  --backend openai-chat \
  --model ../Qwen2-7B-Instruct \
  --dataset-name sharegpt \
  --endpoint /v1/chat/completions \
  --tokenizer ../Qwen2-7B-Instruct \
  --max-concurrency 10 \
  --save-result \
  --result-dir results \
  --dataset-path /path/to/ShareGPT_V3_unfiltered_cleaned_split.json
```

Test results based on Angel-vLLM inference acceleration (concurrency=10, all other test conditions consistent):

Monitoring Metric	No Angel-vLLM Feature	With FP8 Quantization	Performance Improvement
Successful requests	1000	1000	-
Total input tokens	217393	217393	-
Request throughput (req/s)	0.68	1.08	+58.82%
Output token throughput (tok/s)	383.75	597.04	+55.58%
Total Token throughput (tok/s)	531.74	832.25	+56.51%
Mean TTFT (ms)	113.27	86.90	+23.28%
Median TTFT (ms)	76.52	55.64	+27.29%
P99 TTFT (ms)	874.30	708.10	+19.01%
Mean TPOT (ms)	24.33	16.56	+31.94%

Median TPOT (ms)	24.17	16.46	+31.90%
P99 TPOT (ms)	27.34	18.68	+31.68%
Mean ITL (ms)	24.22	16.52	+31.79%
Median ITL (ms)	23.45	15.83	+32.49%
P99 ITL (ms)	69.92	50.20	+28.20%

Lookahead Parallel Decoding Acceleration Performance

Since the acceleration principle of lookahead parallel decoding is that tokens that have been encountered by the service will be accelerated, generally, the acceleration effect improves as the number of sessions increases. The figure below shows an intuitive comparison of the generation speed for the second identical request with the default parameters versus with lookahead parallel decoding enabled.



Chat Templates

The chat template is used to convert user-input chat into `prompt` text for the LLM. For details, see the introductions on Hugging Face.

Chat templates are generally described using Jinja templates. For new open-source chat models (typically with Instruct or Chat suffixes), you can usually find the `chat_template` field in the `tokenizer_config.json` file or the `chat_template.json` file within the model directory. This field represents the chat template for the model. If no special settings are configured, this chat template will be used by default.

To facilitate the deployment of models that do not come with a `chat_template` field, and to enable function call capabilities for models that support it, if you specify the `MODEL_ID` or `CONV_TEMPLATE` environment variables, or if the model directory contains a `ti_model_config.json` file, and the image is started with the default startup command, the inference framework will automatically match the model's chat template according to the following rules and order (where the matching rules for `CONV_TEMPLATE` and `MODEL_ID` are in an OR relationship).

Model Series	CONV_TEMPLATE	MODEL_ID (Case-Insensitive)	Default Added Startup Command	Default Additional stop_tokens
Non-chat model	generate	-	<code>--chat-template examples/template_generate.jinja</code>	-
Hunyuan Large	hunyuan	Contains hunyuan-large.	<code>--chat-template examples/template_hunyuan.jinja</code>	[127960, 127967]
Tencent industry LLMs	shennong_chat	Contains shennong or sn-.	<code>--chat-template examples/template_shennong.jinja</code>	-
Llama-3.2 Vision Instruct model	llama-3.2-vision	Contains llama-3.2, with vision and instruct or chat.	<code>--chat-template examples/tool_chat_template_llama3.2_vision.jinja --enable-auto-tool-choice --tool-call-parser llama3_json --enforce-eager --max-num-seqs 8</code>	-
Llama-3.2 Instruct model	llama-3.2	Contains llama-3.2 and instruct or chat; does not contain vision.	<code>--chat-template examples/tool_chat_template_llama3.2_json.jinja --enable-auto-tool-choice --tool-call-parser llama3_json</code>	-
Llama-3.1 Instruct model	llama-3.1	Contains llama-3.1 and instruct or chat.	<code>--chat-template examples/tool_chat_template_llama3.1_json.jinja --enable-auto-tool-choice --tool-call-parser llama3_json</code>	-

Qwen2.5 Instruct model	qwen2.5	Contains qwen2.5 and instruct or chat.	<code>--enable-auto-tool-choice</code> <code>--tool-call-parser hermes</code>	-
Baichuan 2 Chat model	baichuan2-chat	Contains baichuan2 and instruct or chat.	<code>--chat-template examples/template_baichuan.jinja</code>	-
Llama2 Chat model	llama-2	Contains llama-2 and instruct or chat.	<code>--chat-template examples/template_llama2.jinja</code>	-
Llama3 Chat model	llama-3	Contains llama-3 and instruct or chat.	<code>--chat-template examples/template_llama3.jinja</code>	[128001, 128009]
Qwen Chat model	qwen or qwen-7b-chat	Contains qwen and instruct or chat; does not contain vl.	<code>--chat-template examples/template_qwen.jinja</code>	[151643, 151644, 151645]

Inference Resource Requirements

Inference resource requirements primarily include CPU, memory, and GPU. We mainly calculate the required computing resources based on GPU memory. It is recommended that CPU and memory be allocated proportionally according to the instance type and the number of GPU cards, with memory larger than the storage space occupied by model weights.

LLM inference demands high GPU memory. We can calculate the required GPU memory size and the minimum number of GPUs required for running LLM inference using this image through the following methods:

```
GPU memory required for model weights (G) ≈ Model weight parameter scale
(B) × Model weight precision (byte)
GPU memory required for KV cache (G) = Model layer size (hidden_size) ×
Number of model layers (num_hidden_layers) × Number of model KV heads
(num_key_value_heads) / Number of model attention heads
(num_attention_heads) × Model context length (max_position_embeddings) ×
Model KV-cache precision (byte) × 2/1024/1024/1024
```

```

Required GPU memory > (GPU memory required for model weights + GPU memory
required for KV cache)/GPU memory utilization (gpu_memory_utilization)
----
Minimum number of GPUs required = ceil (Required GPU memory/GPU memory per
GPU card)
With the requirement num_attention_heads % number of GPUs == 0

```

Generally, the model weight precision and KV cache precision are float16 or bfloat16 (that is, 2 bytes) before quantization. If INT8 or FP8 quantization is applied, they become 1 byte; if INT4 or NF4 quantization is applied, they become 0.5 bytes. Other key parameters can be found in the `onfig.json` file within the model directory.

Example:

Open-Source Model Name	Meta-Llama-3.1-8B-Instruct	Qwen2-72B-Instruct
Model Weight Parameter Scale (Billion)	8	72
Model Weight Precision (Byte) (Adjustable by Enabling <code>--quantization</code>)	2	2
Total GPU Memory Occupied by Model Weights (G)	16	144
Model Layer Size (hidden_size)	4096	8192
Number of Model Layers (num_hidden_layers)	32	80
Number of Model KV Heads (num_key_value_heads)	8	8
Number of Model Attention Heads (num_attention_heads)	32	64
Model Context Length (max_position_embeddings) (Adjustable via the <code>--max-model-len</code> Parameter)	131072	32768

Model KV Cache Precision (Byte)	2	2
Total GPU Memory Occupied by Model KV Cache (G)	16	10
GPU Memory Utilization (Adjustable via the --gpu-memory-utilization Parameter)	0.9	0.9
Minimum GPU Memory Required (G)	35.56	171.11
Minimum Number of GPUs with 24 G GPU Memory	2	8
Minimum Number of GPUs with 40 G GPU Memory	1	8
Minimum Number of GPUs with 48 G GPU Memory	1	4

LLM Training and Evaluation

End-to-End Use Cases: Using DeepSeek-R1 to Distill Other Models

Last updated: 2026-03-12 10:28:53

Overview

[DeepSeek-R1](#) provides exceptional inference capabilities. However, its huge model size (671B) requires significant inference resources. To address this, we can transfer DeepSeek-R1's inference capabilities to smaller open-source models (such as [Llama-3.2-3B](#)) through model distillation, thereby reducing inference costs while maintaining performance.

Model distillation is a technique that transfers knowledge from a complex model (the teacher model) to a simpler model (the student model), aiming to compress model size, increase inference speed, and preserve performance as much as possible. This document walks you through completing model distillation using the existing product capabilities of Tencent Cloud TI-ONE Platform (TI-ONE). The overall steps are as follows:

1. Step 1: Deploying a DeepSeek-R1 Inference Service
2. Step 2: Distilling Data Based on DeepSeek-R1
3. Step 3: Fine-Tuning the Student Model Using the Distillation Data Set
4. Step 4: Evaluating the Distillation Effectiveness of the Student Model

The outcome of this practice is that you will obtain a Llama-3.2-3B-distill model with chain of thought (CoT) inference capabilities and achieve a measurable improvement in evaluation performance on the GSM8K data set (the objective evaluation accuracy increases from 70% to 77%).

In the following text, DeepSeek-R1 is referred to as the teacher model, Llama-3.2-3B as the student model, the data with long CoT generated by DeepSeek-R1 inference is called the distillation data set, and the official data set is the raw data set.

Prerequisites

Preparation of Computing and Storage Resources

1. Resources required for teacher model inference: Deploying the teacher model requires at least 1 PNV6 instance with 8 GPU cards. For higher inference concurrency, it is recommended to deploy the inference service in a distributed manner using 2 HCCPNV6 instances with 8 GPU cards. For more information about deployment resources, see [Guide on Resources Required by LLM Inference](#). Therefore, before you start the practice, it is

recommended to manage at least 2 HCCPNV6 instances with 8 GPU cards. In this mode, you need to purchase Cloud Virtual Machine (CVM) instances in advance and add them to a TI-ONE resource group. For specific costs, see [Model List and Resource/Price Reference](#).

2. Resources required for fine-tuning the student model: In this practice scenario, fine-tuning the Llama-3.2-3b model requires the HCCPNV6 instance type with at least 4 GPU cards.
3. To improve training speed, it is recommended to use [Cloud File Storage \(CFS\) Turbo](#). For more information about how to activate the CFS service, see [Guide on Using CFS Instances in Various Scenarios](#).

Material Preparations

Models and Images

The teacher model (DeepSeek-R1), the student model (Llama-3.2-3B-Chat), and the images for running these models are all built into the platform. You do not need to download them separately.

Data Sets

The data set used for distillation in this document is [OpenO1-SFT](#). Before executing subsequent steps, you need to download this data set to your local machine.

The data set used for evaluating distillation effectiveness is GSM8K. The platform has uploaded this data set to Cloud Object Storage (COS) for you, and you can [download it directly](#).

Code

The code involved in this practice mainly includes the script for calling the teacher model to generate inference results and scripts related to data set preprocessing. You need to download the teacher model calling script in advance through the following link:

[generate_reasoning_streams.py](#).

The data set preprocessing script can be copied directly during the practice steps below.

Detailed Steps

Step 1: Deploying a DeepSeek-R1 Inference Service

To obtain the inference results of DeepSeek-R1 for subsequent distillation, first, you need to deploy an inference service on TI-ONE. You can refer to [Quick Deployment and Tryout of DeepSeek Series Models](#) to complete the deployment of DeepSeek-R1. Note: Generating the distillation data set requires high concurrency. Therefore, ensure that you use the multi-machine distributed deployment mode (it is recommended to use a 2-machine HCCPNV6 deployment).

After deployment is complete, you can verify that the service deployment is successful through the Online Experience feature of the Online Services module.

Step 2: Distilling Data Based on DeepSeek-R1

This step demonstrates the end-to-end process of distilling data based on the teacher model (DeepSeek-R1) through the Data Preprocessing module of TI-ONE. The main workflow is as follows:

1. Create a data preprocessing task
2. Call and verify the teacher model inference service
3. Upload and clean the raw data set
4. Call DeepSeek-R1 to generate the distillation data set
5. Clean the distillation data set

2.1. Creating a Data Preprocessing Task

The Data Preprocessing module of TI-ONE has built-in common data preprocessing code, making it convenient for you to perform subsequent tasks such as data cleaning. Therefore, we use the data preprocessing feature of TI-ONE for data preparation. For more information about data preprocessing, see [Preprocessing LLM Fine-Tuning Data](#).

1. Go to the Data Preprocessing module and create a data preprocessing task.
2. Create a data preprocessing dev machine. Note that the data preprocessing dev machine is only used to call model services and data preprocessing. Therefore, it does not require GPU computing resources. In this practice, it is recommended to use 8C16G resources. Additionally, to use the Virtual Private Cloud (VPC) high-speed calling address when you call the inference service later, it is recommended to use the same resource group as the teacher model inference service.

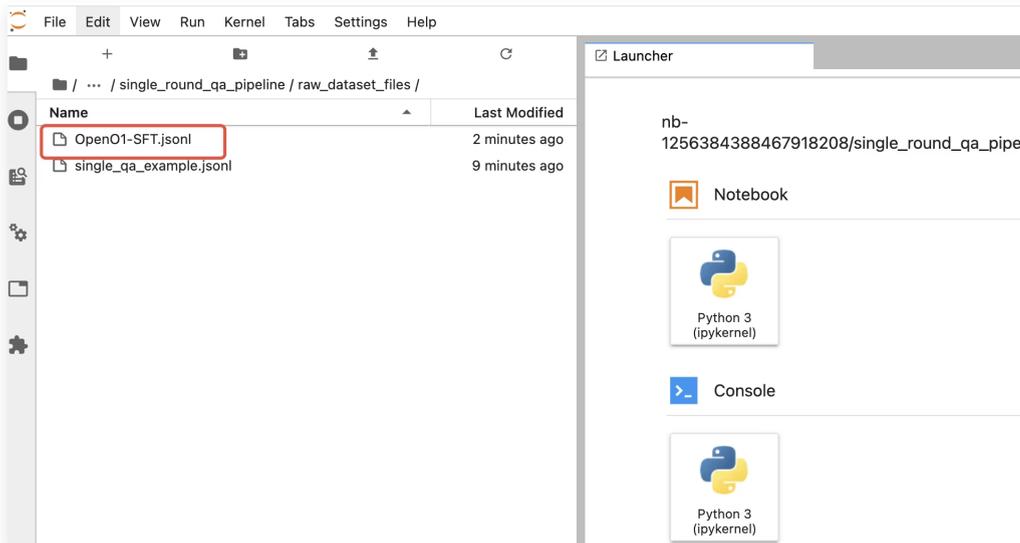
2.2. Calling and Verifying the Teacher Model Inference Service

After the dev machine starts up, you can go to the terminal and run the following command to verify whether the call chain to the deployed teacher model inference service is working. Replace <svc-ip>, <ms-id>, and <auth_token> with the correct values, as shown in the following figure. It is recommended to use the VPC calling address to initiate online service calls. This avoids issues such as result filtering caused by the Web Application Firewall (WAF) gateway.

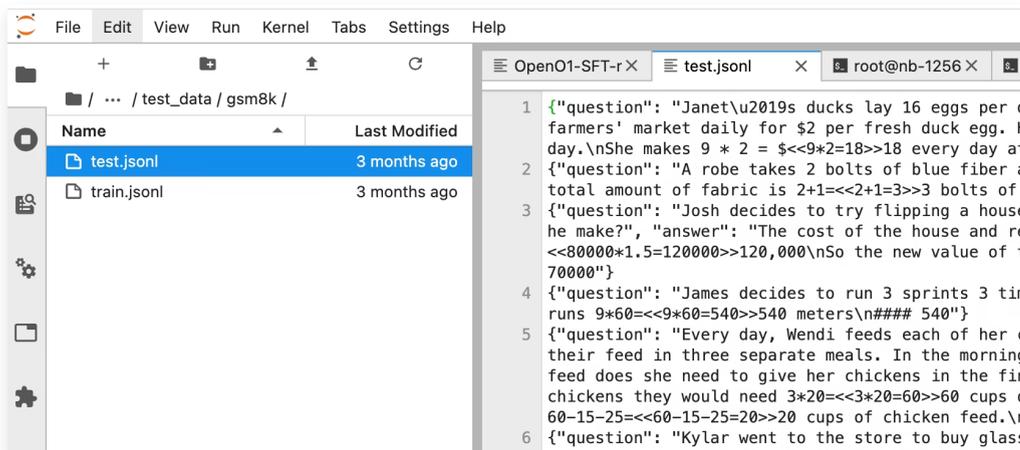
```
curl -X POST http://<svc-ip>/<ms-id>/v1/chat/completions -H
'Authorization: <auth_token>' -H 'Content-Type: application/json' -d'{
"model": "<ms-id>", "messages": [ { "role": "user", "content": "Describe
your understanding of AI." } ] }'
```

2.3. Uploading and Cleaning the Raw Data Set

1. Upload the raw data. Upload the locally downloaded OpenO1-SFT data (OpenO1-SFT.jsonl) to the following path on the dev machine: `/home/tione/notebook/<your-nb-id, this directory is automatically generated by the platform>/single_round_qa_pipeline/raw_dataset_files/`, as shown in the following figure. This data set is about 380 MB. Data uploading on the Dev Machines page takes about 1 minute. You can also use Tencent Cloud Object Storage (COS) for transfer.



Similarly, you need to upload the GSM8K data to the following path on the dev machine: `/home/tione/notebook/<your-nb-id, this directory is automatically generated by the platform>/test_data/gsm8k` (using the test.jsonl data from the unzipped folder).



Our fine-tuning task in this practice aims to enhance the student model's mathematical computation capabilities. The OpenO1SFT data set contains a significant amount of data irrelevant to our fine-tuning scenario (including code generation and open-ended Q&A). Including this data in the distillation data set would occupy numerous calls to the teacher model inference service, thereby increasing distillation costs. Therefore, before actually making the calls, we need to remove this portion of data from the data set, retaining only the data relevant to the distillation scenario (mathematical computation). Next, we proceed to perform data preprocessing on the created data preprocessing dev machine.

2. Perform data sampling and parsing.

In the `quick_start.ipynb` file, modify the file name of the raw data set, and perform data sampling. You can then view the data set information.

Parse the raw data. Note that the custom function `convert_dict_to_tione_format` requires specifying the corresponding fields according to the data set. You can view the data distribution.

3. Clean the data.

Since our primary goal is to enhance model performance in mathematical scenarios, we focus on filtering questions related to mathematics and inference. Custom cleaning rules can be defined during data cleaning. Analysis of the raw data reveals a significant number of coding-related prompts and general Q&A content, which we aim to filter out through this cleaning process.

First, disable all system default cleaning rules through the `config/clean_config.yaml` configuration. Then, write custom cleaning rules.

```
import re
from typing import Union, Dict

class CodeChecker:
    def __init__(self):
        # Define common programming language names (can be extended).
        self.languages = ["c++", "python", "javascript", "java", "c#",
            "typescript", "ruby", "php", "program", "tensorflow", "API", "json",
            "function", "algorithm", "script", "linux", "Kubernetete"]

    def is_code_like(self, text: str) -> bool:
        # Convert text to lowercase for case insensitivity.
        lower_text = text.lower()

        # Check whether the text contains any programming language
names.
        for lang in self.languages:
            if lang in lower_text:
                return True
        return False

# Register the filter function.
@filter_func.FilterFunctionManager.register(zh_name="Contain common
programming names.")
def check_code(text: str, config: Dict = None) -> Union[bool, str]:
```

```
checker = CodeChecker()
if checker.is_code_like(text):
    return True, "Contain common programming names."
return False, ""

# Register the filter function.
@filter_func.FilterFunctionManager.register(zh_name="Do not contain more
than 5 digits.")
def filter_three_or_more_digits(text: str, config: Dict = None) ->
Union[bool, str]:
    # Count the number of digits in the text.
    num_count = len(re.findall(r'\d', text))

    # If the number of digits is less than or equal to 5, filter it out.
    if num_count <= 5:
        return True, "Do not contain more than 5 digits."
    return False, ""
```

As can be seen, after cleaning, we obtain 28,000 data points and save them to the following path: `/home/tione/notebook/<nb-id>/single_round_qa_pipeline/intermediate_dataset_files/02_Open01-SFT_cleaned.jsonl` . The following is a data example:

```
{"system": "You are the intelligent question-answering robot on the
Tencent Cloud TI platform, capable of understanding user questions and
providing fluent answers.", "question": "Anna goes trick-or-treating in
a subdivision where she gets 14 pieces of candy per house. However, for
every fifth house, she gets 2 mini chocolate bars in addition to the
regular candy. Her brother Billy goes trick-or-tricking in a neighboring
subdivision where he gets 11 pieces of candy per house. If Anna visits
75 houses and Billy visits 90 houses, how many more pieces of candy does
Anna get?", "answer": "<Thought>\nAlright, let's tackle this problem
about Anna and Billy trick-or-treating. So, Anna is going to a
subdivision where she gets 14 pieces of candy per house. But wait,
there's a twist-every fifth house she visits, she gets 2 extra mini
chocolate bars in addition to the regular candy. That's interesting. Her
brother Billy, on the other hand, is going to a neighboring subdivision
where he consistently gets 11 pieces of candy per house. The question is
asking how many more pieces of candy Anna gets compared to Billy after
visiting their respective number of houses: Anna visits 75 houses, and
```

Billy visits 90 houses.

First, let's break down Anna's candy collection. Normally, she gets 14 pieces per house, so if she visits 75 houses, the straightforward calculation would be 14 pieces multiplied by 75 houses. Let me do that: 14 times 75 equals 1050 pieces of candy. So, without considering the extra mini chocolate bars, Anna would have 1050 pieces.

But wait, there's more to the story. Every fifth house, she gets 2 extra mini chocolate bars. How many times does she hit every fifth house in her 75-house adventure? Well, 75 divided by 5 equals 15. That means she gets those extra 2 pieces of candy 15 times. So, 15 occasions multiplied by 2 extra pieces equal 30 extra pieces of candy. Adding that to her initial total, 1050 plus 30 equals 1080 pieces of candy. Anna ends up with 1080 pieces.

Now, let's look at Billy. He gets 11 pieces of candy per house, and he visits 90 houses. So, 11 multiplied by 90 houses equals 990 pieces of candy for Billy.

The question is asking how many more pieces of candy Anna gets compared to Billy. So, I need to subtract Billy's total from Anna's total. That would be 1080 pieces (Anna) minus 990 pieces (Billy), which equals 90 pieces. Therefore, Anna gets 90 more pieces of candy than Billy.

But hold on a second—let me double-check to make sure I didn't make any mistakes. For Anna: 14 pieces per house times 75 houses is indeed 1050, plus 30 extra pieces for the 15 houses where she gets bonus candy equals 1080. For Billy: 11 pieces per house times 90 houses is definitely 990. Subtracting 990 from 1080 does give us 90. Yes, that seems correct.

Is there another way to approach this? Maybe by calculating the average per house including the extras for Anna. Let's see: Anna gets an extra 2 pieces every 5 houses, so on average, that's an extra 0.4 pieces per house (since 2 divided by 5 is 0.4). So, her average per house would be 14 plus 0.4, which is 14.4 pieces per house. Multiplying that by 75 houses: 14.4 times 75 equals 1080. Yep, that matches the previous calculation.

Alternatively, could there be a way to visualize this? If I imagine walking down a street with 75 houses, every fifth house has an extra candy incentive. Counting those extra pieces could be another way to confirm the total. But I've already done the math, and it seems consistent.

Another angle: What if I considered the total number of extra pieces Anna gets compared to Billy? Since Anna gets 2 extra pieces every 5 houses, and she visits 75 houses, that's 15 times 2 extra pieces, which is 30 extra pieces. Billy doesn't get any extra pieces, so all of Anna's advantage comes from that 30-piece difference, plus the difference in their per-house candy rates.

Comparing their per-house rates: Anna gets 14 per house, Billy gets 11. That's a difference of 3 pieces per house. Over 75 houses,

```
that's 225 extra pieces from the per-house rate alone. Wait a minute,
that doesn't match my earlier total difference. Did I make a
miscalculation here?
Ah, I see the mistake. Billy visits 90 houses,
not 75. So, for a fair comparison, Anna's 75 houses at 14 per house
equals 1050, and with the extra 30 pieces, that's 1080. Billy's 90
houses at 11 per house is 990. The difference is still 90 pieces. So, my
initial calculation stands.
In conclusion, after visiting their
respective numbers of houses, Anna ends up with 90 more pieces of candy
than Billy.
Anna gets 90 more pieces of candy
than Billy.
```

Next, we will call the R1 model to generate CoT results for these 28,000 data entries.

2.4. Calling DeepSeek-R1 to Generate the Distillation Data Set

First, save the cleaned data to a separate directory for the distillation data set generation:

```
cd /home/tione/notebook/<nb-id>
mkdir -p deepseek_r1
cp single_round_qa_pipeline/intermediate_dataset_files/02_Open01-
SFT_cleaned.jsonl deepseek_r1/Open01-SFT-math.jsonl
```

Next, call the data generation script to obtain R1 inference results. We provide a script adapted for TI-ONE based on the [official script](#) of Hugging Face. You can directly [download](#) the script to your dev machine. Using this script, you only need to specify the data set path, and it will automatically call the teacher model inference service to generate the distillation data set (saved to a JSONL file).

```
pip3 install datasets aiofiles uvloop
python3 generate_reasoning_stream.py -i deepseek_r1/Open01-SFT-math.jsonl
-o deepseek_r1/Open01-SFT-math-generate.jsonl --prompt-column question --
max-concurrent 100 --model-addr http://<svc-ip>/<ms-id> --model-token
<ms-token> --model-service <ms-id>
```

- `prompt-column` is the actual field name in the data set that contains the prompt. Here, the cleaned data field is named `question`.
- The `max-concurrent` parameter specifies the maximum concurrency. The R1 service deployed on 2 machines can support up to 128 concurrent calls with relatively high throughput. It is recommended to first set it to 1 for testing, and then change it to 100 for actual execution.

to generate a qualified data set. Note: Determining the correctness of R1 inference results requires complex logic (such as using MathVerify). This document only demonstrates simple data cleaning based on rules (for example, the data is too long, and the CoT is incomplete).

1. Data analysis

We copy the generated file `OpenO1-SFT-math-generate.jsonl` to the `raw_dataset_files` directory for data preprocessing.

We specify this file as the input and execute it again. A total of 25,000 data entries are generated.

Next, we perform token distribution analysis. At the same time, we download the DeepSeek tokenizer to the notebook and extract it to `/home/tione/notebook/<nb-id>/tokenizer`.

```
cd /home/tione/notebook/<nb-id>/single_round_qa_pipeline/  
wget https://tione-public-cos-1308945662.cos.ap-  
shanghai.myqcloud.com/tools/deepseek_tools/deepseek_tokenizer.tgz  
tar xf deepseek_tokenizer.tgz
```

Modify the data parsing code, as shown in the following figure.

```
# Parse the data.  
  
# Output path for parsed data, in JSONL format. It is recommended to use  
the intermediate_dataset_files folder for unified management.  
step1_out_file = os.path.join(  
    "intermediate_dataset_files",  
    "01_{}_parsed.jsonl".format(dataset_name)  
)  
  
import random  
# You can override this function to parse different JSON formats. For  
your input file, we will parse it into multiple lines of JSON and  
convert the JSON format to Python dict format.  
# In this function, you can convert the dict of the sampled JSON schema  
to the format specified by TI-ONE: {"system": "xxxx", "question":  
"xxxx", "answer": "xxxx"}.  
def convert_dict_to_tione_format(item: dict) -> dict:  
    systems = ["You are an AI assistant that helps solve mathematical  
problems.",  
              "You will be given a problem. Please reason step by step,  
and put your final answer within \\boxed{{}}:\\n"]  
    try:
```

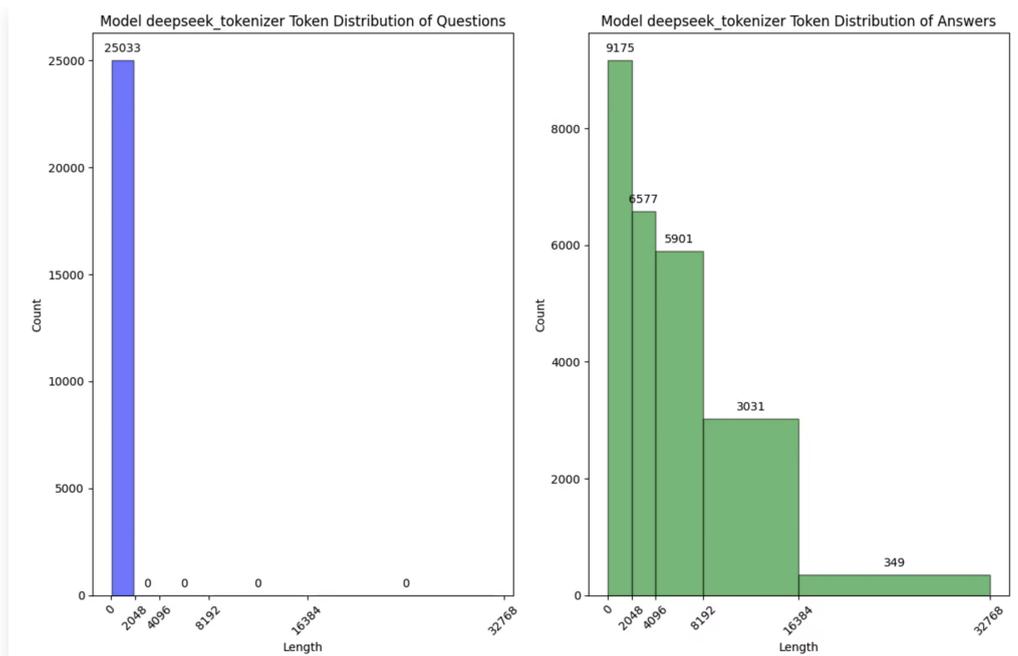
```
question = item["question"]
answer = item["generations"][0]
# Sample construction: Obtain the system role field system. The
specific content can be modified as needed.
# Construct the sample format required by TI-ONE.
convert_dict = {
    "system": random.choice(systems),
    "question": question,
    "answer": answer,
}
except:
    # Data parsing exception.
    return None
return convert_dict

# Use the function defined in step1_parse_raw_data.py in the same
directory to parse the raw data.
step1_parse_raw_data.parse_raw_data_single_round_qa(step1_in_file,
step1_out_file, record_convert_fn=convert_dict_to_tione_format)

# The tokenizer_model_path parameter specifies the local model path for
token calculation. The default value is python/len, which means not
using a large model and directly calculating the string length.
tokenizer_model_path = "./deepseek_tokenizer"

# After parsing is completed, perform raw data statistics on the output
file from Step 1.
# The num_bins parameter specifies the number of buckets for the
distribution histogram. When num_bins is set to 0, the system's built-in
buckets are used. The default value is 10.
analysis_raw_data.statistic_raw_data_single_round_qa(step1_out_file,
num_bins = 0, tokenizer_model_path = tokenizer_model_path)
```

From the output of the data parsing step, we can clearly observe the distribution of token lengths in the data.



2. Data cleaning

To improve training speed, we will filter out inference results with token lengths exceeding 4,000. Additionally, to ensure data quality, we will only retain inference results with a complete CoT. For this purpose, we define and execute the following custom filtering functions and execute cleaning:

```

from tools import data_tokenizer
from typing import Union, Dict
tokenizer = data_tokenizer.Tokenizer("./deepseek_tokenizer")
length_threshold = 4096

# Register the filter function.
@filter_func.FilterFunctionManager.register(zh_name="Token length
filtering.")
def check_token_len(text: str, config: Dict = None) -> Union[bool, str]:
    text_len = tokenizer.tokenize_len(text)
    if text_len > length_threshold :
        return True, "Length shorter than {}".format(length_threshold)
    return False, ""

# Register the filter function.
@filter_func.FilterFunctionManager.register(zh_name="Check CoT
completeness.")
def check_think_tag(text: str, config: Dict = None) -> Union[bool, str]:
    if "<think>" not in text or "</think>" not in text:
        return True, "Incomplete CoT."

```

```
return False, ""
```

Run the data cleaning code:

After processing, 15,000 data entries remain.

3. Deduplication

Directly reuse the deduplication feature in data preprocessing. By modifying the following parameters, we can remove samples with high similarity.

4. Converting to training format

The input data format for the built-in LLM in TI-ONE is as follows:

```
{
  "system": "xxx",
  "conversation": [
    {
      "prompt": "xxx",
      "response": "xxx"
    }
  ]
}
```

To obtain data in this format, we can perform the final step of data preprocessing: training format generation, which produces the final data for distilling a small model.

At this point, we have completed the construction of the distillation data set.

Note: Data quality has a significant impact on the effectiveness of fine-tuning. This document only demonstrates simple data preprocessing logic. When you find that the fine-tuning results do not meet expectations, you may need to write more complex data processing logic to improve data quality.

Step 3: Fine-Tuning the Student Model Using the Distillation Data Set

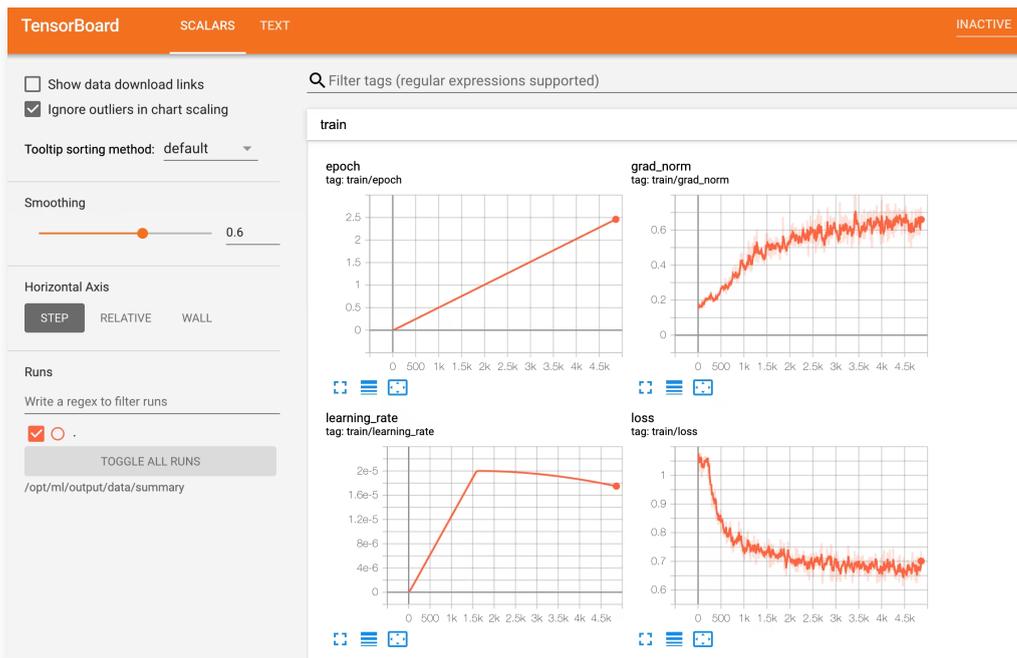
After obtaining the distillation data set, you can distill the student model (using fine-tuning of the built-in Llama-3.2-3B model as an example). You can initiate a fine-tuning task for Llama-3.2-3B-Chat by performing the following steps. For detailed process, see [Fine-Tuning Built-in Open-Source LLMs](#).

1. On the details page of the newly created task, select the built-in Llama-3.2-3B-Chat model and specify the required training resources. It is recommended to select training resources with at least 4 GPU cards. Following the equal division of CPU and memory for a CVM instance with 8 GPU cards, the recommended configuration is 190 cores and 1,107 GB of

memory. If sufficient computing resources are available, you can specify more to accelerate training speed.

On the details page of the newly created task, delete the built-in data and replace it with your business data. Additionally, because DeepSeek-R1's results with CoT are long, you also need to adjust hyperparameters to match your distillation data set. During data cleaning, filtering is based on the answer field. However, during actual training, filtering is based on the question and answer fields. Therefore, the MaxSequenceLength specified in this document is 8,000, not the 4,000 threshold used during cleaning. Also, correctly configure your model output path so that subsequent evaluation tasks can be created to verify the effectiveness of the trained model (this document assumes that the executed model is saved to `cfs:///demo/deepseek/train/distill`). The hyperparameters configured during fine-tuning in this document are as follows. For more information about hyperparameters, see [Fine-Tuning Built-in Open-Source LLMs](#).

After the task starts running, you can monitor the training process by viewing monitoring metrics or logging in to the training instance container. You can also click TensorBoard on the page to automatically open the metric monitoring panel and observe the convergence of loss, as shown in the following figure.



During task execution, the platform automatically saves the fine-tuned model checkpoints in the configured CFS output path. At this point, you can directly create a quick test task on the Checkpoint page to experience the effect of the distilled model. For objective evaluation, the recommended resource configuration is a single GPU card with 47 cores and 276 GB of memory.

After the quick test service is running, click Experience to ask the model questions. You will observe that the distilled model checkpoints have begun to acquire the capability to think and output long CoT.

Step 4: Evaluating the Distillation Effectiveness of the Student Model

In Step 3, we can observe that the trained student model checkpoints have gradually acquired long CoT thinking ability. We further conduct an objective evaluation to score the model before and after distillation to observe the effectiveness of fine-tuning.

In the Model Evaluation module of TI-ONE, create an objective evaluation task. For the custom evaluation set address, specify the GSM8K data set uploaded in Step 2.3. For the models to be evaluated, select both the built-in Llama-3.2_3B_Chat model before distillation and the model generated after this fine-tuning for batch evaluation.

In the inference hyperparameter settings, configure the parameters as follows:

```
{
  "temperature": 0.6,
  "top_p": 0.95,
  "frequency_penalty": 0,
  "seed": 12345
}
```

After the evaluation is completed, you can obtain the scoring metrics of the models before and after distillation on the specified data set. We can observe that the distilled model demonstrates significant improvement in metrics compared with the pre-distillation model on this evaluation set. Additionally, combined with the chat results from the quick test above, the model has also developed the capability to output long CoT.

End-to-End Use Cases: Fine-Tuning DeepSeek-R1

Last updated: 2026-03-12 10:32:28

Overview

Tencent Cloud TI-ONE Platform (TI-ONE) has built-in full series of DeepSeek models, including V3 and R1 models, and six small models distilled based on DeepSeek-R1. You can click New Task in the [Task-based Modeling](#) module of the platform and select the built-in LLMs in the training image to initiate fine-tuning of the full series of DeepSeek models.

This document introduces how to use TI-ONE to perform supervised fine-tuning of the DeepSeek-R1-671B model. The overall steps of this practice are as follows:

- Step 1: Upload a data set.
- Step 2: Start a fine-tuning task.
- Step 3: Perform model conversion.
- Step 4: Deploy a service.
- Step 5: Verify inference results.

The result of this practice is that you have changed the self-awareness of the model through fine-tuning.

Hereinafter, the DeepSeek-R1-671B model is referred to as the R1 model.

Prerequisites

Preparation of Computing and Storage Resources

1. Fine-tuning the R1 model requires GPU computing resources, and the recommended instance type is HCCPNV6. This document takes a full-parameter SFT of the model as an example, which requires at least 32 HCCPNV6 instances as the computing resources. Therefore, it is recommended to provide and manage at least 32 HCCPNV6 instances with 8 GPU cards before starting the practice. In this mode, you need to purchase Cloud Virtual Machine (CVM) instances in advance and add them to the TI-ONE resource group.
2. During model fine-tuning, the storage of both the data set and model checkpoints relies on Cloud File Storage (CFS). Enable CFS in the [CFS console](#). Given the large size of the R1 model files, Turbo series storage is recommended to ensure the speed of saving checkpoints during training, subsequent use of the model to deploy inference services, and starting evaluations. For differences between CFS instances, see [Storage Types and Specifications](#). Additionally, ensure network connectivity between the purchased CFS instance and the above-mentioned computing resources, that is, CVM instances. For detailed operations, see [Guide on Using CFS Instances in Various Scenarios](#).

Material Preparations

Models, Images, and Training Code

The model, image, and code involved in this R1 model fine-tuning practice are all built-in resources of the platform. You do not need to download them separately.

Data Sets

The example scenario for fine-tuning in this document is fine-tuning for the self-awareness of the model. The download address for the data set used for fine-tuning is as follows:

[identity.jsonl](#)

Detailed Steps

Step 1: Uploading a Data Set

Note: Currently, TI-ONE does not support direct data upload through the console. To resolve this issue, you need to create an Ops dev machine that mounts CFS instances and use the dev machine service to upload or download LLMs, training code, and other files.

Choose [Training Workshop > Dev Machines](#), and click **New** to create a dev machine.

- **Image:** Select any built-in image, as this dev machine is only used for downloading data sets.
- **CVM Instance Source:** Purchase from the TI-ONE platform or select from Cloud Virtual Machine (CVM) instances.
- **Billing Mode:** Select either pay-as-you-go or yearly/monthly subscription billing mode. For billing rules supported by TI-ONE, see [Billing Overview](#).
- **Storage Path Configuration:** Select CFS. The name should be the CFS instance you applied for and configured in the Prerequisites section. The default path is the root directory / and is used to specify the storage path for your training data.
- **Other Settings:** Not required by default.

Upload the downloaded data set to the dev machine.

Step 2: Starting a Fine-Tuning Task

1. Choose [Training Workshop > Task-based Modeling](#), click **New Task**, and fill in the task configuration information:
 - **Task Name:** Enter your custom task name.
 - **Training Image:** Select a built-in LLM/DeepSeek Series Model/DeepSeek-R1-671B.
 - **CVM Instance Source:** Select **Select from CVM Instances**.

- Resource Group: Select the resource group containing 32 nodes.
- Resource Application: Fully use the GPU, CPU, and memory resources of a node. For an HCCPNV6 instance, the entire instance can provide 380 CPU cores and 2,214 GB of memory. If the entire instance is idle, it is recommended to fill in 380 CPU cores and 2,214 GB of memory, as shown below.
- Tag: It can be filled in as needed and is not involved in this practice.
- Description: It can be filled in as needed and is not involved in this practice.
- CLS Log Shipping: It is disabled by default.
- Auto Restart: It is disabled by default.
- Storage Path Settings: The key parameters on this page are configured as follows:
 - Storage Path Settings: The default option is CFS (currently, the built-in code, data, and models of the platform only support CFS for LLM fine-tuning).
 - Select a storage path.
 - For the first line Platform CFS: By default, the system has configured the supporting training code for fine-tuning the LLM.
 - For the second line Platform CFS:
 - By default, the system has configured a set of sample data for fine-tuning the LLM.
 - In this case, custom business data is used to fine-tune the LLM. You can delete this line and add a line at the bottom, set the purpose to **User Self-owned Data**, and select the CFS instance where your data set is located and the corresponding source path. Assume that the path of your data set is <your CFS_id>/dataset.
 - The third line Platform CFS: By default, the system has configured a built-in model of the platform. (The R1 model has a very large size, and pulling it directly from the default shared storage of the platform is slow. To accelerate model pulling, you are advised to contact the TI-ONE team through your sales or pre-sales architect for manual cache acceleration support.)
 - The fourth line User CFS: Select the CFS instance where your training output is located and the corresponding source path. Assume that the output path you selected is <your CFS_id>/output.
- Code Package: No need to select (in LLM fine-tuning scenarios, you do not need to select a code package).
- Startup Command: It is automatically populated by the system by default and requires no modification (used to start the default supporting training code for the LLM).
- Training Output: No need to select.
- Tuning Parameters: They are automatically populated by the system by default. The hyperparameters filled in this case are as follows. In actual fine-tuning scenarios, you

can adjust parameters such as `train_iters`, `warmup_iters`, `seq_len`, and `pad_len` based on the amount of training data.

```
{
  "batch_size": "1",
  "global_batch_size": "128",
  "lr": "7e-6",
  "min_lr": "7e-7",
  "seq_len": "1024",
  "pad_len": "1024",
  "precision": "bf16",
  "save_interval": "200",
  "train_iters": "10",
  "warmup_iters": "1",
  "sft": "true",
  "sft_packing": "false",
  "tp_size": "1",
  "pp_size": "8",
  "cp_size": "1",
  "ep_size": "32",
  "enable_sp": "true",
  "enable_do": "true",
  "enable_fl": "false",
  "ac": "full",
  "optimizer_offload": "false"
}
```

The complete hyperparameter description is as follows:

ⓘ Hyperparameter description:

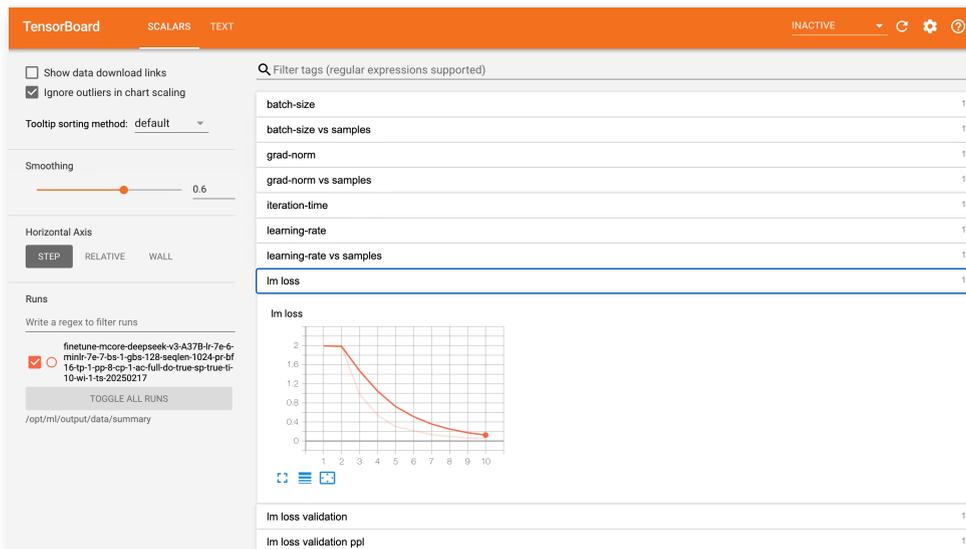
The hyperparameter list for fine-tuning the V3/R1 model is based on the Megatron framework and differs from the hyperparameter lists for other models. For the hyperparameter lists of other models, see [Preset Parameters for Tuning](#).

- `batch_size`: number of samples processed per iteration within one data parallel group.
- `global_batch_size`: total number of samples processed per iteration across multiple data parallel groups.
- `lr`: learning rate.

- `min_lr`: minimum learning rate.
- `seq_len`: sequence length.
- `pad_len`: padding length.
- `precision`: training precision: fp16, bf16, or fp8.
- `save_interval`: interval for saving checkpoints.
- `train_iters`: training iteration steps.
- `warmup_iters`: warm-up iteration steps.
- `sft`: whether to perform fine-tuning training: true or false.
- `sft_packing`: whether to enable data packing during SFT.
- `tp_size`: model parallelism, which can only be set to 1 currently.
- `pp_size`: pipeline parallelism.
- `cp_size`: context parallelism.
- `ep_size`: expert parallelism.
- `enable_sp`: whether to use sequence parallelism: true or false.
- `enable_do`: whether to use the Megatron version of Zero-1 GPU memory reduction optimizer: true or false.
- `enable_fl`: whether to prioritize using Flash Attention: true or false.
- `ac`: checkpoint activation mode: sel, full, offload, or false.
- `optimizer_offload`: whether to enable the Offload optimizer: false, static, or auto.

2. After the task configuration is completed, click **OK** to return to the task list. On this page, you can view the model training progress and training logs and monitor information such as training resource consumption and training metric convergence.
3. During the execution of a training task, you can click the name of the task to go to the Logs tab and monitor the training status. After the training task is started, it will go through the following stages:
 - **Data preprocessing**: The first node (master-0 selected from the node drop-down list) prints data preprocessing information.
 - **Starting training**: The last node (worker-30 node selected from the node drop-down list) prints training iteration steps, learning rate, lm loss, and other information. By default, training information logs will be printed every 1 step.

Click **Tensorboard** to view the training metric monitoring information.



After the training is completed, the model will be automatically saved in the output CFS path specified in the training configuration.

Step 3: Performing Model Conversion

Note:

The model conversion step is required for fine-tuning the V3/R1 model only, but **is not required for fine-tuning other models (such as R1 Distill series models)**. A trained model can be directly evaluated or used for inference.

TI-ONE uses the Megatron-LM-based training framework for R1 models, and checkpoints are saved in a non-Hugging Face format. Therefore, model format conversion is required before model inference. To simplify your operations, the platform has integrated the model format conversion steps into the built-in training code of Task-based Modeling. To complete the model conversion, you only need to copy the fine-tuning training task and modify partial parameters. The specific steps are as follows:

1. Copy the fine-tuning training task, change the value of the number of nodes to 1, and change the MODE variable in the startup command to mg2hf.
2. Keep other parameter settings unchanged. Click **OK** to run the task. The task will convert the last model output by the fine-tuning training task into the Hugging Face format. Save the model in the directory where the original checkpoint directory is located, and name it with the suffix `_hf`. If CFS is of turbo type, the expected model conversion time is 1 hour.

Step 4: Deploying a Service

Choose [Model Services > Online Services](#), and click **New Service**.

You can set Service Name to a custom name, set Deployment Mode to Multi-machine Distributed Deployment, set Model Source to CFS, and select the CFS instance where the model is stored. Set the path to the directory where the model in the Hugging Face format (after model

conversion) is located in CFS, and set Running Environment to the built-in LLM/sglang image. For resource configuration, use 2 HCCPNV6 instances with 16 GPU cards. To accelerate startup, add the environment variable `ENABLE_TORCH_COMPILE=0`.

Step 5: Verifying Inference Results

After the service is successfully deployed, you can go to the online experience page to chat with the model. Through the chat, you will find that the model's self-awareness has changed.

Note: This practice focuses on enhancing R1's self-awareness. To improve other general capabilities, it is required to properly filter, clean, and adjust the ratio of data sets for fine-tuning. Considering R1's inherent strong capabilities, enhancing its general capabilities through fine-tuning poses certain thresholds and challenges. We encourage algorithm engineers to continue exploring and practicing on Tencent Cloud TI-ONE.

Fine-Tuning a Custom LLM Using Task-based Modeling

Last updated: 2026-03-12 10:20:15

Overview

This document takes [qwen2-0.5b-instruct](#) as an example to demonstrate how to fine-tune a custom LLM using Tencent Cloud TI-ONE Platform (TI-ONE) and deploy the fine-tuned model for inference via Online Services. Here, we use the open-source LLaMA-Factory as the training framework, perform full-parameter SFT on qwen2-0.5b-instruct, and finally use the built-in image Angel-vLLM of the platform to deploy the trained model for inference.

Prerequisites

Application for Cloud File Storage (CFS) or Data Accelerator Goose FileSystem extreme (GooseFSx): In the process of fine-tuning custom LLMs, the storage used for training data, models, and code is CFS or GooseFSx. Therefore, it is required to apply for CFS or GooseFSx first.

Operation Steps

Material Preparations

Note:

Currently, TI-ONE does not support direct data upload through the console. To resolve this issue, you need to create an Ops dev machine that mounts CFS or GooseFSx instances and use the dev machine service to upload or download LLMs, training code, and other files.

Choose **Training Workshop > Dev Machines** and click **Create** to create a dev machine for debugging training code.

- **Image:** Select any built-in image, as this dev machine is only used for downloading model files and training code.
- **CVM Instance Source:** Purchase from the TI-ONE platform or select from Cloud Virtual Machine (CVM) instances.
- **Billing Mode:** Select either pay-as-you-go or yearly/monthly subscription billing mode. For billing rules supported by TI-ONE, see [Billing Overview](#).
- **Storage Path Configuration:** Select CFS or GooseFSx. The name should be the CFS or GooseFSx instance you applied for and configured in the Prerequisites section. The path is the root directory / by default and is used to specify the storage location for your custom LLM.

- **Other Settings:** Not required by default.

Model Files

After successful creation, choose **Dev Machines > Python3(ipynkernel)** to download the required model through the script. You can search for the LLMs you need in [ModelScope](#) or [Hugging Face](#), download the models through Python scripts in ModelScope, and save them to CFS. This document takes the **Qwen2-0.5B-Instruct** model as an example, and the code for download is as follows:

```
!pip install modelscope

from modelscope import snapshot_download
#qwen/Qwen2-0.5b-Instruct is the name of the model to be downloaded,
cache_dir is the address where the downloaded model is saved, and ./
indicates saving the downloaded model in the root directory of the mount
directory of CFS.
model_dir = snapshot_download('qwen/Qwen2-0.5b-Instruct', cache_dir='./')
```

Note:

The specified `cache_dir` address for the downloaded model is the location in Task-based Modeling, namely, the mounted dev machine's path + `cache_dir`. For example, if the mounted dev machine's path is `/dir1` and `cache_dir` is `/dir2`, the file location in CFS will be `/dir1/dir2`.

Copy the above download script, replace relevant content with the model to be downloaded, paste the modified script into the new `.ipynb` file, and click **Run** to start downloading the model.

```

File Edit View Run Kernel Tabs Settings Help
root@mb-1237656010088X
Python 3 (ipykernel)
[ ]: | pip install modelscope

from modelscope import snapshot_download
#qwen/Qwen2-0.5B-Instruct为模型名称，cache_dir为下载模型保存的地址，这里 '.' 表示将下载模型保存在CFS的根目录中
model_dir = snapshot_download('qwen/Qwen2-0.5B-Instruct', cache_dir='.')

Looking in indexes: http://mirrors.tencentyun.com/pypi/simple
Collecting modelscope
  Downloading http://mirrors.tencentyun.com/pypi/packages/94/34/cc9faf34851d4b75fb94a9c5748ff6c67cc55c1bf44014e638a2087dc02/m
odelscope-1.22.3-py3-none-any.whl (5.8 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 8.4 MB/s eta 0:00:00:01:00:01
Requirement already satisfied: requests>=2.25 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.32.3)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from modelscope) (4.66.2)
Requirement already satisfied: urllib3>=1.26 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->model
scope) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->modelscope) (3.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.25->modelscope)
(2024.2.2)
DEPRECATION: mlx-tools<5.2.0 has a non-standard version number. pip 24.1 will enforce this behaviour change. A possible rep
lacement is to upgrade to a newer version of mlx-tools or contact the author to suggest that they release a version with a co
rrecting version number. Discussion can be found at https://github.com/ppya/pip/issues/12063 class="ansi-yellow-fg">
Installing collected packages: modelscope
Successfully installed modelscope-1.22.3
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package man
ager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv class="ansi-yellow-fg">

[notice] A new release of pip is available: 24.0 -> 24.3.1
[notice] To update, run: python -m pip install --upgrade pip
Downloading Model to directory: ./qwen/Qwen2-0.5B-Instruct
2025-01-21 15:30:32,159 - modelscope - INFO - Got 10 files, start to download ...
Processing 10 items: 90% ██████████ | 9.00/10.0 [00:01-00:00, 9.12it/s]

[ ]:

```

Training Code

Downloading LLaMA-Factory Code

Directly download the code to CFS via git clone by clicking Terminal to enter the following command. If you need the latest code, use the main branch.

```
git clone -b v0.9.2 https://github.com/hiyouga/LLaMA-Factory.git
```

Alternatively, [download](#) the open-source code locally first, and then upload the LLaMA-Factory source code to the dev machine and unzip it.

If you enable SSH Connection during creation, you can use SCP to upload local materials to the dev machine.

```
scp -r -P <port> Qwen2-0.5B-Instruct
root@host:/home/tione/notebook/workspace
```

Tips: If your file is downloaded via Git, you can try to delete the .git directory under the directory to reduce the number of files to be uploaded and improve the upload speed.

Modifying the Training Configuration File

Access the LLaMa-Factory/examples/train_full directory, copy qwen2vl_full_sft.yaml, rename it to qwen2_full_sft.yaml, and edit the corresponding content.

```

1  ### model
2  model_name_or_path: /opt/ml/pretrain_model
3  image_resolution: 262144
4  video_resolution: 16384
5  trust_remote_code: true
6
7  ### method
8  stage: sft
9  do_train: true
10 finetuning_type: full
11 freeze_vision_tower: true # choices: [true, false]
12 freeze_multi_modal_projector: true # choices: [true, false]
13 train_mm_proj_only: false # choices: [true, false]
14 deepspeed: examples/deepspeed/ds_z3_config.json # choices: [ds_z0_config.json, ds_z2_config.json, ds_z3_config.json]
15
16
17 ### dataset
18 dataset: identity
19 template: qwen
20 cutoff_len: 2048
21 max_samples: 1000
22 overwrite_caches: true
23 preprocessing_num_workers: 16
24
25 ### output
26 output_dir: /llm-private/output
27 logging_steps: 10
28 save_steps: 500
29 plot_loss: true
30 overwrite_output_dir: true
31
32 ### train
33 per_device_train_batch_size: 1
34 gradient_accumulation_steps: 2
35 learning_rate: 1.0e-4
36 num_train_epochs: 3.0
37 lr_scheduler_type: cosine
38 warmup_ratio: 0.1
39 bf16: true
40 ddp_timeout: 18000000
41
42 ### eval
43 # val_size: 0.1
44 # per_device_eval_batch_size: 1

```

The reference data set used for training is located in the LLaMa-Factory/data directory. The data set used here is identity.json (you can replace `{{name}}` and `{{author}}` within it to verify the fine-tuning effect). Below are the main hyperparameters that need to be modified. You can also adjust them as needed according to the official documentation:

```

# Base model path.
model_name_or_path: /opt/ml/pretrain_model

# Training data.
dataset: identity
template: qwen

# Training output path.
output_dir: /llm-private/output

# Training epochs (which can be appropriately reduced for quick
verification).
num_train_epochs: 3.0

```

ⓘ Note:

The directory here corresponds to the subsequent container mount directory for task-based modeling.

- If you need to customize the data set, modify the dataset field, and go to the LLaMa-Factory/data directory to modify the dataset_info.json file to configure the corresponding

data set information.

- If you need to modify other training parameters, modify them accordingly.
- If the model is small and prone to overfitting, it is recommended to reduce the value of `learning_rate`.

Creating a Startup Script

Create a `start.sh` file in the LLaMA-Factory directory, which is used as a training startup script with the following content:

```
#!/bin/bash
TRAIN_CONFIG=$1
DISTRIBUTED_ARGS="
    --nproc_per_node ${NPROC_PER_NODE:-$(nvidia-smi -L | wc -l)} \
    --nnodes ${WORLD_SIZE:-1} \
    --node_rank ${RANK:-0} \
    --master_addr ${MASTER_ADDR:-127.0.0.1} \
    --master_port ${MASTER_PORT:-23456}
"

set -ex
torchrun $DISTRIBUTED_ARGS src/train.py $TRAIN_CONFIG
```

The above startup script supports distributed training for task-based modeling.

Launching a Fine-Tuning Task in Task-based Modeling

Creating a Training Task

Choose **Training Workshop > Task-based Modeling > Create Task**, select the image you have prepared as the training image, set the training mode to **DDP**, and select resources as needed. If multi-machine distributed training is required, you can select 8 GPU cards and set the number of nodes to be greater than 1.

- **Training Image:** Select the built-in image `PyTorch/tilearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu`. This image has been configured with the running environment for LLM training by default.
- **CVM Instance Source:** See the configuration of [Dev Machines > Computing Resources](#).
- **CVM Instance Specifications:** Select training resources appropriately. If multi-machine distributed training is required, you can select 8 GPU cards and set the number of nodes to be greater than 1.
- **Storage Path Configuration:** The **training code**, **base model**, and **training output path** need to be mounted. The container mount path corresponds to the relevant parameters in the

training configuration file mentioned above. You can also place the training code, base model, and training output path in the same large directory, mount the root directory to the container, and modify the startup command to run the `cd` command to access LLaMA-Factory.

- **Startup Command:** (Since the main branch of LLaMA-Factory relies on a later version of the transformers library, we have online upgraded the third-party library version corresponding to the training image of the platform in the startup command. The final startup command written is as follows:)

```
cd /opt/ml/code
pip3 install -r requirements.txt -i
https://mirrors.tencentyun.com/pypi/simple
bash start.sh examples/train_full/qwen2_full_sft.yaml
```

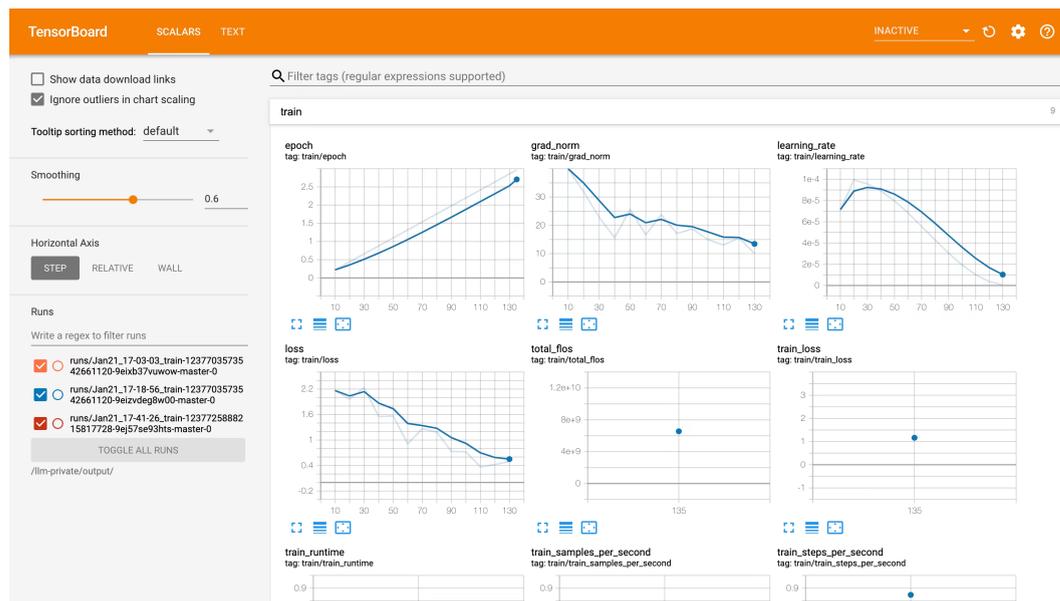
Viewing Training Status

After the task is started, you can click **Logs** to view training logs and other information.

After training is completed, the final checkpoint will be saved to the mounted output directory `/project/output`, which is used for subsequent deployment of the fine-tuned model.

Additionally, you can select **Task List** and click **TensorBoard** in the Operation column to configure a TensorBoard task. The path `/project/output` is the output path for model training in CFS.

After confirmation, you can view the status of the training process, such as loss and gradient.



Deploying the Model After Fine-Tuning

Deploying a Service

Choose **Model Services** > **Online Services** > **New Service**.

- **Service Name:** Customize a service name.
- **Model Source:** Select CFS, select the CFS instance where the model is stored, and fill in the actual training output path. If you need to select a certain checkpoint, specify the path up to the checkpoint-level directory.
- **Running Environment:** Select the built-in LLM/Angel-vLLM(2.1) image.

Note:

The platform provides built-in training images that include [Built-in Training Image List](#). Theoretically, the built-in images can support most Hugging Face LLMs. If they cannot meet your requirements, submit a ticket to contact us. Besides the built-in images, you can also install the software packages you need according to [Custom Training Image Specifications](#). Then, you can select your custom images on the image selection page for training. In such scenarios, before using an image in task-based modeling, you should first go to the Dev Machines page to ensure the image can successfully start the corresponding training task. When creating training images, follow the [Custom Training Image Specifications](#) of the platform.

- **CVM Instance Specifications:** Select resources based on the actual model size or your available resources. The CVM instance resources required for LLM inference depend on the model parameter scale. It is recommended to evaluate the required resources by seeing [Guide on Resources Required by LLM Inference](#).

You can start the inference service with default configurations. To learn more about the features of the Angel-vLLM inference image and how to use advanced settings, see [Inference Acceleration Based on the Built-in Image Angel-vLLM](#). A specific example is shown below:

Service Call – Online Experience

Return to the upper-level menu and click the **Service Call** tab of the service. In the API calling address field, enter `/v1/chat/completions` and the request body. The popular OpenAI Chat Completions API is currently supported, as shown below:

Service Call – Local Test Scripts

Calling address:

Local test script:

Note:

You need to modify the calling address and prepare test data.

```
# -*- coding: UTF-8 -*-
```

```
import time
import requests
import argparse
import json

from concurrent.futures import ThreadPoolExecutor, as_completed

def generate(prompt, log_stream=False):
    start = time.time()
    header = {
        "content-type": "application/json",
    }

    # print(prompt, type(prompt))
    data = {
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.7,
        "max_tokens": 256,
    }
    result = requests.post(f"{args.server}/v1/chat/completions",
        headers=header,
        json=data,
        stream=True
    )
    response = ""
    for part in result.iter_lines():
        if part:
            if "content" in part.decode("utf-8"):
                print(part.decode("utf-8"))
                content = json.loads(part.decode("utf-8"))["choices"][0]
                ["message"]["content"] # Filter the string to remove the data: prefix,
                convert the string into JSON format, and then extract the text.
                response += content
                if log_stream:
                    print(content, end = "", flush = True)
    if log_stream:
        print()
    end = time.time()
    return {"prompt": prompt, "generated_text": response, "time_cost": end
        - start}
```

```
def main():
    # Prepare the test set.
    prompts = []
    with open(args.test_data, "r", encoding="utf8") as data:
        for line in data.readlines():
            line_dict = json.loads(line.strip())
            prompt = line_dict['question']
            prompts.append(prompt)

    # Alternatively, use a few simple test examples:
    prompts = ["Hello!", "Who are you?"]
    task_list = []
    t1 = time.time()
    with ThreadPoolExecutor(max_workers=args.concurrency) as executor:
        for prompt in prompts:
            task_list.append(executor.submit(generate, prompt,
args.verbose))
        for item in as_completed(task_list):
            result = item.result()
            print(result)

    total_time = time.time() - t1
    req_per_sec = len(prompts) / total_time
    print(f"total_time={total_time:.4f}s, requests/s={req_per_sec:.4f}")

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    # Change the calling address to the address of the online service.
    parser.add_argument("-s", "--server", type=str, default="https://ms-
xxxxxxxx-yyyyyy.cs.com/ms-xxxxxxxx", help="Address of the inference
service")
    parser.add_argument("-d", "--test-data", type=str,
default="./test.jsonl", help="Test JSONL file")
    parser.add_argument("-c", "--concurrency", type=int, default=4,
help="Number of concurrent requests")
    parser.add_argument("-v", "--verbose", action="store_true",
help="Print streaming details")
    args = parser.parse_args()
    main()
```

Response example:

```
{'prompt': 'Who are you?', 'generated_text': 'Hello, I am Xiatian, an AI  
assistant developed by Tencent. I can help you solve various language-  
related issues and tasks.', 'time_cost': 1.1000895500183105}  
total_time=1.1006s, requests/s=0.9086
```

Using Task-based Modeling for GRPO Reinforcement Learning Training

Last updated: 2026-03-12 10:43:02

Overview

This document takes [qwen3-0.6b](#) as an example to demonstrate how to use Tencent Cloud TI-ONE Platform (TI-ONE) to perform Group Relative Policy Optimization (GRPO) reinforcement learning training for LLMs and deploy the models obtained from training for inference via Online Services. Here, the open-source framework [veRL](#) is used as the training framework to perform GRPO reinforcement learning training tasks on qwen3-0.6b, and finally, the built-in image Angel-vLLM in the platform is used to deploy the trained model for inference.

Prerequisites

Application for Cloud File Storage (CFS) or Data Accelerator Goose FileSystem extreme (GooseFSx): In the process of fine-tuning custom LLMs, the storage used for training data, models, and code is CFS or GooseFSx. Therefore, it is required to apply for CFS or GooseFSx first.

Operation Steps

Material Preparations

Note:

Currently, TI-ONE does not support direct data upload through the console. To resolve this issue, you need to create an Ops dev machine that mounts CFS or GooseFSx instances and use the dev machine service to upload or download LLMs, training code, and other files.

Choose **Training Workshop > Dev Machines > Create** to create a dev machine for debugging training code:

- **Image:** Select the built-in generic image `verl0.6.1-vllm0.11.0-torch2.8-py312-cuda12.8-gpu`.
- **CVM Instance Source:** Purchase from the TI-ONE platform or select from Cloud Virtual Machine (CVM) instances.
- **Billing Mode:** Select either pay-as-you-go or yearly/monthly subscription billing mode. For billing rules supported by TI-ONE, see [Billing Overview](#).

- **Storage Path Configuration:** Select CFS or GooseFSx. The name should be the CFS or GooseFSx instance you applied for and configured in the Prerequisites section. The source path defaults to the root directory / and is used to specify the storage location for your custom LLM.
- **Other Settings:** Not required by default.

Model Files

After successful creation, choose **Dev Machines > Open**, select VSCode to access the Dev Machines module, open the terminal to perform operations, and download the required model in the mount directory via scripts or Cloud Object Storage (COS).

Script download

You can search for the LLMs you need in [ModelScope](#) or [Hugging Face](#), download the models through Python scripts in ModelScope, and save them to CFS. This document takes the Qwen3-0.6B model as an example, and the code for download is as follows:

```
modelscope download --model Qwen/Qwen3-0.6B --local_dir  
/home/tione/notebook/models/Qwen3-0.6B
```

COS download

You can first upload the model you need to COS, and then use COSCLI in the dev machine to download it from COS to the dev machine.

You can upload and download the model you need through the [console](#). In the dev machine, you can use [COSCLI](#) to download the model. A command for reference is as follows.

```
coscli sync -r cos://Directory name local directory name.
```

Training Code

- Training framework: veRL

The image has a built-in veRL framework repository in the `/workspace/verl` path. You can also download the latest open-source code.

Data Sets

This document takes the [GSM8K](#) data set as an example to demonstrate the training and inference deployment of reinforcement learning tasks for the qwen-0.6b model. In brief, you need to first download the data set and then preprocess it using the preprocessing code of veRL.

```
modelscope download --dataset AI-ModelScope/gsm8k --local_dir
/home/tione/notebook/download/gsm8k
rm /home/tione/notebook/download/gsm8k/dataset_infos.json
python3 /workspace/verl/examples/data_preprocess/gsm8k.py --local_save_dir
/home/tione/notebook/data/gsm8k --local_dataset_path
/home/tione/notebook/download/gsm8k
```

The `--local_save_dir` parameter specifies the location where you want to store the downloaded data set. The preprocessed data set will be used in our subsequent training and inference.

Note:

The directory format of the data set downloaded from ModelScope is slightly different from that downloaded from Hugging Face. To ensure compatibility with the data set loading logic of the datasets library of Hugging Face, you need to manually delete the `dataset_infos.json` file in the download directory. If you directly download the data set from Hugging Face, the aforementioned `rm` operation is not required.

Writing a Startup Script

You can refer to the [training script](#) for the GSM8K data set provided by veRL to create your training script. The following is an example startup script for running on TI-ONE. You can create a startup script file at `/home/tione/notebook/start.sh` in the dev machine.

```
#!/bin/bash
set -e

### Training resource parameters.
# Number of training nodes. If task-based modeling is used for multi-
machine Ray training and no GPU card is allocated to the head node,
manually configure this parameter to the number of worker nodes to which
GPU cards are actually allocated.
NNODES="${WORLD_SIZE:-1}"
# Number of GPU cards per node. If task-based modeling is used for multi-
machine Ray training and no GPU card is allocated to the head node,
manually configure this parameter to the actual number of GPU cards per
worker node.
GPU_PER_NODE="${GPU_NUM_PER_NODE:-$(nvidia-smi -L | wc -l)}"

### Path parameters.
```

```
# Working directory: This directory must exist and should not contain too
large files such as data sets, models, or checkpoints, as this may cause
task submission delays or failures (alternatively, you can use the
excludes option in runtime_env to filter out files unnecessary for task
submission).
```

```
WORKING_DIR=/home/tione/notebook/verl
```

```
# Data set path.
```

```
TRAIN_FILES=/home/tione/notebook/data/gsm8k/train.parquet
```

```
VAL_FILES=/home/tione/notebook/data/gsm8k/test.parquet
```

```
# Base model path.
```

```
MODEL_PATH=/home/tione/notebook/models/Qwen3-0.6B
```

```
### Ensure unbuffered log output.
```

```
export PYTHONUNBUFFERED=1
```

```
### Training commands and parameters.
```

```
python3 -m verl.trainer.main_ppo \  
  algorithm.adv_estimator=grpo \  
  trainer.n_gpus_per_node=$GPU_PER_NODE \  
  trainer.nnodes=$NNODES \  
  data.train_files=$TRAIN_FILES \  
  data.val_files=$VAL_FILES \  
  data.train_batch_size=256 \  
  data.max_prompt_length=512 \  
  data.max_response_length=512 \  
  data.filter_overlong_prompts=True \  
  data.truncation='error' \  
  actor_rollout_ref.model.path=$MODEL_PATH \  
  actor_rollout_ref.actor.optim.lr=1e-6 \  
  actor_rollout_ref.model.use_remove_padding=True \  
  actor_rollout_ref.actor.ppo_mini_batch_size=128 \  
  actor_rollout_ref.actor.ppo_micro_batch_size_per_gpu=40 \  
  actor_rollout_ref.actor.use_kl_loss=True \  
  actor_rollout_ref.actor.kl_loss_coef=0.001 \  
  actor_rollout_ref.actor.kl_loss_type=low_var_kl \  
  actor_rollout_ref.actor.entropy_coeff=0 \  
  actor_rollout_ref.model.enable_gradient_checkpointing=True \  
  actor_rollout_ref.actor.fsdp_config.param_offload=False \  
  actor_rollout_ref.actor.fsdp_config.optimizer_offload=False \  
  actor_rollout_ref.rollout.log_prob_micro_batch_size_per_gpu=40 \  
  actor_rollout_ref.rollout.tensor_model_parallel_size=1 \  

```

```
actor_rollout_ref.rollout.name=vllm \  
actor_rollout_ref.rollout.gpu_memory_utilization=0.7 \  
actor_rollout_ref.rollout.n=5 \  
actor_rollout_ref.rollout.disable_log_stats=False \  
actor_rollout_ref.ref.log_prob_micro_batch_size_per_gpu=40 \  
actor_rollout_ref.ref.fsdps_config.param_offload=True \  
algorithm.use_kl_in_reward=False \  
trainer.critic_warmup=0 \  
trainer.logger='["console","tensorboard"]' \  
trainer.default_local_dir=$WORKING_DIR/checkpoints \  
trainer.project_name='verl_grpo_example_gsm8k' \  
trainer.experiment_name='qwen3_0.6b_function_rm' \  
trainer.save_freq=40 \  
trainer.test_freq=20 \  
trainer.total_epochs=10
```

For all specific parameters, see [Config Explanation](#). We provide explanations for the parameters you may need to modify:

- **TRAIN_FILES:** specifies the parquet files used for the training set.
- **VAL_FILES:** specifies the parquet files used for the validation set.
- **MODEL_PATH:** specifies the path of the model to be trained.
- **WORKING_DIR:** working directory. Training products are stored in this directory by default.
- **data.train_batch_size:** training batch size.
- **trainer.save_freq:** the frequency (by iteration) at which model parameters are saved. To save space, model parameters are stored in a sharded format for model parallelism.
- **trainer.test_freq:** the test frequency (by iteration). A data entry will be used for result validation.
- **trainer.total_epochs:** total training epochs.
- **trainer.project_name / trainer.experiment_name:** the project and experiment names for this training. The tensorboard logs generated during training will be saved in the `$WORKING_DIR/tensorboard_log/${trainer.project_name}/${trainer.experiment_name}` directory.
- **trainer.default_local_dir:** the path for saving the checkpoints generated during training.
- Other parameters such as `batchsize`, `tensor_model_parallel_size`, and `gpu_memory_utilization` should be adjusted based on the actual model size and CVM instance resources by referring to the official documentation [Config Explanation](#) of veRL and corresponding issues for errors.

Creating a Training Task

Choose **Training Workshop > Task-based Modeling > Create Task**, select the image you have prepared as the training image, set the training mode to Ray, and select resources as needed. If multi-machine distributed training is required, you can select 8 GPU cards and set the number of nodes to be greater than 1.

- **Training Image:** Select the built-in generic image `ver10.6.1-vllm0.11.0-torch2.8-py312-cuda12.8-gpu`.
- **Training Mode:** veRL natively supports using [Ray](#) for distributed training deployment. Here, we can directly select the training mode of Ray when creating a task-based modeling task.
- **CVM Instance Source:** See the Dev Machines section.
- **CVM Instance Specifications:** Select training resources appropriately. If multi-machine distributed training is required, it is recommended to configure the same number of GPU cards (typically 8 GPU cards) for both head and worker nodes. The total number of training nodes is the sum of head and worker nodes. Configuring the head node as a pure CPU node is also supported, with GPU cards only allocated to the worker nodes for training (the number of nodes and the number of GPU cards per node should be correctly configured in the training script).
- **Storage Path Configuration:** Configure the mount type and training output, with the same source path and container path as configured for the dev machine above.
- **Startup Command:**

```
bash /home/tione/notebook/start.sh # Replace the path with the path to
the startup script mentioned above, or directly paste the full content of
the startup script.
```

- **Advanced Framework Parameters – RuntimeEnv Configuration:**

```
{
  "working_dir": "<Your working directory>",
  "excludes": ["/.git/", "/checkpoints/"],
  "env_vars": {
    "TORCH_NCCL_AVOID_RECORD_STREAMS": "1",
    "CUDA_DEVICE_MAX_CONNECTIONS": "1",
    "VLLM_ALLREDUCE_USE_SYMM_MEM": "0",
    "PYTHONUNBUFFERED": "1",
    "NCCL_NVLS_ENABLE": "0"
  }
}
```

Viewing Training Status

After a task is started, you can click the name of this task-based modeling task to go to the log page and view the training log information.

Obtaining Training Output Results

After the training is completed, you can access the CFS instance to which the training task is mounted to view the training output results. You can mount a dev machine to which the training output path is mounted and then perform model evaluation and testing in this dev machine. Subsequently, you can also mount the CFS path where the model is located through the **Online Services** module of the platform and release it as an online service.

Guide on Resources Required by LLM Training

Last updated: 2026-02-26 15:20:00

This document introduces the configuration resources that can guarantee the normal running of models when LLM training is performed on Tencent Cloud TI-ONE Platform (TI-ONE). It is for your reference only.

The following table describes the recommended resources for training the open-source LLMs built in the platform.

	Recommended Resource (SFT-FULL) BatchSize=1; MaxSequenceLength=2048	Recommended Resource (SFT-LORA) BatchSize=1; MaxSequenceLength=2048
Models below 7B	HCCPNV6 instance type: 1 GPU card for models below 3B; 2 GPU cards for 7B/8B models.	HCCPNV6 instance type: 1 GPU card
13B models	HCCPNV6 instance type: 4 GPU cards	HCCPNV6 instance type: 1 GPU card
32B models	HCCPNV6 instance type: 8 GPU cards	HCCPNV6 instance type: 2 GPU cards
70B models	HCCPNV6 instance type: 2 CVM instances with 16 GPU cards	HCCPNV6 instance type: 4 GPU cards
DeepSeek-R1-671b/DeepSeek-V3-671b	HCCPNV6 instance type: 32 CVM instances with 256 GPU cards	Not supported
Hunyuan-large	HCCPNV6 instance type: 8 CVM instances with 64 GPU cards	HCCPNV6 instance type: 8 GPU cards

- The open-source LLMs built in the platform use the fine-tuning mode (Low-Rank Adaptation, short for LORA) by default, and you can use the `FinetuningType` parameter to configure this mode.
- A 7B model requires a single node with 100 cores and 500 GB of memory. A 13B or 70B model requires a single node with 150 cores and 1 TB of memory. For larger models, it is

recommended to use up all full-instance resources.

- Certain models use the tilearn acceleration technology, which can achieve an acceleration of about 30% when these models are trained on the recommended resources.

LLM and MLLM Data Preparation

Preprocessing LLM Fine-Tuning Data

Last updated: 2026-02-11 18:28:42

Overview

High-quality training data is crucial for improving the model fine-tuning effect. To help you prepare high-quality training data, Tencent Cloud TI-ONE Platform (TI-ONE) has integrated the standard data preprocessing process used by its R&D staff into the Data Construction module of Data Center. You can modify the process according to your specific business scenarios to obtain a custom data preprocessing pipeline.

This document uses a simple raw data sample as an example to describe how to generate a dataset for fine-tuning an LLM in the Data Construction module of TI-ONE.

Prerequisites

You have activated Cloud File Storage (CFS) as instructed in the [CFS usage documentation](#), as all your data files need to be stored on CFS.

Material Preparations

To perform the data preprocessing process, you need to prepare the following materials:

- **Raw dataset:** Ensure that you have downloaded a raw dataset to your local computer, or you have a specific link to access the raw dataset. This document assumes that you are using the sample data provided by the platform: `dataset_preparation_demo.json`.
- **Data preprocessing code:** If you have no special requirements, you can directly use the built-in data preprocessing code of the platform. You only need to modify certain fields in the sample code according to your dataset format. If the built-in data preprocessing code does not meet your requirements, you can also prepare part of the preprocessing code.

How to Prepare Raw Training Data

Defining Business Requirements

You need to prepare a dataset based on your objectives or business requirements. Typically, fine-tuning an open-source base model with a business-specific dataset can significantly enhance the model's business capabilities. For a specific task scenario, preparing approximately several thousand high-quality fine-tuning data entries enables the LLM to acquire this capability and deliver an excellent effect.

You can also use mainstream LLMs to test your business questions in batches in advance, so that you can determine whether existing LLMs can meet the requirements of this task scenario

or even provide better answers than those in the dataset you have prepared. If such a situation occurs, the fine-tuning dataset needs to be further improved.

Defining High-Quality Training Data Standards

1. The Q&A in the dataset must meet your requirements (such as for the accuracy, logic, and style of answers). The answers must be absolutely correct, and the expression has been optimized to the utmost. This is because the dataset is used to solve practical problems and serves as a critical basis for LLM learning. The LLM will comprehensively understand the answers and expression style of the dataset, and quickly adapt to them during learning. Consequently, any imperfections in your data will be learned by the LLM, ultimately compromising the final effect.
2. The instructions in the dataset need to be diversified to prevent the LLM from overfitting to certain phrases and thereby neglecting the actual meaning of natural language. For example, if you design only one type of instruction in the format of "Based on the following known information, answer the question and provide reasons. Known information: xxxxx. Question: xxxx" for a reading comprehension dataset, the LLM fine-tuned with the data may develop the following pattern: Once the phrase "Based on the following known information" is input, the LLM will naturally provide reasons while answering the question. This situation occurs because after learning from this batch of data, the LLM may mistakenly equate "Based on the following known information" with "Provide reasons when answering the question". Consequently, if you input "Based on the following known information, answer the question. Requirements: Only answer the question and do not provide reasons. Known information: xxxxx. Question: xxxx", the LLM will still provide reasons. This instruction overfitting phenomenon needs to be avoided, and the solution is to expand the instruction set of the dataset.

Preprocessing Process

Creating a Data Construction Task

In the **Data Center > Data Construction** module, click **New Task**, and then select a data preprocessing pipeline according to your business scenario. Each data preprocessing pipeline corresponds to a standardized data processing workflow. You can select a data preprocessing pipeline as needed.

Create Data Construction Task Data Construction

Currently, the platform has different built-in Data Construction pipelines for various scenarios for you, including Supervised single-turn QA pipeline, Supervised multi-turn QA pipeline, Unsupervised-pipeline. The following built-in content is provided in the pipeline: 1. The end-to-end pipeline code framework for data generation in this scenario. 2. Some atomic methods for data generation in this scenario. 3. A quick_start.ipynb document to help users quickly get started with the Data Construction Pipeline process.

Task name: Enter a name
Max 60 characters; supports English letters, digits, underscores (_), hyphens (-) and Chinese characters, must start with a letter, digit, or Chinese character

Region: **Shanghai**

Tag: Tag Key Tag Value Key-Value Clipboard

Data Construction Pipeline: Supervised single-turn QA pipeline Annotated single-turn QA pairs, suitable for training for SFT fine-tuning of LLM large models. Supervised multi-turn QA pipeline annotated multi-turn QA pairs, suitable for training for SFT fine-tuning of LLM large models. Unsupervised-pipeline unlabeled plain text data, suitable for pretraining of LLM large models.

General Algorithm Task: **Select a general algorithm task.**

Enter a keyword to search.

- Open-ended Q&A
- Basic language capability
- Dialogue
- Role playing
- Agent
- Chain of thought
- Reading comprehension
- Text understanding
- Information extraction
- Knowledge mining
- Text creation

Selected (0)

Level-1 Category	Level-2 Category	Level-3 Category

Please select one or more algorithm task categories that are suitable to your business scenarios. For different algorithm task categories, the platform will mix corresponding suitable built-in data (including supervision data) by default with your custom data for proportion fine-tuning in the subsequent model training phase, further improving model performance.

Remarks: Max 500 characters 0 / 500

Since the sample data provided in this document is subsequently used for supervised single-turn Q&A training, set **Data Preprocessing Pipeline to Supervised Single-Turn Q&A Pipeline**. After creating the task, click **Redirect to the Corresponding Dev Machine** to start creating (for the first time) or redirect to the data preprocessing dev machine.

About General Algorithm Tasks

To reduce the time required for preparing a dataset and help enhance the fine-tuning effect, the platform provides built-in datasets owned by Tencent. You can use them based on your business scenarios.

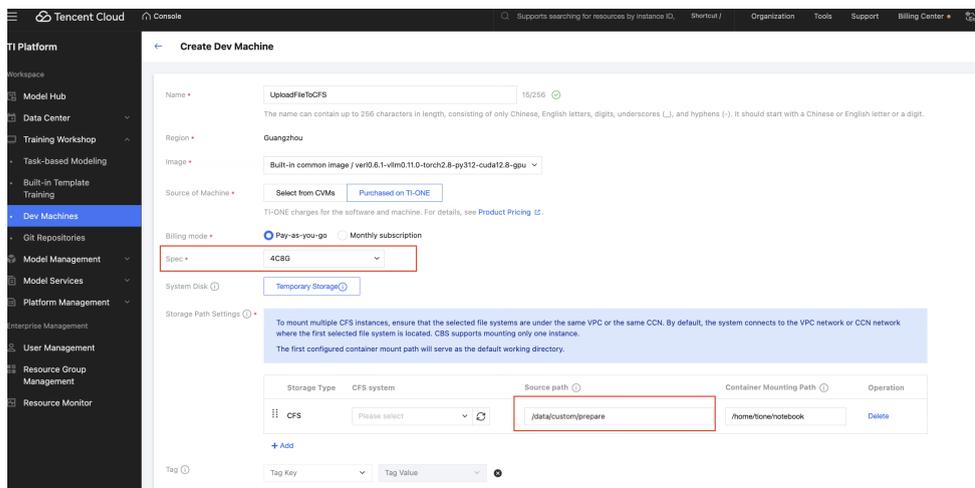
Note: The built-in data is not involved in the data preprocessing process mentioned in this document. However, when you subsequently fine-tune your LLM in the Training Workshop module, the platform will embed the built-in datasets corresponding to your selected business scenarios at a certain ratio into your fine-tuning dataset during the creation of the training task to enhance the fine-tuning effect.

If you do not need the built-in datasets of the platform, do not select any business scenarios in General Algorithm Tasks.

Creating a Data Construction Dev Machine

On the data preprocessing task list page, when you click **Redirect to the Corresponding Dev Machine** for the first time, it will trigger the dev machine creation process.

Since the built-in data preprocessing process of the platform involves calling a model (which consumes CPU and memory resources) and necessitates loading raw data into memory in a single operation for subsequent processing, you need to reserve sufficient Cloud Virtual Machine (CVM) instance specifications. This document uses 8C16G as an example. You can dynamically adjust the specifications based on the size of the raw dataset, but it is recommended that you use at least 4C8G computing resources for the subsequent processes. When creating a dev machine, you also need to **specify CFS information to store the raw dataset**, intermediate results, and processed data files. This document assumes that you mount the following CFS directory to `/home/tione/notebook`, with the mounted CFS source path being `/data/custom/prepare`.



Click **Confirm** to return to the dev machine list page. After the dev machine is created and enters the Running status, click **Open** in the **Operation** column and select **Jupyter** to start the data preprocessing process.

Uploading a Raw Data Set

You can upload your local raw dataset to CFS as instructed in [Fine-Tuning a Custom LLM Using Task-Based Modeling](#). Note: Since you have created a dev machine in the data preprocessing process, you can directly reuse it to upload the data.

Additionally, since the built-in data processing pipeline of the platform requires you to place your files in the specified directory, it is recommended that you upload your raw data set to the following directory:

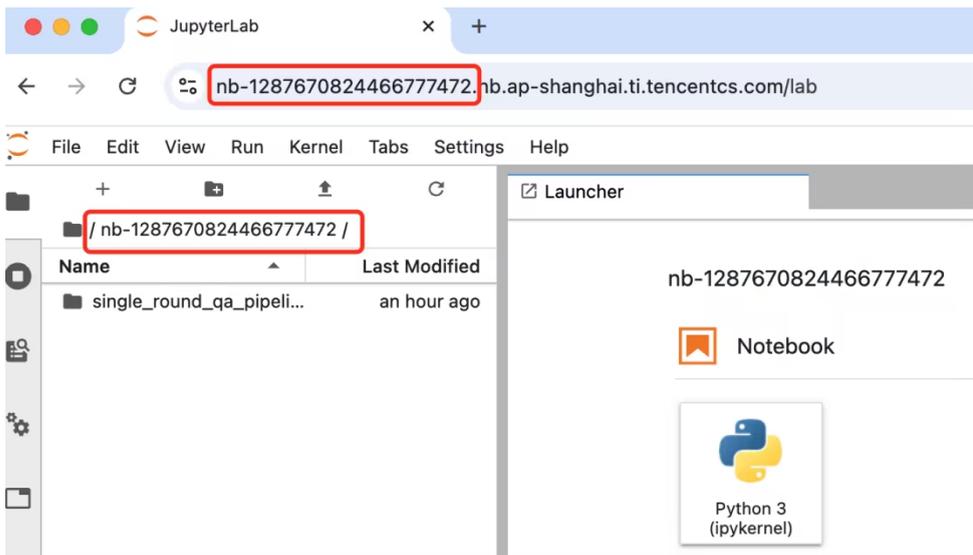
- **Dev machine:** `/home/tione/notebook/<nb-id>/single_round_qa_pipeline/raw_dataset_file`

It corresponds to the following CFS directory:

- **cfs:** `/data/custom/prepare/<nb-id>/single_round_qa_pipeline/raw_dataset_files`

Specifically,

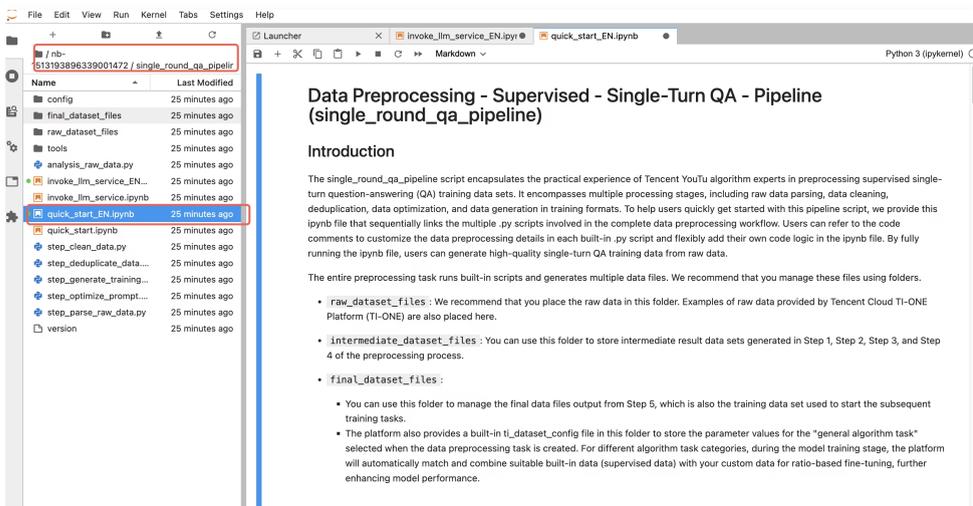
– The `nb-id` field indicates the ID of the dev machine you create. You can view the ID of the current dev machine in the URL or address bar of the browser.



- `single_round_qa_pipeline` is the script path for supervised single-turn Q&A. If you select the supervised multi-turn Q&A pipeline, the directory name is `multi_round_qa_pipeline`. If you select the unsupervised pipeline, the directory name is `unsupervised_pipeline`.

Data Construction

After creating the dev machine, click **Open** to go to the working path described earlier: `/home/ti-one/notebook/<nb-id>/single_round_qa_pipeline/`. Open `quick_start.ipynb` to view the complete data preprocessing process, as shown in the figure below.



It can be seen that the complete data preprocessing process is divided into the following steps:

- **Step 0: Initialize the environment.** Import necessary Python packages.
- **Step 1: Parse the raw data.** Convert the raw data in uncertain formats to the triplet format (`system, question, answer`) required by TI-ONE, and output it as a JSONL file.
- **Step 2: Clean the data.** Clean the data based on the configurations you specify. For example, filter the data (to remove non-compliant samples) or rewrite it (by converting it between traditional and simplified Chinese or between full-width and half-width characters).

- **Step 3: Deduplicate the data.** Calculate the embedding similarity of the cleaned data and deduplicate it.
- **Step 4: Optimize the data.** Optimize the prompt of training data to improve the understanding of instructions.
- **Step 5: Generate the training format.** Output necessary statistical analysis results of training data, prompt length, response length, and data formats that meet training requirements.

Among the six steps above, only Step 1 requires modification according to the input data format. For all other steps, you can use the default settings of the platform.

If the data preprocessed based on the default rules of the platform still falls short of your expectations, you can make targeted adjustments and optimizations according to the instructions for each step. Example:

- **Data cleaning:** You can open `config/clean_config.yaml` to modify the cleaning rule configuration.
- **Data cleaning:** You can customize and register a cleaning function according to the requirements of the platform.
- **Data optimization:** You can open `config/instruction_dict.yaml` and populate it with rich instruction replacement rules.

Step 1: Parsing the Raw Data (Which Requires Customizing a Parsing Function)

For the single-turn supervised pipeline in this example, the platform requires that the input data to be cleaned, deduplicated, or processed in other ways must be in JSONL format, and that each line as a JSON object is in the following triplet format:

```
{"instruction": "You are a chatbot that can understand and answer questions.", "question": "Xu, let us travel to Chongming next time. Is there any 5A-level scenic spot?", "answer": "There is one, Xisha Pearl Lake Scenic Area."}
```

Therefore, for your raw input format (such as JSON/JSONL/Parquet), you need to write a custom parsing function to convert the raw data format to the format required by the platform.

For the raw data, after you specify your file path and execute the code block for raw data sampling, we will output the sampling format of your raw data.

Step 1.1: Sampling the Raw Data

The built-in script in this step reads the raw data from `step1_in_file`, automatically counts and samples a specified number of sample data, outputs the corresponding data schema, and displays it below. This way, users can clearly understand the format of the input data. The required input data format for each sample in the TI-ONE's single-turn chat data preprocessing pipeline is defined below. You can refer to the schema displayed here to determine the data conversion method.

```
{"system": "xxxx", "question": "xxxx", "answer": "xxxx"}
```

```
[ ]: # Sample data
# The raw file path for parsing input raw data. It is recommended to uniformly manage raw inputs using the "raw_dataset_files"
# input supports files and folders.
input_path = "single_qa_example.jsonl" # A single file

step1_in_file = os.path.join("raw_dataset_files", input_path)
dataset_name = os.path.splitext(os.path.basename(input_path))[0].replace(".", "_")
print("Data set name:{}".format(dataset_name))

# Use the function defined in analysis_raw_data.py in the same directory to sample raw data.
analysis_raw_data.sample_raw_data_single_round_qa(step1_in_file, sample_num = 5)
```

```
{
  "$schema": "http://json-schema.org/schema#",
  "type": "object",
  "properties": {
    "input": {
      "type": "string"
    },
    "instruction": {
      "type": "string"
    },
    "output": {
      "type": "string"
    }
  },
  "required": [
    "input",
    "instruction",
    "output"
  ]
}

ID records
{
  "input": "What investment and wealth management methods do you usually use?",
  "instruction": "Please briefly answer the following question:",
  "output": "I'm not too sure either. I usually use bank savings, which are relatively low-risk."
}
{
  "input": "I found a counterfeit bill on my way home the other day. What should I do to handle it legally?",
  "instruction": "Please briefly answer the following question:",
  "output": "It should be promptly submitted to the People's Bank of China, public security authorities, or financial institutions that handle RMB deposits and withdrawals."
}
{
  "input": "My company needs my personal credit report. Where should I go to check it?",
  "instruction": "Please briefly answer the following question:",
  "output": "It can be checked through various methods, such as: bank inquiries, the official website of the People's Bank of China, and some mobile apps."
}
{
  "input": "Which industries can be considered part of the financial services sector?",
  "instruction": "Please briefly answer the following question:",
  "output": "Industries such as banking, securities, insurance, trust, and funds."
}
{
  "input": "Is there only one type of bank card?",
  "instruction": "Please briefly answer the following question:",
  "output": "No, there are two types of bank cards: debit cards and credit cards."
}
```

- If your raw data is in JSON or JSONL format, we will sample and display part of it.
- If your raw data is in Parquet format, we will convert it to the JSON format and then sample and display part of it.

After sampling the raw data and understanding its format, you need to convert it to the input format required by TI-ONE. For your input file, we will parse it into the multi-line JSON format and convert the JSON format to the Python dict format. To convert your JSON dict format to the format required by TI-ONE, you need to customize a `convert_dict_to_tione_format` function to implement your own conversion logic. See the following code segment for implementation:

```
# Parse data
# Output path for parsed data, in JSONL format. It is recommended to use the "intermediate_dataset_files" folder for centralized management.
step1_out_file = os.path.join(
  "intermediate_dataset_files", "%01_i_parsed.jsonl".format(dataset_name)
)

# You can override this function to parse JSON data in different formats. For your input file, we will parse it into multiple lines of JSON and convert the JSON format to a Python dict f
# In this function, you need to convert the dict into the format required by TI-ONE, {"system": "xxxx", "question": "xxxx", "answer": "xxxx"}, based on the sampled JSON schema.
def convert_dict_to_tione_format(item: dict) -> dict:
    try:
        question = item["instruction"] + "\n" + item["input"]
        answer = item["output"]
        # Sample construction: Obtain the "system" role field. Users can modify the specific content as needed.
        system = "You are the Intelligent Qn robot of TI-ONE, capable of understanding user questions and providing fluent answers."
        # Construct the sample format required by TI-ONE.
        convert_dict = {
            "system": system,
            "question": question,
            "answer": answer,
        }
    except:
        # Data parsing exception.
        return None
    return convert_dict

# Use the function defined in step_parse_raw_data.py in the same directory to parse the raw data.
step_parse_raw_data.parse_raw_data_single_round_qa(step1_in_file, step1_out_file, record_convert_fn=convert_dict_to_tione_format)

# After parsing is complete, perform raw data statistics on the output file from Step 1.
# The parameter num_bins specifies the number of evenly distributed bins for the histogram. When num_bins=0, the system's built-in bins are used ([2k, 4k, 8k, 16k, 32k]). The default val
analysis_raw_data.statistic_raw_data_single_round_qa(step1_out_file, num_bins = 10)
```

After completing Step 1, we will output statistical information about the raw samples, including the number of samples, the length distribution of question/answer fields in supervised training data, and statistics on Chinese and English data entries.

Step 2: Cleaning the Data (Which Supports Customizing Cleaning Rules)

Data cleaning functions fall into two categories: rewriting and filtering. The default built-in rules of the platform may meet the requirements of most scenarios. For special scenarios, you can also customize and register a cleaning function to add your custom cleaning rules here.

```
use this report to fine-tune and optimize your cleaning rules, and re-run the code block to ensure the resulting data quality meets your requirements.

j): # Path to the input .jsonl file for data cleaning, usually the output from the previous step.
step2_in_file = step1_out_file

# Path to the output .jsonl file for data cleaning. It is recommended to use the "intermediate_dataset_files" folder for centralized output management.
step2_out_file = os.path.join(
    "intermediate_dataset_files", "02_{}_cleaned.jsonl".format(dataset_name)
)
# Specify the specific field to be cleaned ("question" or "answer")
clean_key = "answer"
# Data cleaning configuration file
clean_config = "config/clean_config.yaml"

# Use the function defined in step2_clean_data.py under the same directory to perform data cleaning. sample_num specifies the amount of sampled data to show the differences before and after cleaning.
# To avoid truncation during the training process, you can set max_sequence_token_length to specify the maximum token sequence length (including system, question, and answer). Sample data:
step_clean_data.clean_data_single_round_qa(step2_in_file, step2_out_file,
                                          clean_config, clean_key, sample_num=10,
                                          max_sequence_token_length=32*1024)
```

After completing the cleaning process, we will output the data volumes before and after cleaning, sample part of the cleaned data, and display it in a visualized way to help you make a comparison and determine whether the cleaning rules meet your expectations.

Step 3: Deduplicating the Data (Deduplication Logic Modification Not Recommended)

The built-in data deduplication logic of the platform is complex and involves a time-consuming data deduplication process (model loading and data computation typically take over 5 minutes). Since this logic involves much expertise, it is not recommended that you implement it by yourself. However, you can specify the fields that require deduplication, such as question/answer.

```
j): # Path to the input .jsonl file for data deduplication, usually the output from the previous step.
step3_in_file = step2_out_file

# Path to the output .jsonl file for data deduplication. It is recommended to use the "intermediate_dataset_files" folder for centralized management.
step3_out_file = os.path.join(
    "intermediate_dataset_files", "03_{}_deduplicated.jsonl".format(dataset_name)
)
# Specify the specific fields requiring deduplication ("question", "answer", or multiple keys. We will concatenate them for deduplication).
dedup_key = ["answer"]
# Threshold for determining duplicates. Samples with text similarity greater than the threshold will be filtered out.
threshold = 0.9

# Use the function defined in step3_deduplicate_data.py under the same directory to perform data deduplication. sample_num specifies the amount of sampled data to show the deduplication process.
step_deduplicate_data.dedup_data_single_round_qa(step3_in_file, step3_out_file, dedup_key, threshold, sample_num = 10)
print("finished!")

Loading embedding model ...
Loading embedding model ... Done!
去重key: ['answer']
Extracting embeddings ...
Setting threads num to 2
100%|██████████| 8/8 [00:01:00:00, 6.82it/s]
Extracting embeddings ... Done!
Start Deduplication
100%|██████████| 8/8 [00:01:00:00, 4.37it/s]
100%|██████████| 8/8 [00:00:00:00, 61008.06it/s]
去重前数据量: 8
去重后数据量: 8
finished!
```

Step 4: Optimizing the Data (Which Allows Modifying Configuration Files)

You can modify the instruction list of `config/instruction_dict.py` as needed. The custom configuration based on scenarios can increase the diversity of instructions.

After completing data optimization, before generating training data, we will automatically collect and sample a specified amount of example data, run data statistics scripts, output the average length of Q&A pairs, length distribution histograms, and distribution pie charts in Chinese and English, so that you can understand whether the execution results of the current data preprocessing pipeline meet your expectations.

The average length of Q&A pairs, a distribution histogram of lengths, and a pie chart of Chinese and English distributions.

```
# Perform sampling and statistics on the files processed in Step 4.
analyse_in_file = step4_out_file

# Use the function defined in analysis_raw_data.py under the same directory for data sampling. sample_num: number of samples to display.
analysis_raw_data.sample_raw_data_single_round_qa(analyse_in_file, sample_num = 10)

# Use the function defined in analysis_raw_data.py to perform data statistics.
# The parameter num_bins specifies the number of evenly distributed bins for the histogram. When num_bins=0, the system's built-in bins are used. The default value is 10.
analysis_raw_data.statistic_raw_data_single_round_qa(analyse_in_file, num_bins = 10)
```

Step 5: Generating the Training Format (Generation Logic Modification Not Recommended)

```
# Path to the input .jsonl file for training data generation, usually the output from the previous step.
step5_in_file = step4_out_file

# Path to the output .jsonl file for training data generation. It is recommended to manage the final output uniformly under the "final_dataset_files" folder.
step5_out_file = os.path.join(
    "final_dataset_files", "{}_for_training.jsonl".format(dataset_name)
)

# Use the function defined in step5_generate_training_data.py under the same directory to generate training data.
step_generate_training_data.generate_training_data_single_round_qa(step5_in_file, step5_out_file)
```

After completing all the above processes, we have stored the preprocessed data in the following path of the dev machine: `/home/tione/notebook/<nb-id>/single_round_qa_pipeline/final_dataset_files` (that is, `cfs:///data/custom/prepare/<nb-id>/single_round_qa_pipeline/final_dataset_files`). You can use it in the LLM Fine-Tuning module.

The output results of this step include the mapping path of CFS. You can directly mount the CFS path in task-based modeling as the source mounting path for training data.

Using Preprocessed Training Data in the LLM Fine-Tuning Module

After preprocessing your own data set, you can use it for your training task as instructed in [use cases of fine-tuning LLMs](#).

Creating an LLM&MLLM Dataset and Labeling Task

Last updated: 2026-02-11 18:28:51

Overview

Based on a real-world requirement for LLM data labeling, this document describes how to create an LLM dataset and the corresponding labeling task in the Tencent Cloud TI-ONE Platform (TI-ONE) data center. The process for creating an MLLM labeling task is similar to that for creating an LLM labeling task.

Using **filtering of high-quality fine-tuning data**, a scenario that is common in Tencent YouTu, as an example, this document assumes that one sample from the raw data to be labeled is as follows:

```
{"instruction": "Briefly answer the following question:", "input": "What investment and financial management methods do you usually use?", "output": "I am not quite sure. I generally opt for bank savings, which are relatively low-risk."}
```

This labeling task requires that annotators manually read the instruction, input, and output fields in the data, and then determine the accuracy of the content description for this sample as Correct, Discard, or Questionable. Ultimately, only samples determined as Correct will be added to subsequent LLM fine-tuning; samples determined as Questionable will involve additional experts to jointly determine their quality; samples determined as Discard will be directly discarded and will not participate in fine-tuning.

The complete data file to be labeled needs to be saved in JSONL format. (Currently, TI-ONE only supports managing and labeling LLM data files in JSONL format. It is recommended that the file size does not exceed 1 GB.)

```

llmdemo.json
1 {"instruction": "Please briefly answer the following question:", "input": "What investment and wealth management methods do you usually use?", "output": "I'm not too sure either. I usually use bank savings, which are relatively low-risk."}
2 {"instruction": "Please briefly answer the following question:", "input": "Could you tell me what fund splitting means?", "output": "Fund splitting is a method of recalculating fund assets by maintaining the total asset value of fund investors unchanged, while changing the corresponding relationship between the net asset value per fund share and the total number of fund shares."}
3 {"instruction": "Please briefly answer the following question:", "input": "Which industries can be considered part of the financial services sector?", "output": "Industries such as banking, securities, insurance, trust, and funds."}
4 {"instruction": "Please briefly answer the following question:", "input": "How is the overall discount/premium rate calculated? Please provide me with a calculation formula.", "output": "Overall discount/premium rate = (Combined price - Net asset value of the parent fund) / Net asset value of the parent fund."}
5 {"instruction": "Please briefly answer the following question:", "input": "Is there only one type of bank card?", "output": "No, there are two types of bank cards: debit cards and credit cards."}
6 {"instruction": "Please briefly answer the following question:", "input": "I found a counterfeit bill on my way home the other day. What should I do to handle it legally?", "output": "It should be promptly submitted to the People's Bank of China, public security authorities, or financial institutions that handle RMB deposits and withdrawals."}
7 {"instruction": "Please briefly answer the following question:", "input": "My company needs my personal credit report. Where should I go to check it?", "output": "It can be checked through various methods, such as: bank inquiries, the official website of the People's Bank of China, and some mobile apps."}
8 {"instruction": "Please briefly answer the following question:", "input": "I want to deposit some money in the bank. How will a decrease in the deposit interest rate affect me?", "output": "A decrease in the deposit interest rate may directly reduce your deposit earnings."}

```

Operation Steps

Uploading Data Files to CFS

Since the storage type of LLM data files that can be managed by TI-ONE only supports Cloud File Storage (CFS), you need to first ensure that the raw data files are stored on CFS.

If your raw data files are already stored on CFS, you can skip this step.

If your raw data files are stored locally, you can upload them to CFS via the following steps.

First, you can create a dev machine in the **Training Workshop > Dev Machines** module and use this dev machine to mount your CFS.

The screenshot displays the 'Create Dev Machine' configuration page in the Tencent Cloud TI-ONE Platform. The page includes a sidebar with navigation options like 'Model Hub', 'Automatic Learning', 'Task Management', 'Data Center', 'Training Workshop', 'Task-based Modeling', 'Built-in Template Training', 'Dev Machines', 'Git Repositories', 'Model Management', 'Model Services', 'Platform Management', 'Enterprise Management', 'User Management', 'Resource Group Management', and 'Resource Monitor'. The main configuration area includes fields for Name, Region, Image, Source of Machine, Billing mode, Spec, System Disk, and Storage Path Settings. A table at the bottom shows the Storage Type (CFS), CFS system (Please select), Source path (/), and Container Mounting Path (/home/tione/notebook). The Configuration price is 0.17 CNY/hour.

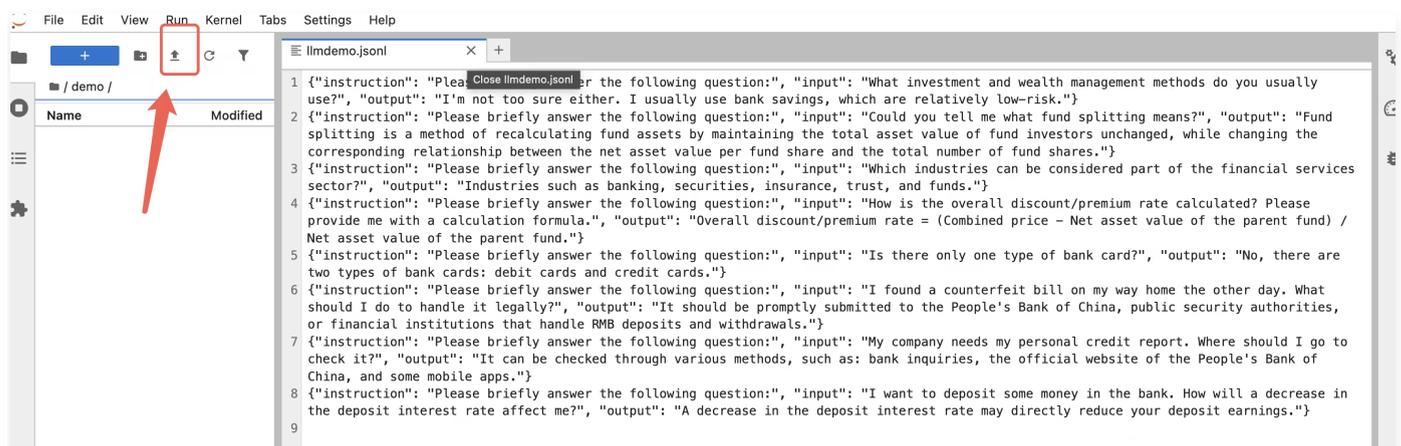
Introduction to key parameter configurations

- Name: Enter a custom instance name.

- **Image:** You can choose any built-in image. Since this dev machine is only used to mount CFS for uploading local data files, there are no requirements for the image.
- **CVM Instance Specifications:** You can choose the smallest computing power, as long as it can ensure that the dev machine can be started. You do not need to configure very large computing power.
- **Storage Path Settings:** Select your own CFS instance and enter the source path where the data files to be labeled will be saved. Subsequently, locally uploaded data files will be stored under this source path.

Then, when the dev machine is in the **Running** status, click **Open** in the Operation column to enter the dev machine.

Finally, click **Upload** in the red box area of the image below to upload your local data files to be labeled to the mounted CFS path. This CFS path will be the dataset path you need to register in Data Center later (to better manage your data files, you can also create a folder named dataset1 here and upload your data files to this folder).

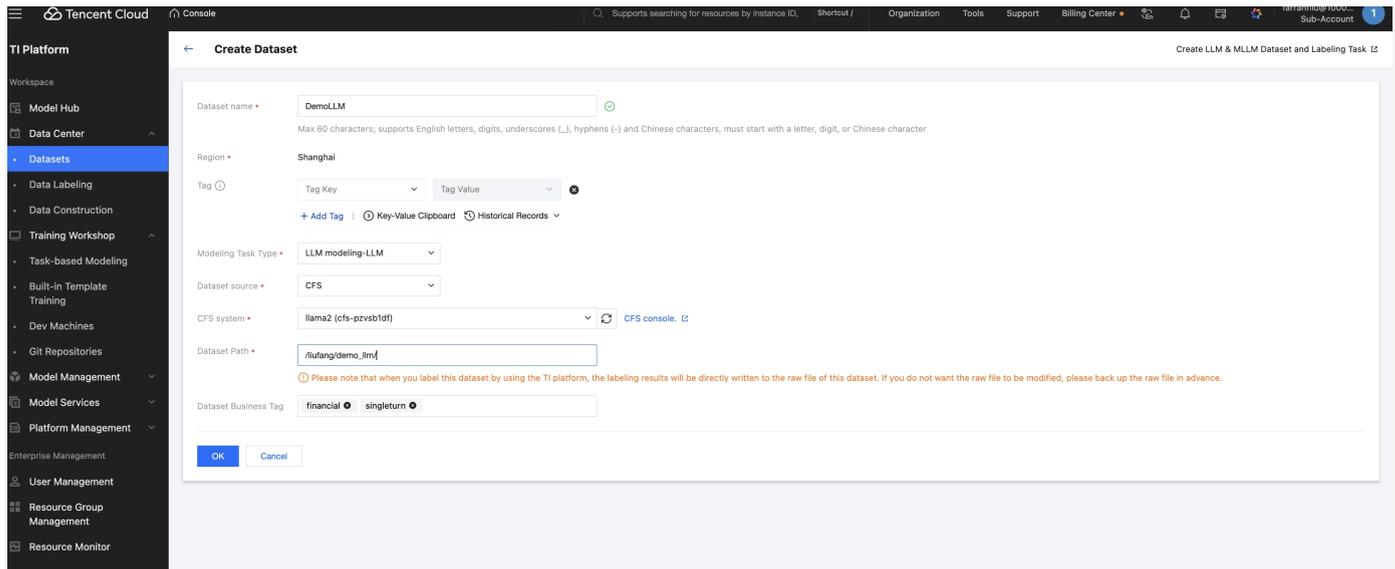


Creating a Dataset

In the **Data Center > Dataset** module, click **Create Dataset** to register data files stored on CFS to the Data Center module of TI-ONE for unified management. Currently, Data Center only supports integrating with and managing LLM datasets on the CFS instance.

Note:

- Data Center only associates this dataset with your CFS path and does not copy or dump your original data files.
- When you label this dataset on TI-ONE, labeling results will be written directly and in real time into the original files of your dataset. Therefore, if you do not want the original files to be modified, you need to back up the original files in advance.



Introduction to key parameter configurations

- **Dataset Name:** Enter a custom dataset name.
- **Modeling Task Type:** If Large Language Modeling – LLM is selected, this parameter serves only as a distinguishing marker for the task type and will not introduce strict differences in operation logic in subsequent processes of the product.
- **Dataset CFS Path:** Locate the CFS path where your data files are stored. The path does not need to be precise to the JSONL file. It only needs to be precise to the directory where the JSONL file is located. For example, in the example, the full path of the data file is /elisa0914/dataset1/single_qa_example.jsonl, but you only need to enter the folder name /elisa0914/dataset1/.
- **Dataset Tag:** To facilitate better classification and management of large datasets, you can flexibly and freely assign tags to the datasets. Data Center will support dataset search based on these tags in the future.

Configuring a Schema

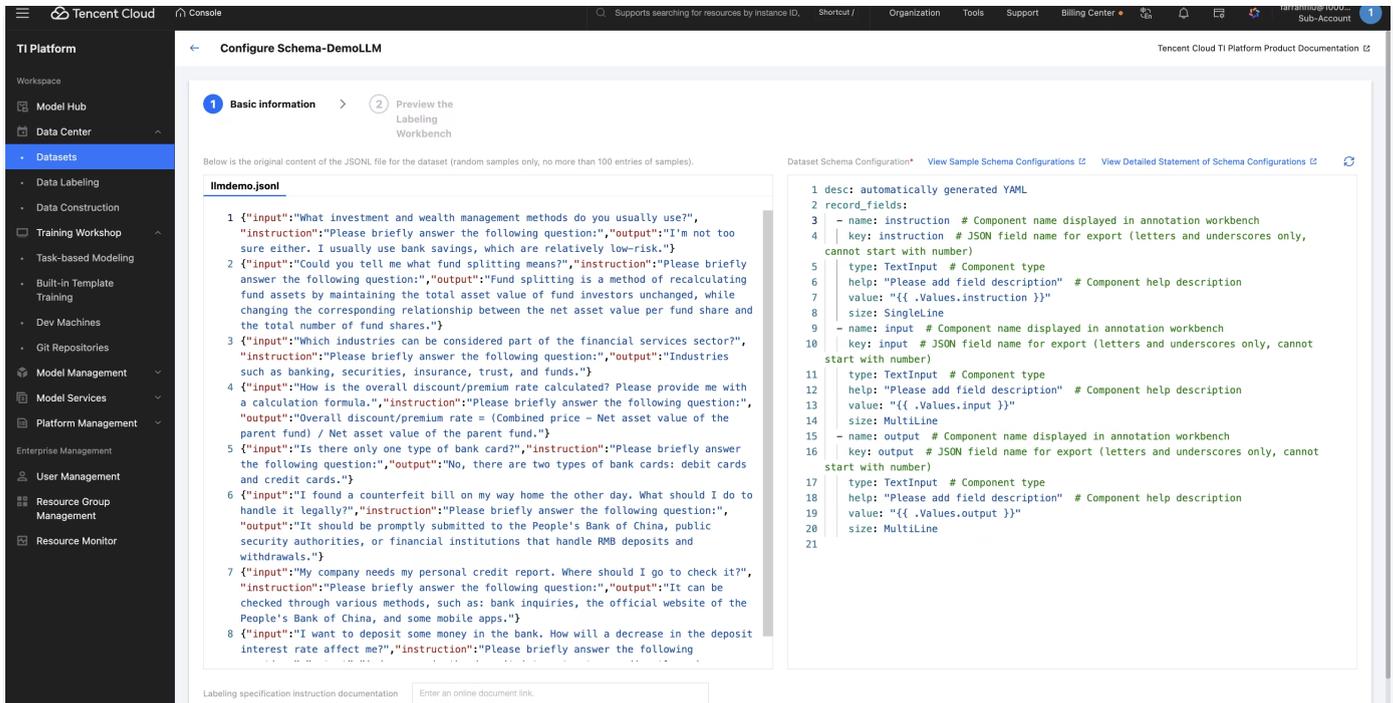
1. Introduction to schema concepts

TI-ONE defines an LLM dataset via schemas. It flexibly defines the type and properties of each field in the dataset via schemas to render the dedicated labeling workbench for each dataset. Specific examples are provided below to describe how to configure schema information. For a more detailed introduction to schema syntax, see [Detailed Syntax of Schema Configuration](#).

2. Specific operations

After a dataset is created, the frontend will automatically return to the dataset list page. At this point, you need to complete configuring the schema information before you click the **name of the dataset** to preview its details and click **Label** to label it. Therefore, you need to first click **Configure Schema** in the **Operation** column.

If you are going to the schema configuration page for the first time, the backend will automatically parse a schema configuration file based on your original JSONL data file by default.



The left side of the page displays part of your original data, making it convenient for you to modify the schema configuration on the right according to the data on the left side. We use the component elements of the first field in the configuration on the right side to describe how to configure the schema information.

```

- name: instruction. # Name of the annotation component displayed on
the annotation console. It supports Chinese characters and letters.

key: instruction. # Field key value saved to the JSON file when the
annotation result of this field is finally exported (it supports letters
and underscores and cannot start with a digit).

type: TextInput. # Defines the component type for this field. For
example, the instruction field here is defined as a text input field.
Valid enumeration values for component types include: TextViewer,
TextInput, StringSelector, ImageViewer, ImageListViewer, ImageListInput,
and List.

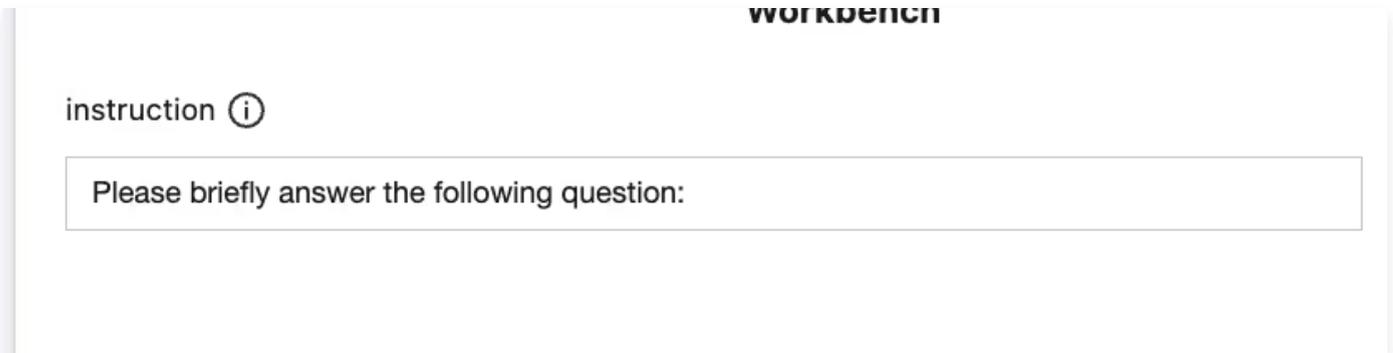
help: "Add a field description." # Component help description,
displayed as a prompt in the annotation console to convey specific
annotation requirements to annotators.

value: "{{ .Values.instruction }}" # Source of the default value for
this field. For example, the source of the default value here is the
instruction field in the data file.

```

```
size: SingleLine. # Defines the size of the text component for this
field. Different size settings result in different component sizes
rendered in the annotation console. If the size is less than 50 bytes, it
is recommended to set it to SingleLine. If the size is 50-300 bytes, it is
recommended to set it to MultiLine. If the size is over 300 bytes, it is
recommended to set it to LongArticle for rendering.
```

The rendering effect of the above component in the labeling workbench is as follows:

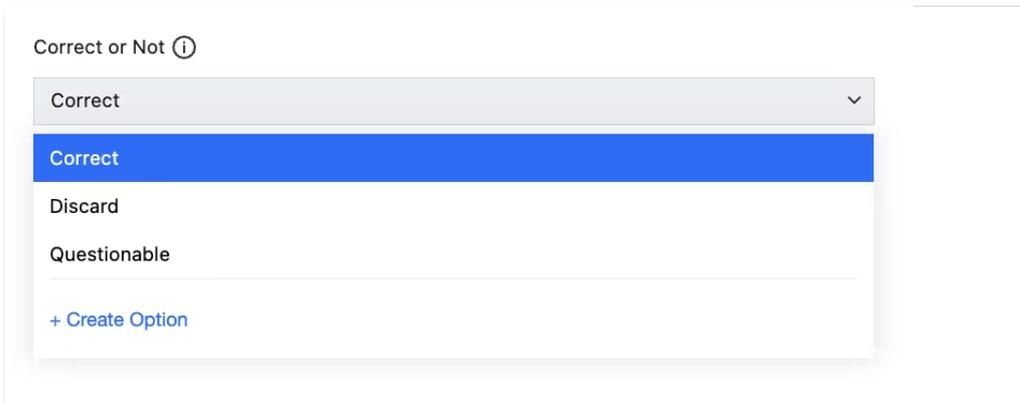


In addition to the schema field information automatically parsed by the platform for you by default, since the requirement of this labeling task is "to determine the description accuracy of this sample (as correct, discard, questionable)", you also need to manually add an labeling about whether the record of this component is correct or not:

```
- name: correct or not. # Add an annotation field to annotate whether
the record of this sample is correct or not.
key: correct. # JSON field key corresponding to this component when
the result of the added field is annotated in the exported annotation
result JSON file.
type: StringSelector. # Indicates that this component type is a
string selector.
help: Determine whether the answer is correct. # Component help
description, displayed as a prompt in the annotation console to convey
specific annotation requirements to annotators.
option: SingleSelector. # Indicates that this component allows only
one choice. Valid enumeration values for component types include:
SingleSelector/MultiSelector.
choices: # Specifies the content of the specific single choice.
- Correct
- Discard
- Questionable
value: # Specifies the default selected choice(s) for this selector.
The value is an array type. A single-choice component allows only one
```

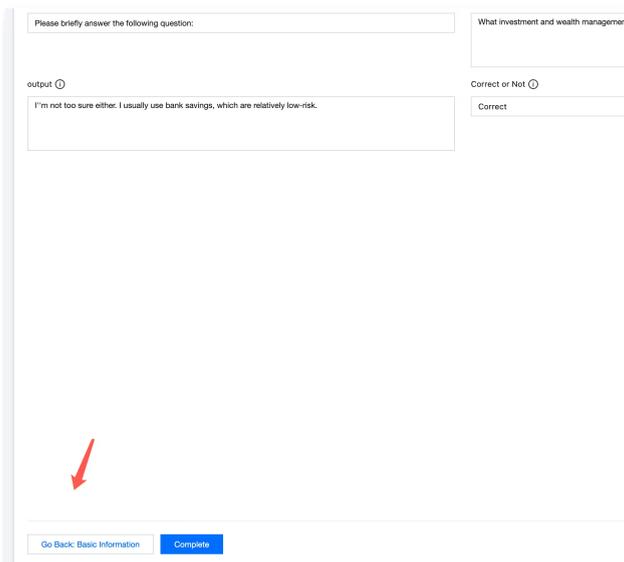
choice, while a multi-choice component allows multiple choices.
– Correct

The rendering effect of the above component in the labeling workbench is as follows:



During schema configuration, you can also click **Next: Preview the Labeling Workbench** to view the schema configuration effect in real time and ensure that the configuration meets your expectations. If you find that the labeling workbench does not meet your expectations, you can click **Go Back: Basic Information** to modify the configuration.

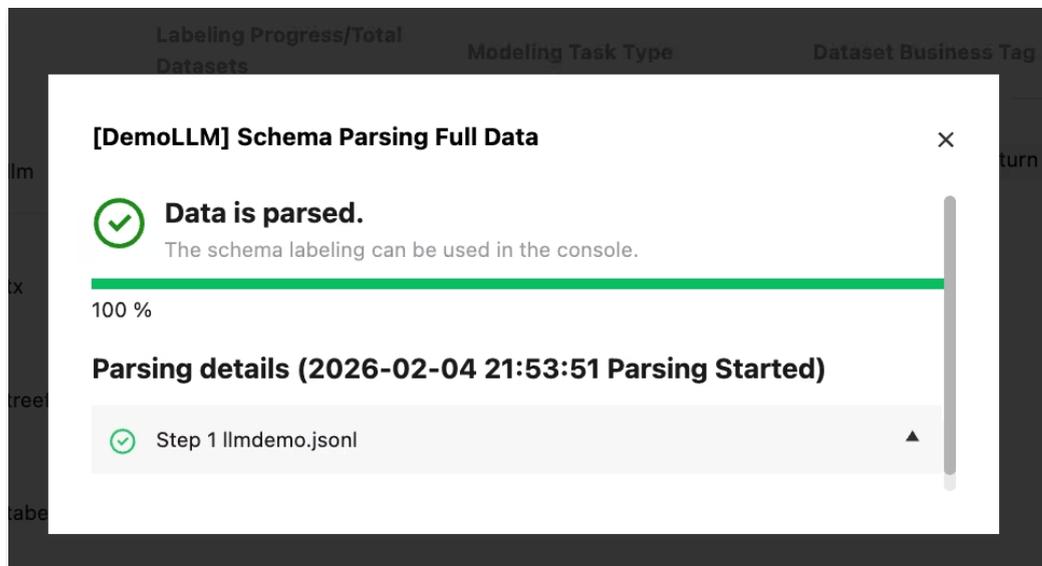
The screenshot shows the "Configure Schema-DemoLLM" interface. The left sidebar contains navigation options: Model Hub, Data Center, Datasets, Data Labeling, Data Construction, Training Workshop, Task-based Modeling, Built-in Template Training, Dev Machines, Git Repositories, Model Management, Model Services, Platform Management, Enterprise Management, User Management, Resource Group Management, and Resource Monitor. The main content area is divided into two panes. The left pane shows the "Basic information" tab with a list of 8 sample questions and their corresponding instructions and expected outputs. The right pane shows the "Dataset Schema Configuration" tab with a JSON schema configuration. A red box highlights the configuration for the "Correct or Not" component, which is a StringSelector with a help text and a value array containing "Correct", "Discard", "Questionable", and "Correct".



See [Detailed Syntax of Schema Configuration](#) for the syntax of all schema components, in addition to the components used in this example.

Labeling a Dataset

After the schema of the dataset is successfully configured, the page will automatically return to the dataset list. At this point, the platform will parse all samples in the data files based on the schema information you have configured. You can click **View Progress** in the Status column on the list page to view the parsing progress of all samples in real time.

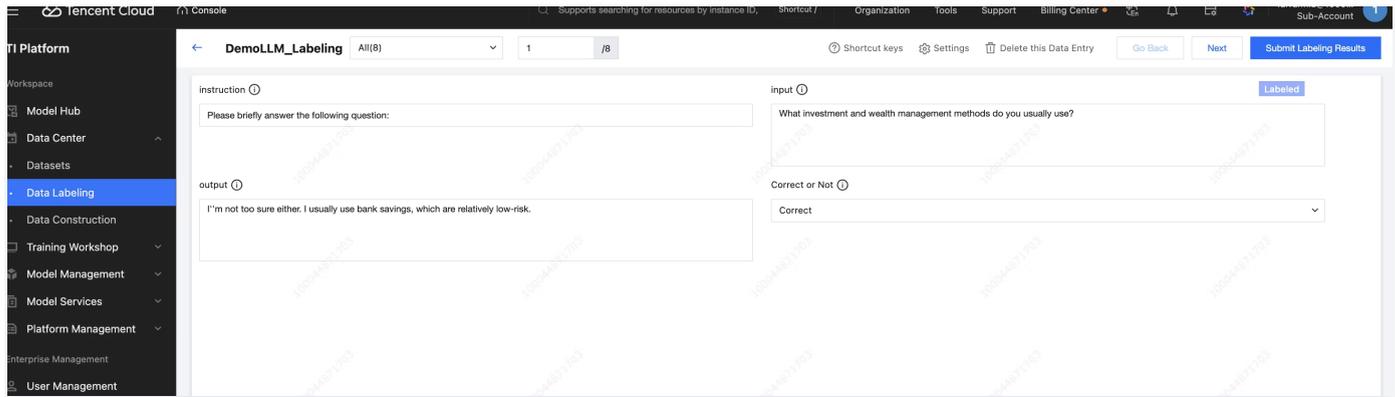


After parsing is completed, you can click **Label** in the **Operation** column to perform labeling. After you click the button, the platform will automatically render the dedicated labeling workbench for this dataset based on the schema you have configured. This process may take varying amounts of time depending on the size of the data.

Time consumption test data for a dataset is as follows:

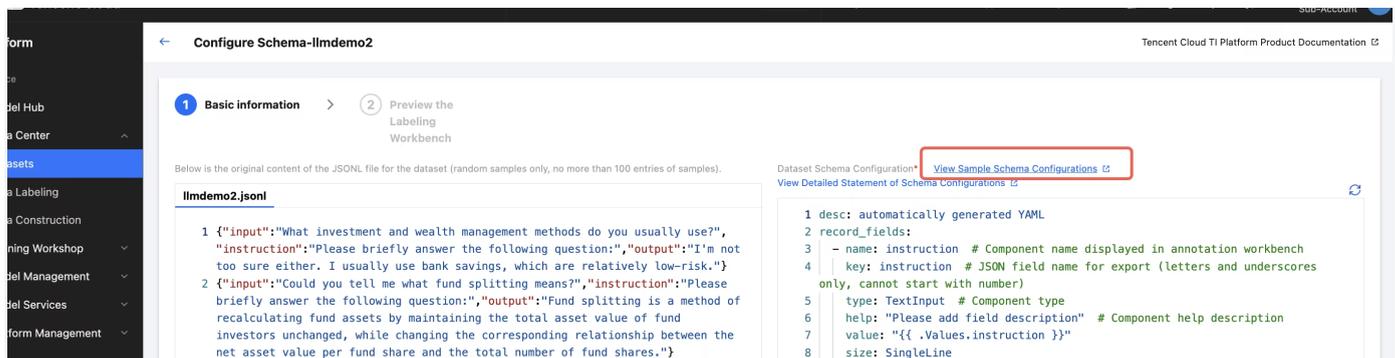
For 106 MB of text data (15,000 samples), the estimated time consumption is approximately 6 minutes for full schema parsing and 3 minutes for labeling task creation.

The dedicated labeling workbench for this dataset is as follows. Note that you must click **Submit Labeling Results**. Only after you click the **submit** button will the labeling result be directly written back to the corresponding CFS data files of this dataset.



Others: Labeling MLLM Data

The above example in this document describes how to create an LLM labeling task. The process of creating an MLLM dataset and labeling task is largely consistent with that of creating an LLM dataset and labeling task. For more examples of MLLM schema configuration, click **View Sample Schema Configurations** on the schema configuration page.



Configure Schema-llmdemo2

Schema Configuration Samples

Filter High-Quality Text Q&A Pairs Add Q&A Pairs Flexibly for a Single Image **Multi-Image Text Mixed Q&A** Document Parsing

Original JSONL sample example This labeling task involves question and answer comprehension of multiple images in the case of hybrid text and images. It is used to train the image understanding capabilities of multimodal large models. Please ensure that the 001.jpg, 002.jpg, and 003.jpg image files and the JSONL file are in the same folder.

```
1 { "images": [ "001.jpg", "002.jpg", "003.jpg" ], "qa_chunks": [ { "question": "Please refer to the first image 001, the second image 002, and the third image 003, and answer the first question based on multiple images", "response": "This is the answer to the first question." } ] }
```

Labeling Workbench Style

Multi-image Visual Question Answering



Upload Image

question Refined Response reponse

Please refer to Image 1, Image 2, and Image 3, and answer the first question based on these multiple images

This is the answer to the first question.

This is the answer to the first question.

schema Copy

```
1 desc: Q&As with text and multiple images.
2 | record_fields:
3 # The first component definition displayed on the labeling workbench
4 - name: images # Name of the labeling component
5 key: images # The JSON field key corresponding to the image list
6 help: Please refer to the picture below # Component help description
7 type: ImageListInput # Indicates that the component is an image list
8 value:
9 # Use a circular reference image list.
10 {{- range .Values.images }} # Indicates that the path list is expanded
11 - {{ . }}
12 {{- end }}
13 # The second component definition displayed on the labeling workbench
14 - name: Q&A pair # Name of the labeling component
15 key: qa_chunks # The JSON field key corresponding to the Q&A pair
16 type: List # Indicates that the component is a list
17 help: Please correct the answer according to the question and answer
18 value:
19 # Use loop expanding on the list content. Each item in the list is a Q&A pair
20 {{- range .Values.qa_chunks }} # Indicates that the qa_chunks list is expanded
```

In MLLM scenarios, the platform supports the following image formats: JPG, JPEG, PNG, and BMP. As for the supported image size, it is recommended that the image size does not exceed 10 MB.

Detailed Syntax of Schema Configuration

Last updated: 2026-02-11 18:29:02

1. Background

(1) JSONL Files

JSON Lines (JSONL) files are a text file format that stores data in JSON format. Unlike standard JavaScript Object Notation (JSON) files, each line in a JSONL file is a separate JSON object. This format makes data easy for humans to read and for computers to parse, and can be easily used with command line tools and scripts.

A standard JSON file typically contains an array or object, which comprises multiple data items encoded as a single JSON structure. In a JSONL file, however, each line is a JSON object, with each object separated by a line break. This makes the JSONL file particularly suitable for logging and stream data processing, as data can be easily read and processed line by line.

Below is an example of a standard JSONL file:

```
{"name": "Alice", "age": 25}
{"name": "Bob", "age": 30}
```

Each JSON object is independent and can be parsed separately. This format is particularly useful for processing large volumes of data, as it allows for line-by-line reading and processing without loading the entire file into memory at one time.

In the LLM field, JSONL is a very common dataset file format.

(2) Basic Structure of JSON

JSON is a lightweight data exchange format that is easy for humans to read and write, and also easy for computers to parse and generate. It is based on a subset of JavaScript, but it is completely language-independent. Many programming languages support exchanging data in JSON format.

Object: enclosed in curly braces (`{ }`) and consisting of a series of key-value pairs separated by commas (`,`). Example:

```
{
  "name": "Alice",
  "age": 25,
  "isStudent": false
}
```

```
}
```

Array: enclosed in square brackets (`[]`) and can contain any number of values. These values can be strings, digits, objects, arrays, and so on, separated by commas (`,`). Example:

```
["apple", "banana", "cherry"]
```

Key-value pair: The key is a string, and the value can be a string, digit, object, array, Boolean value, or null. The key and value are separated by a colon (`:`). Example:

```
"name": "Alice"
```

(3) YAML Files

We use the YAML Ain't Markup Language (YAML) file format to define our labeling template. The syntax of the YAML file is described below.

YAML is a human-readable data serialization format, commonly used for configuration files, data exchange, and object transfer between different programming languages. YAML is designed to be easy for humans to read and write and for computers to parse and generate.

The basic structure of a YAML file includes the following characteristics:

1. **Key-value pair:** YAML uses a colon followed by a space (`:`) to separate a key and a value.
2. **Indentation:** YAML uses indentation to represent hierarchical relationships. It typically uses the space, rather than the tab character (Tab) for indentation.
3. **List:** In YAML, lists start with a hyphen (`-`), and each list item occupies a separate line.
4. **Dictionary/Mapping:** Dictionaries or mappings in YAML are represented using key-value pairs and can be nested.
5. **Comment:** Comments in YAML start with a hash (`#`) and continue until the end of the line.
6. **Multi-document support:** YAML can contain multiple documents in a single file, with each document separated by three consecutive hyphens (`---`).
7. **Data types:** YAML supports multiple data types, including strings, Boolean values, integers, floating-point numbers, dates, and time.
8. **Complex structures:** YAML supports complex data structures, such as nested lists and dictionaries.

Below is a simple example of a YAML file:

```
# This is a comment.  
name: John Doe  
age: 30  
married: true
```

```
children:
  - name: Jane Doe
    age: 10
  - name: Jim Doe
    age: 8
address:
  street: 123 Main St
  city: Anytown
  zip: 12345
```

In this example, we define a YAML document containing personal information, including a string (`name`), an integer (`age`), a Boolean value (`married`), a list (`children`), and a nested dictionary (`address`).

2. LLM Labeling Scenarios

The Data Center module of Tencent Cloud TI-ONE Platform (TI-ONE) supports the import and labeling of multimodal datasets such as LLMs and MLLMs. Our labeling workbench allows you to customize data parsing and edit labeling pages.

(1) LLM Scenarios

Common labeling in LLM scenarios typically involve labeling single-turn or multi-turn Q&A pairs. Each sample in the raw dataset is a JSON object. Example:

Below is an example of a single-turn Q&A pair:

```
{
  "instruction": "Please briefly answer the following question:"
  "input": "What investment and financial management methods do you usually use?"
  "output": "I generally opt for bank savings, which are relatively low-risk."
}
```

Below is an example of a multi-turn Q&A pair:

```
{
  "conversations": [
    {
      "from": "human",
      "value": "What is insurance? Is it reliable?"
    },
  ],
}
```

```
{
  "from": "model",
  "value": "Insurance is a commercial insurance activity where
the policyholder pays insurance premiums to the insurer according to the
contract agreement, and the insurer bears the liability to pay insurance
compensation for property losses caused by accidents specified in the
contract, or bears the liability to pay insurance benefits when the
insured dies, becomes disabled, falls ill, or meets conditions such as the
age or term specified in the contract. Insurance purchased through legal
channels is safe and reliable."
},
{
  "from": "human",
  "value": "I want to purchase insurance for investment and
financial management. What types of insurance are available?"
},
{
  "from": "model",
  "value": "Annuity insurance, dividend insurance, investment-
linked insurance, and so on."
}
]
```

(2) MLLM Scenarios

Labeling in MLLM scenarios generally involve displaying and processing resources such as images.

In sample data, we typically reference the relative path of images in JSON data and place the images in the corresponding directory.

Below is an example of an image labeling :

```
{
  "img": [
    "images/100230144.jpeg"
  ],
  "target": [
    {
      "question": "Please provide a vivid image description."
      "answer": "The image shows a toy doll inside a packaging box.
The packaging box is primarily white, with a transparent plastic window
```

```
that allows a clear view of the doll inside. The top of the box is printed with the words Pop! Movies, and the bottom says 'Home Alone', indicating that this is a collectible themed after the movie Home Alone. The doll is labeled as Kevin in a clearly visible red font, with a series number of 491. The doll features a simple design and the Funko Pop! style, with large black eyes, a small nose, no mouth, and short hair. He is wearing a red sweater with a fluffy design and green pants. He holds a rifle in his left hand and an iron in his right hand. Both props maintain a minimalist design with few details. The doll stands upright, with brown shoes visible under the green pants. The sides of the packaging box are red, and the front bottom is green, with additional brand and warning tags typically found on collectible packaging, including a warning about potential choking hazards."
```

```
}
```

```
]
```

```
}
```

3. TI-ONE Labeling Workbench Design

To easily modify the sample data in the labeling workbench, we need to convert multiple pieces of information in each sample to multiple labeling components on the frontend page and populate the sample data within the components.

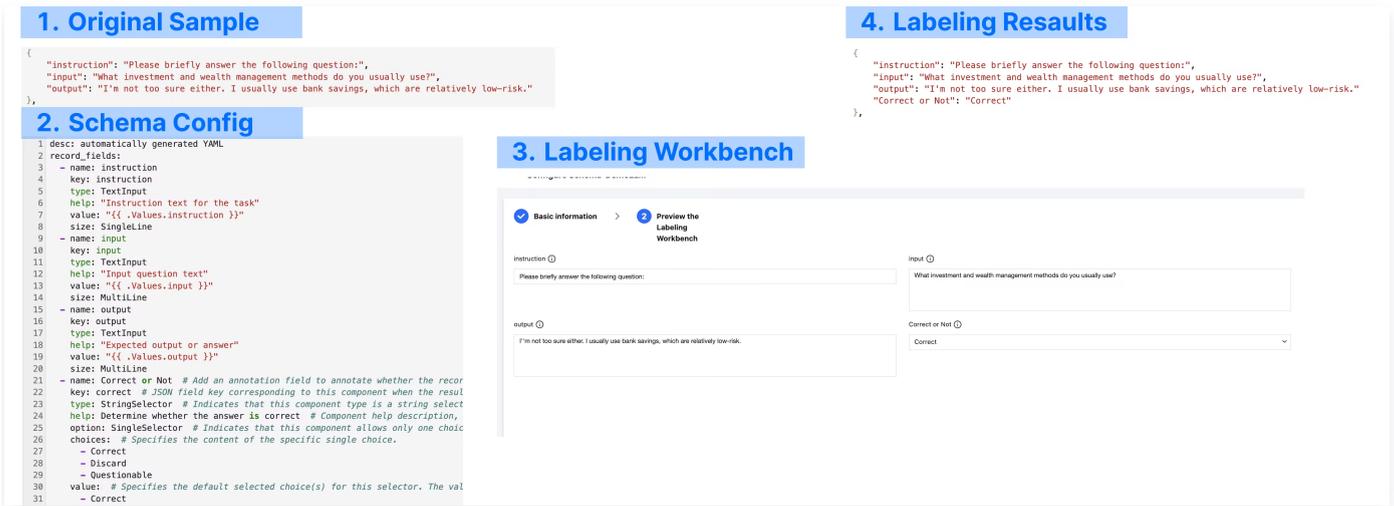
Based on observations and research of the data processing process through internal algorithms, we have designed the dataset schema concept. Through the **schema syntax structure** agreed upon and **built-in component features**, it supports users in easily building labeling pages using raw data and performing editing processes such as data labeling and updates.

The design of labeling pages is divided into two steps:

Step 1: Define the required labeling components based on raw data.

Step 2: Parse data from the raw sample data and populate the corresponding labeling components with it.

Below is an example:



(1) Labeling Component Definition

Based on the review of algorithm labeling scenarios, we have defined the following built-in labeling components:

Labeling	Description	Property	Remarks
TextViewer	Text display box.	type	Component type: TextViewer.
		name	Field name identifier, such as question.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits), such as question.
		help	Labeling recommendation, such as "Enter the answer".
		size	Text box size. Valid values: SingleLine/MultiLine/LongArticle.
		value	Default value.
TextInput	Text input box.	type	Component type: TextInput.
		name	Field name identifier, such as question.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits), such as question.

		help	Labeling recommendation, such as "Enter the recommended answer for the question regarding the image".
		size	Text box size. Valid values: SingleLine/MultiLine/LongArticle.
		value	Default value of the input box.
StringSelector	String type selection button (single-choice/multi-choice).	type	Component type: StringSelector.
		name	Field name identifier.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		option	Single-choice/multi-choice configuration: SingleSelector/MultiSelector.
		help	Labeling recommendation, such as "Determine whether the provided answer is correct".
		choices	Array type; enumeration of selectable choices, such as [Correct, Incorrect, Pending].
		value	Array type; default choice, such as [Correct].
ImageViewer	Single-image display box.	type	Component type: ImageViewer.
		name	Field name identifier.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		help	Labeling recommendation, such as "See the following image(s)".
		value	Image path relative to the dataset, such as cat.jpeg.
ImageListViewer	Multi-image display box.	type	Component type: ImageListViewer.
		name	Field name identifier.

		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		help	Labeling recommendation, such as "See the following image(s)".
		value	Array type, list of image paths relative to the dataset: [cat.jpeg, dog.jpeg].
ImageListInput	Multi-image display box, allowing adding or deleting images (at least one image required).	type	Component type: ImageListInput.
		name	Field name identifier.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		help	Labeling recommendation, such as "See the following image(s)".
		value	Array type, list of image paths relative to the dataset: [cat.jpeg, dog.jpeg].
List	Component list type. Each item in the list can contain multiple components.	type	Component type: List.
		name	Field name identifier.
		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		help	Labeling recommendation, such as "See the following graphic and textual information".
		value	[[field: double-layer nested list, where each element is a [field. Elements are distributed horizontally within each inner list and vertically between inner lists.
ImageBoxLayout	Multiple labeling boxes on the image.	type	Component type: ImageBoxLayout.
		name	Field name identifier.

		key	Field corresponding to the JSON key in the output file after labeling (letters, underscores, and digits).
		help	Labeling recommendation, such as "Annotate the content on the image".
		value	[[field: double-layer nested list where each element is a []field containing a field to be labeled on the box (the content to be labeled can only be TextInput/StringSelector/Box).
Box	Coordinates of the labeling box. They are implicitly generated when the box is drawn in the labeling workbench, and do not need to be manually specified.	type	Component type: Box.
		name	Field name identifier.
		key	The JSON key for boxes in the input file must be the fixed value "box".
		help	Labeling recommendation, such as "Box the content on the image".
		value	String obtained from JSON serialization of box coordinates.

Below is a detailed description of the YAML configuration for each labeling component.

TextViewer

Read-only text display box, which is non-modifiable.

```
# name: field name identifier, with a maximum length of 100 bytes.
name: question.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: question
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: TextViewer
# size: text box size. Valid values: SingleLine/MultiLine/LongArticle.
size: MultiLine
```

```
# value: syntax for this component to extract values from the sample data.
value: "{{ .Values.question }}"
```

TextInput

Editable text display box.

```
# name: field name identifier, with a maximum length of 100 bytes.
name: question.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: question
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: TextInput
# size: text box size. Valid values: SingleLine/MultiLine/LongArticle.
size: MultiLine
# value: syntax for this component to extract values from the sample data.
value: "{{ .Values.xxxx }}"
```

StringSelector

String type selection button (single-choice/multi-choice).

```
# name: field name identifier, with a maximum length of 100 bytes.
name: question.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: question
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: StringSelector
# option: specifies whether only one choice or multiple choices can be
selected. SingleSelector specifies that only one choice can be selected,
while MultiSelector specifies that multiple choices can be selected.
```

```
option: SingleSelector
# choices: list of selectable choices.
choices:
  - Correct
  - Incorrect
  - Pending
# value: selected value. For a single-choice component, the value is an
array with a length of 1. For a multi-choice component, the length can be
greater than or equal to 1.
value:
  - Correct
```

ImageViewer

Single-image display box, which is not editable.

```
# name: field name identifier, with a maximum length of 100 bytes.
name: single image.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: imgpath
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: ImageViewer
# value: relative path of the image. The value is extracted from imgpath
of the sample data.
value: "{{ .Values.imgpath }}"
```

ImageListViewer

Multi-image display box.

```
# name: field name identifier, with a maximum length of 100 bytes.
name: image list.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: imgs
```

```
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: ImageListViewer
# value: relative path of the list image. The value is extracted from imgs
of the sample data.
value:
  # Use a circularly referenced image list.
  {{- range .Values.imgs }}
  - {{ . }}
  {{- end }}
```

ImageListInput

Multi-image display box, allowing adding or deleting images (at least one image required).

```
# name: field name identifier, with a maximum length of 100 bytes.
name: image list (modifiable).
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: imgs
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: ImageListInput
# value: relative path of the list image. The value is extracted from imgs
of the sample data.
value:
  # Use a circularly referenced image list.
  {{- range .Values.imgs }}
  - {{ . }}
  {{- end }}
```

List

Component list type, used for the layout of complex data types.

The value of the component is a double-layer nested list, where each element is a list of components. Elements are distributed horizontally within each inner list and vertically between

inner lists.

Note: Nested List/ImageListViewer/ImageListinput components are currently not supported for List.

```
# name: field name identifier, with a maximum length of 100 bytes.
name: Q&A pair.
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: qa_chunks
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: List
# value: syntax for this component to extract values from the sample data:
[[]]field double-layer nested list, where each element is a []field.
value:
  # Use a loop to unfold the list, where each item is a list of
  components. Elements are distributed vertically between items and
  horizontally within items.
  {{- range .Values.qa_chunks }} # Indicates iterating over the
qa_chunks array within a single JSON sample.
    # First component within a list item.
    - name: question. # Name of the annotation component
displayed in the annotation console.
      key: question. # JSON field key corresponding to
this component when JSON annotation results are exported.
      type: TextViewer. # Indicates that this component type
is a non-editable text display box.
      size: MultiLine. # Indicates that this field is a
multi-line text box. Valid values: SingleLine/MultiLine/LongArticle.
      value: "{{ .question }}". # Indicates that the value is
extracted from the question field of a single element in the qa_chunks
array within a single JSON sample.
    # Second component within a list item.
    - name: modified response. # Name of the annotation
component displayed in the annotation console.
      key: modified_response. # JSON field key corresponding to
this component when JSON annotation results are exported.
```

```

    type: TextInput.                # Indicates that this component
type is a text input box.
    size: MultiLine.                # Indicates that this field is a
multi-line text box. Valid values: SingleLine/MultiLine/LongArticle.
    value: "{{ .response }}"        # Indicates that the default value
of the text input box is extracted from the response field of a single
element in the qa_chunks array within a single JSON sample.
# Third component within a list item.
- name: response.                  # Name of the annotation component
displayed in the annotation console.
    key: response.                  # JSON field key corresponding to
this component when JSON annotation results are exported.
    type: TextViewer.              # Indicates that this component
type is a non-editable text display box.
    size: MultiLine.                # Indicates that this field is a
multi-line text box. Valid values: SingleLine/MultiLine/LongArticle.
    value: "{{ .response }}"        # Indicates that the value is
extracted from the response field of a single element in the qa_chunks
array within a single JSON sample.
{{- end }}

```

ImageBoxList

Labeling box on the image. The labeling field can be customized.

If there are only images and no pre-labeled data, manually add a list that defines the fields to be labeled:

```

# name: field name identifier, with a maximum length of 100 bytes.
name: image_box_list
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: image_box_list
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: ImageBoxList
# value: syntax for this component to extract values from the sample data:
[[]]field double-layer nested list, where each element is a []field.
value:

```

```

- # Defines the fields to be annotated.
- name: text content. # Name of the annotation component
displayed in the annotation console.
  key: text. # JSON field key corresponding to this
component when JSON annotation results are exported.
  type: TextInput. # Text input box.
  size: LongArticle. # Indicates that this field is a multi-line
text box. Valid values: SingleLine/MultiLine/LongArticle.
  value: "". # If pre-annotation is not performed, each
field can be assigned an empty value.
- name: text type. # Name of the annotation component
displayed in the annotation console.
  key: type. # JSON field key corresponding to this
component when JSON annotation results are exported.
  type: StringSelector. # String selector.
  option: SingleSelector. # Single-choice or multi-choice.
  choices: # Candidate items, which are a list.
    - Main text
    - Headline 1
    - Table
  value: # Selected value. For a single-choice
component, the value is an array with a length of 1. For a multi-choice
component, the length can be greater than or equal to 1.
    - "" # If pre-annotation is not performed, an
empty value can be assigned.

```

If there is pre-labeled data, in addition to the labeling fields, the extraction of box coordinates needs to be configured:

```

# name: field name identifier, with a maximum length of 100 bytes.
name: image_box_list
# key: field corresponding to the JSON key in the output file after
annotation (supports only letters, underscores, and digits, with a maximum
length of 100 bytes).
key: image_box_list
# help: detailed field description, used to display help information on
the annotation console.
help: See the following questions.
# type: component type.
type: ImageBoxList

```

```
# value: syntax for this component to extract values from the sample data:
[[[]]field double-layer nested list, where each element is a []field.
value:
  {{- range .Values.image_box_list}}.          # Indicates extracting
  annotations from the image_box_list array within a single JSON sample.
-                                              # Defines the fields to
be annotated.
  - name: text content.                        # Name of the
  annotation component displayed in the annotation console.
    key: text.                                 # JSON field key
  corresponding to this component when JSON annotation results are exported.
    type: TextInput.                          # Text input box.
    size: LongArticle.                        # Indicates that this
  field is a multi-line text box. Valid values:
  SingleLine/MultiLine/LongArticle.
    value: "{{ .text }}"
  - name: text type.                           # Name of the
  annotation component displayed in the annotation console.
    key: type.                                 # JSON field key
  corresponding to this component when JSON annotation results are exported.
    type: StringSelector.                     # String selector.
    option: SingleSelector.                   # Single-choice or
  multi-choice.
    choices:                                  # Candidate items, which
  are a list.
      - Main text
      - Headline 1
      - Table
    value:                                     # Selected value. For a
  single-choice component, the value is an array with a length of 1. For a
  multi-choice component, the length can be greater than or equal to 1.
      - Main text
  - name: box coordinates.                     # Name of the
  annotation component displayed in the annotation console.
    key: text.                                 # JSON field key
  corresponding to this component when JSON annotation results are exported.
    type: Box.                                # Box coordinates.
    value: "{{ .box }}"
```

Box

Labeling box coordinates, which can only be used in ImageBoxList. For specific usage, see ImageBoxList.

(2) Parsing Syntax Definition

The essence of parsing syntax is to define how to extract values from JSON-formatted key values and populate template definitions with them.

For parsing syntax, we use the YAML rendering syntax of the Helm Chart based on Go template syntax.

For template syntax, we use double curly braces (`{{ }}`) to wrap template instructions.

The root node is `.Values`, which contains the values of each record in the user-provided JSONL sample file.

```
data:
  myvalue: {{ .Values.myvalue }}
```

Basic Value Extraction

An object for a line of JSON sample is as follows:

```
{
  "question": "question",
  "answer": "answer"
}
```

We can use the following value extraction syntax:

```
question: {{ .Values.question }}
answer: {{ .Values.answer }}
```

Value Extraction of Array Elements

If JSON is an array, we need to extract the first element of the array.

An object for a line of JSON sample is as follows:

```
{
  "question_list": [
    "question1",
    "question2",
    "question3"
  ]
}
```

```

]
}

```

We can use the following value extraction syntax:

```

question: {{ index .Values.question_list 0 }} # Index 0 represents the
first element.

```

Converting an Array to a Template Array

If JSON is an array, we need to extract all elements of the array and convert them to a list within the YAML template.

An object for a line of JSON sample is as follows:

```

{
  "qa_list": [
    {
      "question": "question1",
      "answer": "answer1"
    },
    {
      "question": "question2",
      "answer": "answer2"
    },
    {
      "question": "question3",
      "answer": "answer3"
    }
  ]
}

```

We can use the following range value extraction syntax:

```

{{- range .Values.qa_list }}
# Note: Internal value extraction is a path relative to range qa_list.
- question: {{ .question }}
  answer: {{ .answer }}
{{ end }}

```

After template rendering:

```
- question: question1
  answer: answer1
- question: question2
  answer: answer3
- question: question3
  answer: answer3
```

Generally, the above syntax can meet basic requirements for schema definition. To handle more complex business data, define your schema based on the Helm Chart syntax.

(3) Schema Template Definition

By concatenating the above components and defining variables using template syntax, we can define our schema template.

Scenario: Filtering High-Quality Text Q&A Pairs

For example, each line in our sample is a Q&A pair, as follows:

```
{
  "question": "What investment and financial management methods do you
usually use?"
  "answer": "I generally opt for bank savings, which are relatively low-
risk. Occasionally, I also want to try financial management methods that
may have higher risks but can yield higher returns, but I do not know how
to choose."
}
```

We define the following labeling schema:

```
desc: Filter high-quality LLM training data.
record_fields:
  # Definition of the first component displayed in the annotation console.
  - name: question. # Name of the annotation component
displayed in the annotation console.
  key: question. # JSON field key corresponding to
this component when JSON annotation results are exported.
  help: See the following questions. # Component help
description.
  type: TextViewer. # Indicates that this component type
is a non-editable text display box.
```

```
    size: SingleLine.                # Indicates that this field is a
single-line text box. Valid values: SingleLine/MultiLine/LongArticle.
    value: "{{ .Values.question }}" # Indicates that the content of this
field is sourced from the question field within a single JSON sample.
# Definition of the second component displayed in the annotation
console.
- name: answer.                    # Name of the annotation component
displayed in the annotation console
  key: answer.                     # JSON field key corresponding to
this component when JSON annotation results are exported.
  type: TextInput.                 # Indicates that this component type
is a text input box.
  help: Correct the answer to the question. # Component
help description.
  size: MultiLine.                # Indicates that this field is a
multi-line text box. Valid values: SingleLine/MultiLine/LongArticle.
  value: "{{ .Values.answer }}".   # Indicates that the default content
of this field is sourced from the answer field within a single JSON
sample.
# Definition of the third component displayed in the annotation console.
- name: correct or not.            # Name of the annotation
component displayed in the annotation console.
  key: correct.                   # JSON field key corresponding to
this component when JSON annotation results are exported.
  type: StringSelector.           # Indicates that this component type
is a string selector.
  help: Determine whether the answer is correct. # Component
help description.
  option: SingleSelector.         # Indicates that this component
allows only one choice. Valid values: SingleSelector/MultiSelector.
  choices:                        # Specifies the content of the
choices.
    - Correct
    - Discard
    - Questionable
  value:                          # Specifies the default selected
choice(s). The value is an array type. A single-choice component allows
only one choice, while a multi-choice component allows multiple choices.
    - Correct
# Definition of the fourth component displayed in the annotation
console.
```

```

- name: reasons for Discard or Questionable.           # Name of
the annotation component displayed in the annotation console.
  key: correct_reason.                                # JSON field key corresponding to
this component when JSON annotation results are exported.
  type: StringSelector.                               # Indicates that this component type
is a string selector.
  option: MultiSelector.                             # Indicates that this component
allows multiple choices. Valid values: SingleSelector/MultiSelector.
  help: reasons for Incorrect.                       # Component help description.
  choices:                                           # Specifies the content of the
choices.
    - No error
    - Logical error
    - Answer not relevant to the question
    - Missing content
  value:                                             # Specifies the default selected
choice(s). The value is an array type. A single-choice component allows
only one choice, while a multi-choice component allows multiple choices.
    - Logical error
    - Missing content

```

The labeling workbench and content can finally be rendered as follows:

The screenshot displays a web-based labeling interface with the following components:

- instruction**: A text box containing the prompt "Please briefly answer the following question:".
- input**: A text box containing the question "What investment and wealth management methods do you usually use?". A blue "Labeled" tag is visible in the top right corner of this box.
- output**: A text box containing the user's response: "I'm not too sure either. I usually use bank savings, which are relatively low-risk."
- Correct or Not**: A dropdown menu currently showing "Correct".

Practices Related to Embodied AI Cosmos-Predict2 Inference and Fine-Tuning Guide

Last updated: 2026-02-26 15:21:29

Scenario Description

Cosmos-Predict2 is an open-source, physical AI world foundation model developed by NVIDIA, part of the Cosmos world foundation model series. It is specifically designed to generate physically plausible future world states from text, images, or videos, providing synthetic data and simulation environments for physical AI systems such as robotics and autonomous driving. This document describes how to deploy and fine-tune the Cosmos-Predict2 series models on Tencent Cloud TI-ONE Platform (TI-ONE).

Prerequisites

1. [Download checkpoints.](#)
2. [Download datasets.](#)

After the download is completed, the checkpoints directory structure is as follows:

```
|— depth-anything
|   └─ Depth-Anything-V2-Small-hf
|— facebook
|   └─ sam2-hiera-large
|— google-t5
|   └─ t5-11b
|— IDEA-Research
|   └─ grounding-dino-tiny
|— meta-llama
|   └─ Llama-Guard-3-8B
└─ nvidia
    └─ Cosmos-1.0-Guardrail
        └─ Cosmos-Guardrail1
            └─ Cosmos-Predict2-14B-Text2Image
            └─ Cosmos-Predict2-14B-Video2World
            └─ Cosmos-Predict2-2B-Text2Image
            └─ Cosmos-Predict2-2B-Video2World
            └─ Cosmos-Reason1-7B
```

```
├─ Cosmos-Tokenize1-CV8x8x8-720p
├─ Cosmos-Transfer1-7B
├─ Cosmos-Transfer1-7B-Sample-AV
├─ Cosmos-Transfer1-7B-Sample-AV-Single2MultiView
├─ Cosmos-UpsamplePrompt1-12B-Transfer
```

- The NVIDIA directory contains additional models, such as transfer1, which is a superset of the Cosmos series.
- Ensure that you have correctly passed in the hf token and agreed to the user agreement for other directories, such as directories of the meta-llama series model files. Otherwise, the download may fail.

Model Deployment

Step 1: Confirming Required Resources

The recommended resources for 14B-Text2World deployment are as follows. You can select based on your available resources.

- Standard deployment: 31 cores and 288 GB GPU-HCCPNV6 x 2
- Standard deployment: 60 cores and 600 GB GPU-HCCPNV6 x 4
- Standard deployment: 125 cores and 1207 GB GPU-HCCPNV6 x 8

Step 2: Creating an Online Service

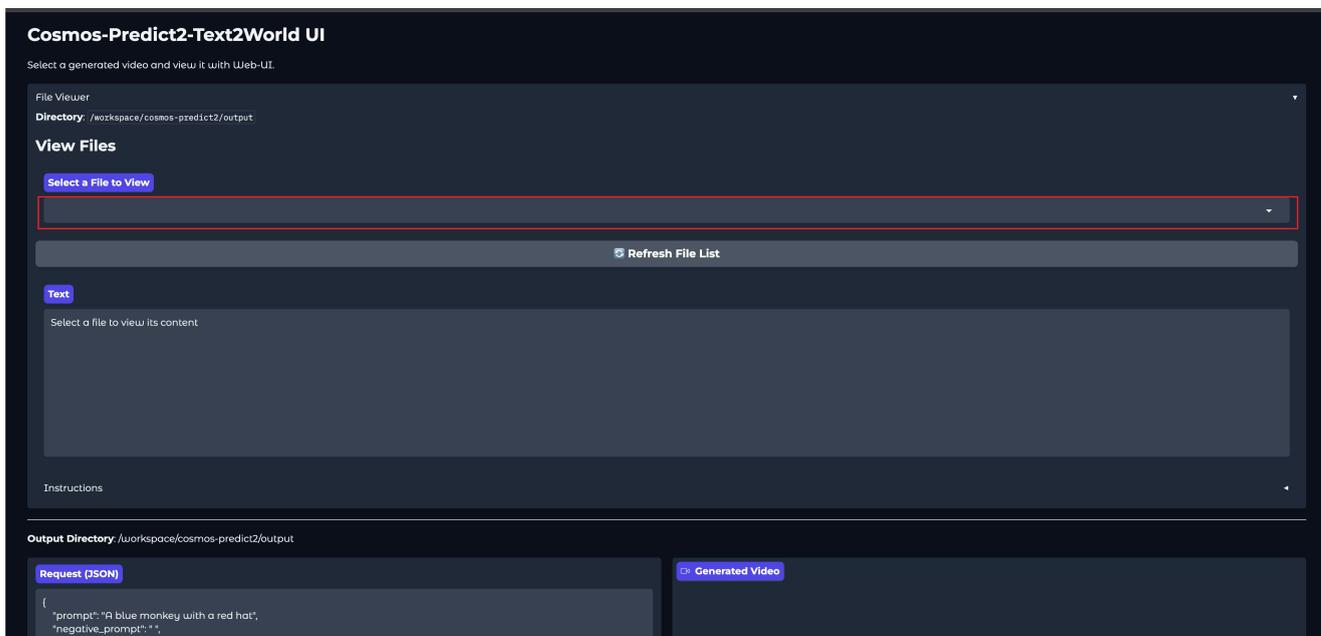
- **Model Source:** Select a Tencent Cloud storage product and the checkpoints path downloaded in the Prerequisites section.
- **Image:** Select built-in `PyTorch/cosmos-predict2-cu128`.
- **Storage Mount:** Configure the container path `/opt/ml/output` corresponding to the output video directory. When the inference container is started, this path is symbolically linked to `/workspace/cosmos-predict2/output`. Therefore, the path shown in the following WebUI is the symbolic link and does not affect the output.
- **Resource Application:** Select the solution based on your business requirements and the recommended resources above.
- **Environment Variables:**
 - **MODEL_SIZE:** Valid values include 2B or 14B. The default value is 14B.
 - **DISABLE_GUARDRAIL:** disables the guardrail model check. It is not enabled by default. Setting the variable to any value will disable the option. This variable is optional for 2B models, and is required for 14B models. Otherwise, it may easily cause out-of-memory (OOM) and CUDA OOM errors.

- **DISABLE_PROMPT_REFINER**: disables the prompt optimization feature. It is not enabled by default, and can optimize short prompts. Setting the variable to any value will disable the option. This parameter is optional for 2B models, and is required for 14B models. Otherwise, it may easily cause OOM and CUDA OOM errors.
- **OFFLOAD_GUARDRAIL**: offloads the guardrail model to memory to reduce GPU memory usage. It is not enabled by default. Setting the variable to any value will disable the option. This option is required for 14B models with **DISABLE_GUARDRAIL** unconfigured. The memory is increased by one level compared to that of the recommended resource configuration.
- **OFFLOAD_PROMPT_REFINER**: offloads the prompt optimization model to memory to reduce GPU memory usage. It is not enabled by default. Setting the parameter to any value will disable the option. This option is required for 14B models with **DISABLE_PROMPT_REFINER** unconfigured. The memory is increased by one level compared to that of the recommended resource configuration.

Step 3: Viewing the WebUI

Wait for the service to start up, and then click **Open WebUI Page**.

Expand File Viewer to preview all videos and files under `/workspace/cosmos-predict2/output`.



Initiate requests and view the generated videos below.

Performance Tips

1. Model service startup with single-GPU card HCCPNV6: When `image_num_sampling_step` is set to 35, and `video_num_sampling_step` is set to 5, a single run takes approximately 13

minutes. Therefore, the time required to complete inference using the default value of 35 is calculated as follows: $13 \times 7 = 91$ minutes.

2. Currently, multi-GPU card CP inference acceleration is supported. You can select an appropriate solution based on the recommended resource configurations above and your business requirements. Taking an 8-GPU card HCCPNV6 as an example, when `image_num_sampling_step` and `video_num_sampling_step` are set to 35 by default, the time required to complete a request is calculated as follows: $91/8 \approx 12$ minutes.
3. If the generation time is too long and the GPU cards are insufficient, you can reduce the value of `video_num_sampling_step` to proportionally decrease the task duration significantly. However, this will also degrade video quality.

Model Fine-Tuning

Step 1: Configuring a Storage Path

1. Both Dev Machines and Task-based Modeling modules require mounting directories to the corresponding paths. In the Task-based Modeling module, you can customize the configuration as instructed in the guide below, while in the Dev Machines module, you need to create a symbolic link by yourself.
2. The corresponding directory requirements are as follows:
 - Mount the checkpoints directory downloaded in the Prerequisites section to `/workspace/cosmos-predict2/checkpoints`.
 - Preprocess the datasets directory downloaded in the Prerequisites section as instructed in the guide below, and mount it to `/workspace/cosmos-predict2/datasets`.
 - If both Text2Image and Video2World have been preprocessed, the final directory structure of `/workspace/cosmos-predict2/datasets` is as follows:

```
├─ cosmos_nemo_assets
│  ├─ metas
│  ├─ t5_xx1
│  └─ videos
└─ cosmos_nemo_assets_images
   ├─ images
   ├─ metas
   └─ t5_xx1
```

Step 2: Fine-Tuning 14B-Text2Image

See [Text2Image Post-training for Cosmos-NeMo-Assets](#).

(Download and process the datasets as instructed in the guide before proceeding with the following steps.)

- Training module: Dev Machines
- Training resource configuration: 128 cores and 1024 GB HCCPNV6 x 8
- Training configuration: `/workspace/cosmos-predict2/cosmos_predict2/configs/base/experiment/cosmos_nemo_assets.py`
- Run the training command in the Dev Machines module:

```
cd /workspace/cosmos-predict2
source $HOME/.local/bin/env && source .venv/bin/activate
export HF_ENDPOINT=https://hf-mirror.com

torchrun --nproc_per_node=8 --master_port=12341 -m scripts.train --
config=cosmos_predict2/configs/base/config.py --
experiment=predict2_text2image_training_14b_cosmos_nemo_assets
```

If you do not modify the configuration, the default output path is `/workspace/cosmos-predict2/checkpoints/posttraining/text2image`. See the files corresponding to the above training configurations to modify parameters such as epoch and learning rate.

- Perform inference testing after training:

```
python examples/text2image.py \
  --model_size 14B \
  --dit_path
  "checkpoints/posttraining/text2image/14b_cosmos_nemo_assets/checkpoints/model/iter_000001000.pt" \
  --prompt "An image of sks teal robot." \
  --save_path output/generated_image_14b_teal_robot.jpg
```

Step 3: Fine-Tuning 14B-Video2World

See [Text2Image Post-training for Cosmos-NeMo-Assets](#) .

(Download and process the datasets as instructed in the guide before proceeding with the following steps.)

- Training module: Task-based Modeling
- Training resource configuration: 128 cores and 1024 GB HCCPNV6 x 8; nodes x 2
- Training configuration: `/workspace/cosmos-predict2/cosmos_predict2/configs/base/experiment/cosmos_nemo_assets.py`
- Startup command in the Task-based Modeling module:

```
set -ex

cd /workspace/cosmos-predict2
source $HOME/.local/bin/env && source .venv/bin/activate
export HF_ENDPOINT=https://hf-mirror.com

DISTRIBUTED_ARGS="
  --nproc_per_node ${NPROC_PER_NODE:-$(nvidia-smi -L | wc -l)} \
  --nnodes ${WORLD_SIZE:-1} \
  --node_rank ${RANK:-0} \
  --master_addr ${MASTER_ADDR:-127.0.0.1} \
  --master_port ${MASTER_PORT:-23456}"

torchrun $DISTRIBUTED_ARGS --rdzv_id 123 --rdzv_backend c10d --
rdzv_endpoint $MASTER_ADDR:12341 -m scripts.train --
config=cosmos_predict2/configs/base/config.py --
experiment=predict2_video2world_training_14b_cosmos_nemo_assets
```

- Storage path settings:
 - Mounted checkpoints directory (required):
 - Source path: the downloaded checkpoints directory
 - Container path: `/workspace/cosmos-predict2/checkpoints`
 - Mounted datasets directory (required):
 - Source path: the downloaded and processed datasets directory
 - Container path: `/workspace/cosmos-predict2/datasets`
 - config directory to be mounted (optional):
 - Source path: the locally modified config directory. Select the subdirectory at the `cosmos_predict2/configs/base` level to prevent mounting failures due to excessively long paths.

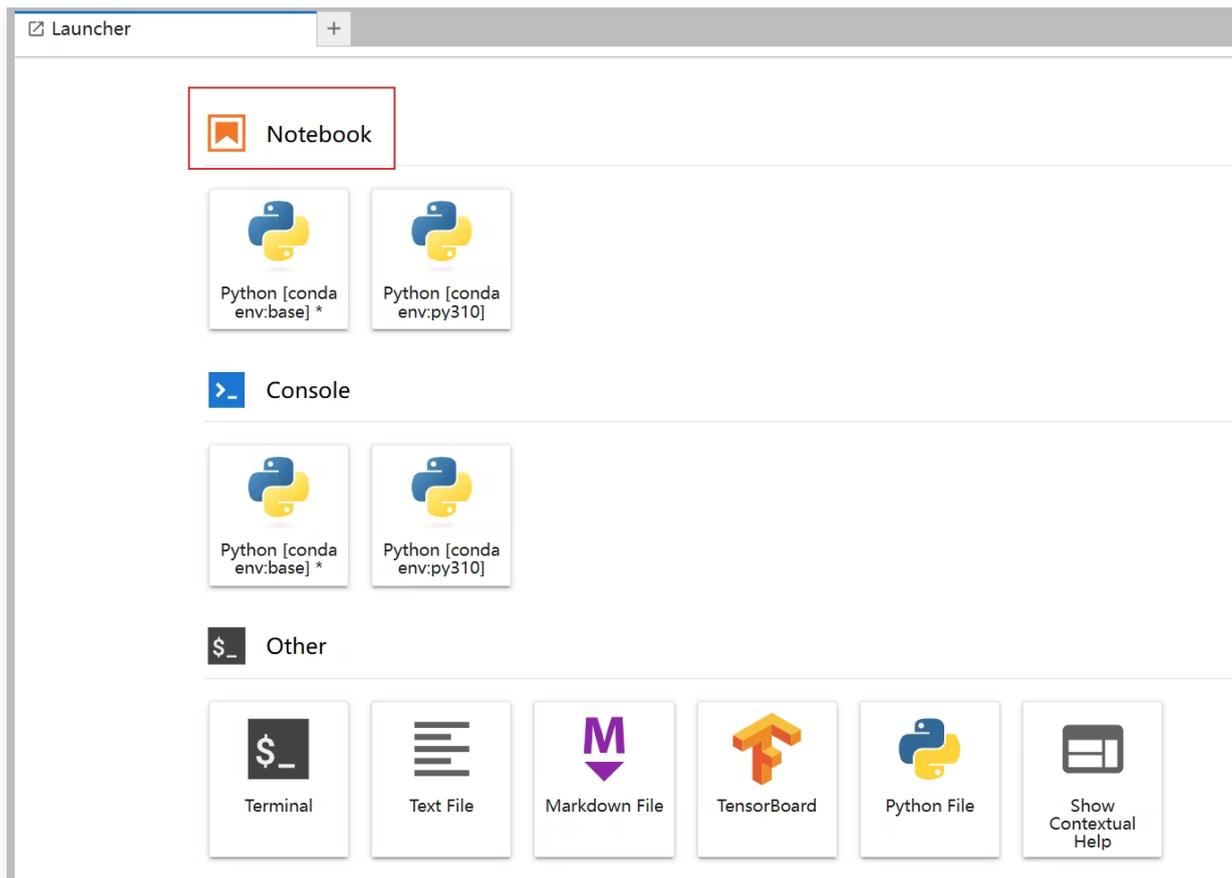
- Container path: `/workspace/cosmos-predict2/cosmos_predict2/configs/base`

If you do not modify the configuration, the default output path is `/workspace/cosmos-predict2/checkpoints/posttraining/video2world`. See the files corresponding to the above training configurations to modify parameters such as epoch and learning rate.

- Perform inference testing after training:

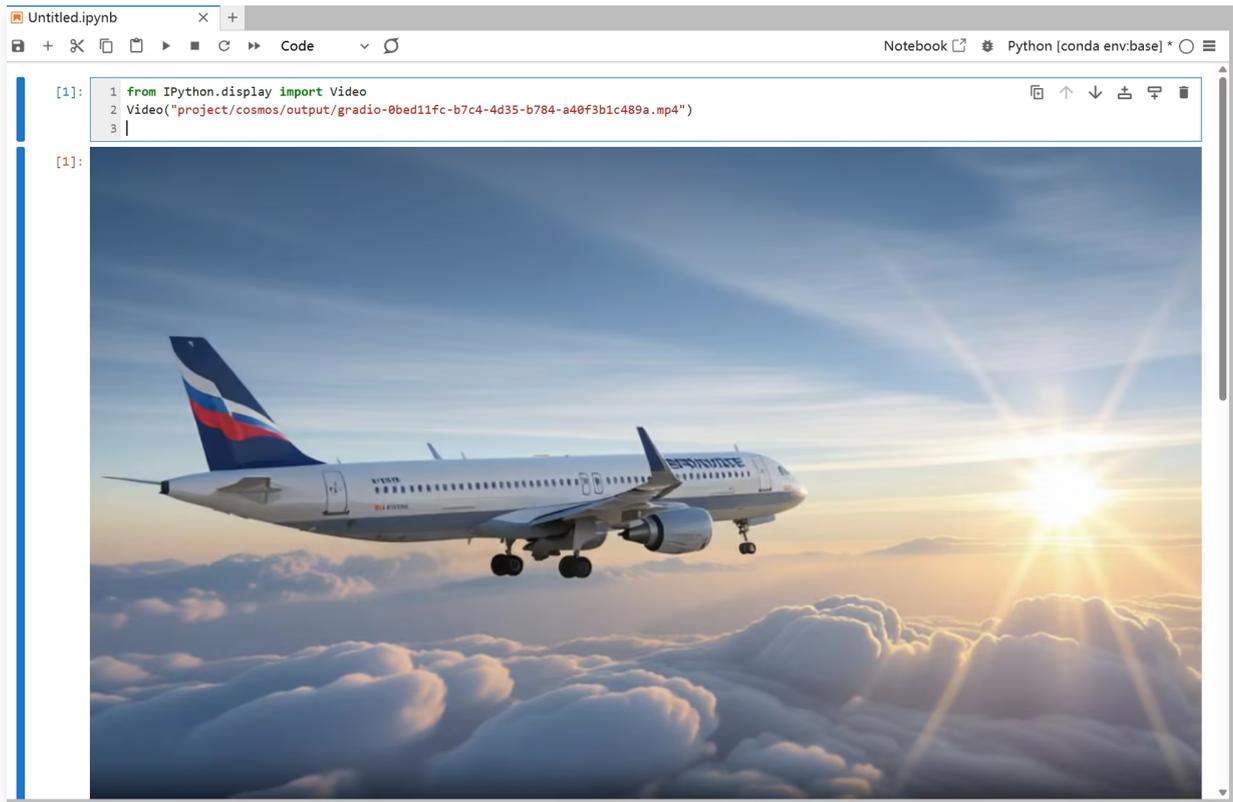
```
python examples/video2world.py \  
  --model_size 14B \  
  --dit_path \  
  "checkpoints/posttraining/video2world/14b_cosmos_nemo_assets/checkpoints/mo-  
del/iter_000001000.pt" \  
  --prompt "A video of sks teal robot." \  
  --input_path \  
  "assets/video2world_cosmos_nemo_assets/output_Digit_Lift_movie.jpg" \  
  --save_path output/cosmos_nemo_assets/generated_video_teal_robot.mp4 \  
  --offload_guardrail \  
  --offload_prompt_refiner
```

- View the generated videos.
 - Access the Dev Machines module and select Jupyter IDE.
 - Create a Notebook in Launcher.



- Input the code below and modify the content to your path to view the locally generated videos.

```
from IPython.display import Video
Video("project/cosmos/output/gradio-0bed11fc-b7c4-4d35-b784-
a40f3b1c489a.mp4")
```



Practices of Running Isaac Sim GUI on a Dev Machine

Last updated: 2026-03-12 10:35:23

Overview

This document describes how to launch the Isaac Sim software on a Tencent Cloud TI-ONE Platform (TI-ONE) dev machine and remotely connect to the dev machine via WebRTC Streaming Client for simulation visualization and interaction.

Prerequisites

1. Purchasing a GPU Rendering Instance and Managing It

Through referring to the GPU rendering instances supported for management by TI-ONE in [Billing Overview](#), purchase the corresponding instance type in the Tencent Cloud Virtual Machine (CVM) console and manage the CVM instances on TI-ONE according to [Purchase Methods](#). For example, the A10 GPU is selected.

2. Purchasing an EIP Instance

Log in to the [Tencent Cloud CVM console](#), apply for an Elastic IP (EIP) instance, and select a general BGP IP address.

For 1080P 30 FPS simulation streaming, approximately 5-20 Mbps bandwidth is required, with 50 Mbps generally recommended.

3. Preparing a CLB Instance

3.1. Purchase a CLB instance.

Log in to the [Tencent Cloud Load Balancer \(CLB\) console](#), and create a CLB instance.

- Set the network type to public network.
- Set the IP address to the public IP address applied in the first step.
- Set the network to the same VPC network as that of the managed CVM instance.

3.2 Configure a CLB security group.

Go to the CLB security group configuration page, configure the security group according to [Configuring a CLB Security Group](#), and ensure that the TCP port 49100 and UDP port 47998 are open on the path from the public network to CLB and CVM.

4. Preparing Isaac Sim Assets (Recommended)

See [Download Isaac Sim](#). It is recommended to download the Assets by following the instructions below to avoid accessing websites outside China by default during runtime to obtain resources, which may easily cause network inaccessibility and lead to failures.

Download the Assets from the official website:

- **Complete Part 1 of 3 (50 GB):** Obtain the file `isaac-sim-assets-complete-5.0.0.zip.001` after the download.
- **Complete Part 2 of 3 (50 GB):** Obtain the file `isaac-sim-assets-complete-5.0.0.zip.002` after the download.
- **Complete Part 3 of 3 (39.34 GB):** Obtain the file `isaac-sim-assets-complete-5.0.0.zip.003` after the download.

Isaac Sim Assets	5.0.0	August 2025	Robots & Sensors
			Materials & Props
			Environments
			Complete Part 1 of 3
			Complete Part 2 of 3
			Complete Part 3 of 3

Decompression method:

```
# Assume that all downloaded files are located under /tmp.

mv /tmp/isaac-sim-assets-complete-5.0.0.zip.* /workspace/IsaacLab/data/
cd /workspace/IsaacLab/data/
# These are multi-part archive files, not multiple independent ZIP files.
There are two decompression methods, and you can choose according to your
needs:
# Method 1 (recommended): Use 7z for automatic decompression.
apt-get install p7zip-full -y
# Decompress the first file, and then 7z will automatically decompress the
remaining files.
7z x isaac-sim-assets-complete-5.0.0.zip.001

# Method 2: Manually merge the files to generate a complete file of
approximately 140 GB, and then use zip to decompress it.
cat isaac-sim-assets-complete-5.0.0.zip.001 isaac-sim-assets-complete-
5.0.0.zip.002 isaac-sim-assets-complete-5.0.0.zip.003 > isaac-sim-assets-
complete-5.0.0.zip
```

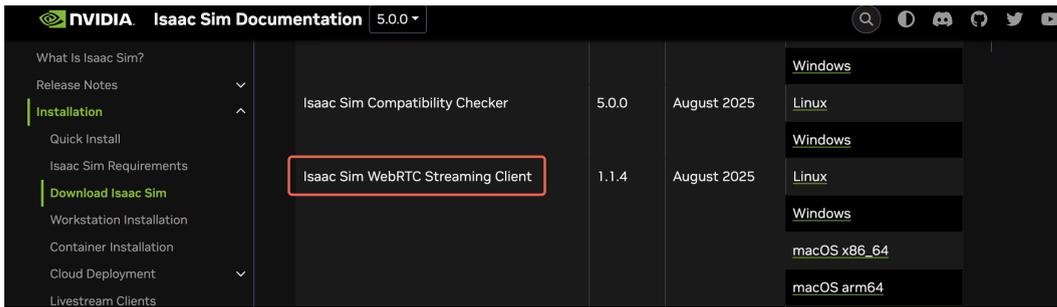
```
unzip "isaac-sim-assets-complete-5.0.0.zip"
```

Finally, the structure of the `/workspace/IsaacLab/data/` directory is as follows:

```
total 140G
drwxr-xr-x 3 root root 8.0K Aug 19 10:59 Assets-rw-r----- 1 root root 34G
Aug 18 16:28 isaac-sim-assets-complete-5.0.0.zip.001
-rw-r----- 1 root root 29G Aug 18 16:28 isaac-sim-assets-complete-
5.0.0.zip.002
-rw-r----- 1 root root 25G Aug 18 16:28 isaac-sim-assets-complete-
5.0.0.zip.003
.
├── Assets
│   └── Isaac
│       └── 5.0
│           ├── Isaac
│           │   ├── Environments
│           │   ├── IsaacLab
│           │   ├── Materials
│           │   ├── People
│           │   ├── Props
│           │   ├── Robots
│           │   ├── Samples
│           │   └── Sensors
│           └── NVIDIA
│               ├── Assets
│               └── Materials
├── isaac-sim-assets-complete-5.0.0.zip.001
├── isaac-sim-assets-complete-5.0.0.zip.002
└── isaac-sim-assets-complete-5.0.0.zip.003
```

5. Installing Isaac Sim WebRTC Streaming Client

Download Isaac Sim WebRTC Streaming Client by clicking [Download Isaac Sim](#) and install it on your local computer. Note that the Isaac Sim version is selected at the top of the page.



Operation Steps

Step 1: Creating a Dev Machine

1. Create a dev machine in the [TI-ONE console](#), and choose Built-in Embodied AI Industry Image > isaac-sim5.0-lab2.2.
 - Resource configuration: Select at least one A10 GPU card, with a minimum of 4 CPU cores and 16GB of memory.
 - Storage configuration:
 - Storage Type: CFS.
 - Source Path: Select the processed path from Preparations—4. Preparing Assets. For example, xxx/Assets.
 - Container Mount Path: custom path. For example, /workspace/IsaacLab/data/Assets.
2. Enable the custom port, select the CLB instance prepared in the previous step, and open ports 49100 (using the TCP access protocol) and 47998 (using the UDP access protocol).

Step 2: Starting the Isaac Sim Server

1. Log in to the dev machine via Secure Shell (SSH) or Notebook. Run the following commands:

```
cd /isaac-sim
sh runheadless.sh --/app/livestream/publicEndpointAddress=<CLB public IP
address> --/app/livestream/port=49100 -
-/persistent/isaac/asset_root/default="/workspace/IsaacLab/data/Assets/Isa
ac/5.0"

# If your network can obtain Assets through accessing websites outside
China,
# you can remove the --/persistent/isaac/asset_root/default= part.
```

- And replace <CLB public IP address> with the public IP address of the CLB instance prepared in the previous step.

- Replace the content after `--/persistent/isaac/asset_root/default=` with your own mount path. The path should contain the section in the red box of the final directory structure of the prepared assets.

```

7  └─ Assets
8  |   └─ Isaac
9  |      └─ 5.0
10 |          └─ Isaac
11 |              └─ Environments
12 |                  └─ IsaacLab
13 |                      └─ Materials
14 |                          └─ People
15 |                              └─ Props
16 |                                  └─ Robots
17 |                                      └─ Samples
18 |                                          └─ Sensors
19 |                                              └─ NVIDIA
20 |                                                  └─ Assets
21 |                                                      └─ Materials
22 └─ isaac-sim-assets-complete-5.0.0.zip.001
23 └─ isaac-sim-assets-complete-5.0.0.zip.002
24 └─ isaac-sim-assets-complete-5.0.0.zip.003

```

2. Wait for Isaac Sim to complete the startup. The first startup may take 3 to 5 minutes. Wait patiently.

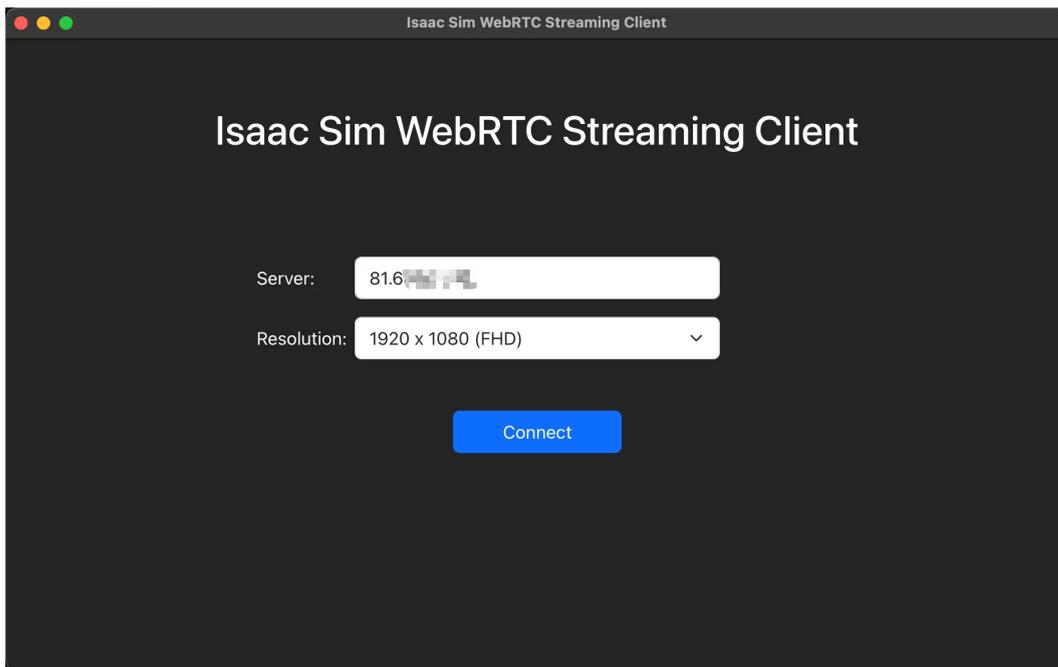
```

2025-10-13T12:48:22Z [25,809ms] [Warning] [omni.kit.imgui_renderer.plugin] _createExtendCursor: No windowing.
2025-10-13T12:48:22Z [25,809ms] [Warning] [omni.kit.imgui_renderer.plugin] _createExtendCursor: No windowing.
2025-10-13T12:49:07Z [70,666ms] [Warning] [usdr.hdr.fabric_scene_delegate.plugin] using high frequency span is disabled
2025-10-13T12:49:07Z [70,666ms] [Warning] [usdr.hdr.fabric_scene_delegate.plugin] using high frequency span is disabled
2025-10-13T12:49:07Z [70,667ms] [Warning] [usdr.hdr.fabric_scene_delegate.plugin] using high frequency span with attrs is disabled
2025-10-13T12:49:07Z [70,717ms] [Warning] [usdr.hdr.fabric_scene_delegate.plugin] using high frequency span is disabled
2025-10-13T12:49:07Z [70,717ms] [Warning] [usdr.hdr.fabric_scene_delegate.plugin] using high frequency span is disabled
[437.234s] Isaac Sim Full Streaming App is loaded.
main: thread_init: already added for thread

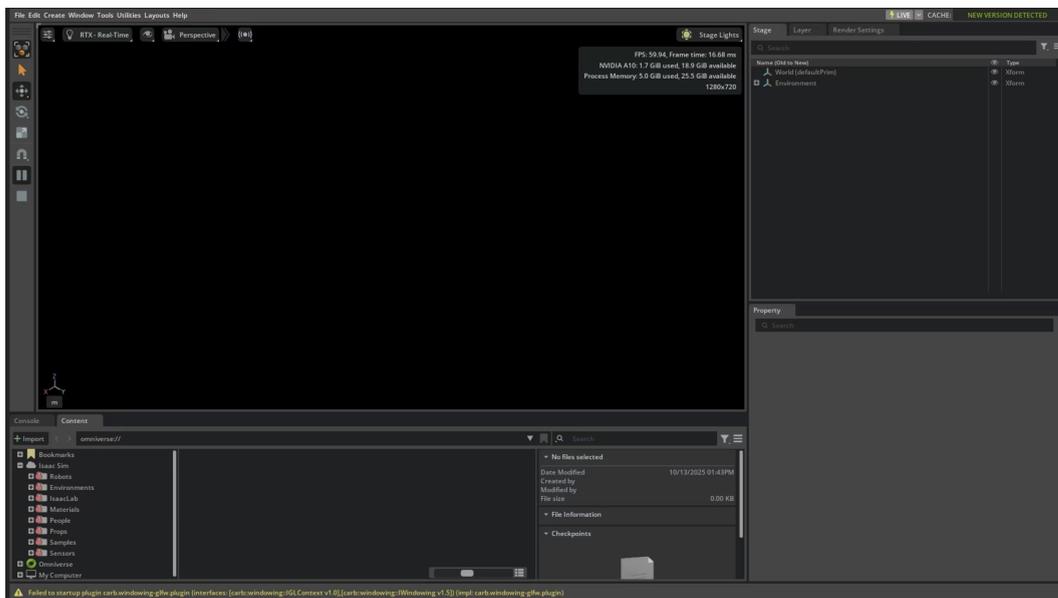
```

Step 3: Opening Isaac Sim WebRTC Streaming Client

1. Click the icon to open the client and enter the public IP address of the CLB instance prepared in the previous step.

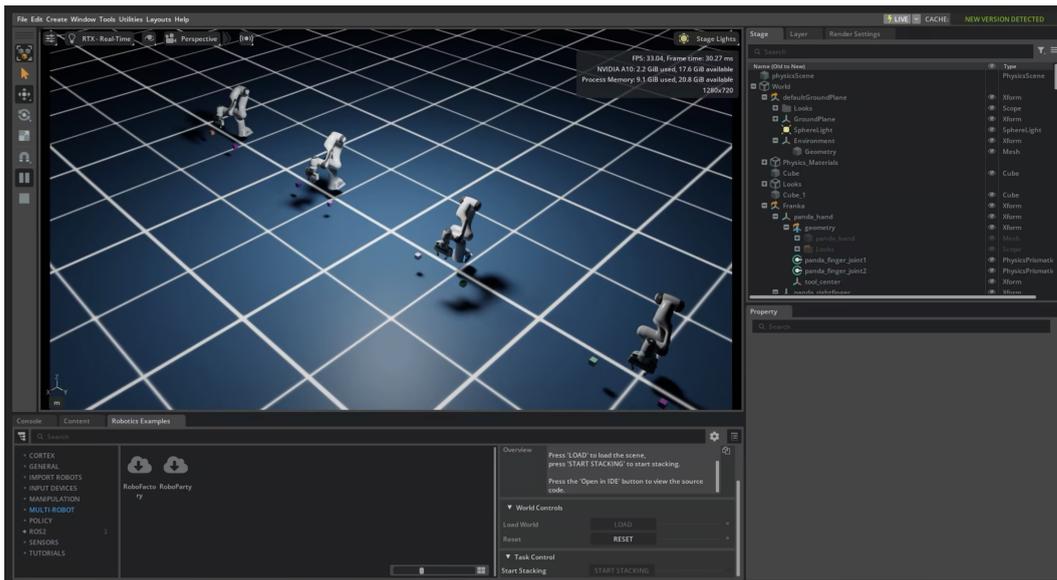


2. Click **Connect** to open the Isaac Sim GUI.

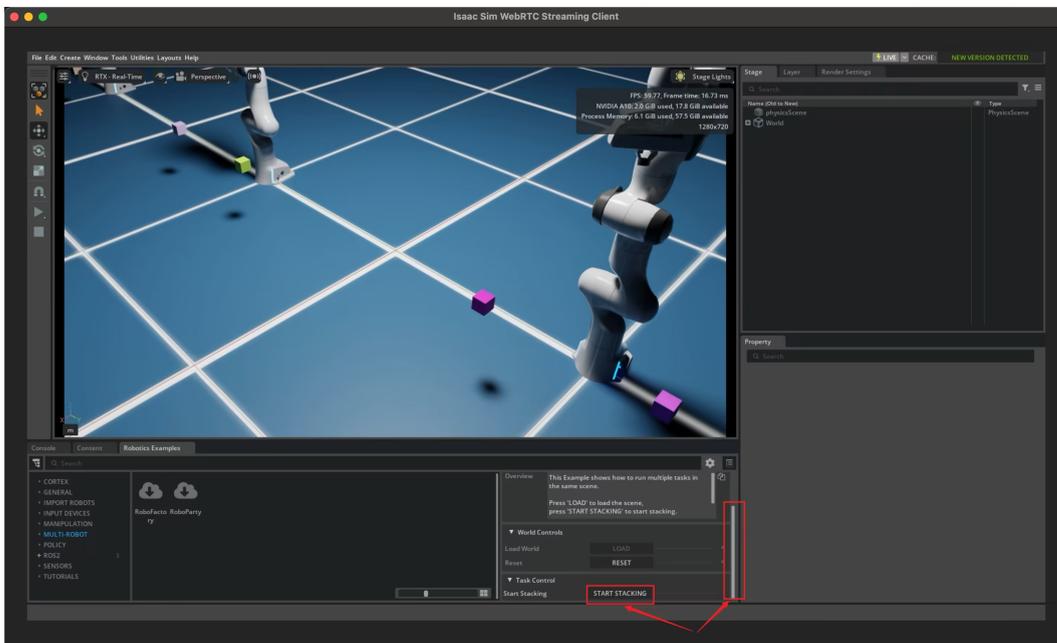


Step 4: Simulation Demonstration

1. Choose **Window > Examples > Robotics Examples**, and the Robotics Examples panel will appear in the lower left corner.
2. Choose **MULTI ROBOT > RoboFactory**.
3. Click **LOAD** on the right.



4. After loading is completed, scroll down and click **START STACKING**. Then you can see the robot start executing the box stacking task.



Reference Documentation

1. [Download Isaac Sim](#)
2. [Livestream Clients](#)
3. [Setup Tips](#)

Multimodal Model Series

Importing and Deploying the SD-WebUI Service

Last updated: 2026-03-10 10:19:30

Overview

Stable Diffusion (SD) is an open-source deep learning model capable of generating corresponding images from text prompts with low computational requirements and high efficiency. [SD-WebUI](#) is a concise and user-friendly UI encapsulated based on SD. Tencent Cloud, based on this, provides a convenient deployment mode and inference acceleration capabilities.

This document describes how to import, deploy, and call the SD-WebUI model on Tencent Cloud TI-ONE Platform (TI-ONE).

This document provides 2 cases for demonstration:

- Basic SD model inference, selecting the Cloud File Storage (CFS) mode as the model source
- SD + ControlNet model inference acceleration

Prerequisites

You need to first activate [CFS](#), and [Cloud Virtual Machine \(CVM\)](#). Subsequent models need to be stored in the file system, and uploading relies on CVM.

Case 1: Basic SD Model Inference (Selecting CFS or GooseFS Mode as the Model Source)

Step 1: Prerequisites—Purchasing CVM and CFS/GooseFS

1. Log in to the CFS or GooseFS product console, select the required storage type, and complete the corresponding configuration.

Note:

1. For the region of CFS, GooseFS, and CVM, select a region (Beijing, Shanghai, or Guangzhou) supported by TI-ONE.
2. The networks for CFS, GooseFS, and CVM must be consistent.

2. Mount CFS or GooseFS on CVM.

Step 2: Uploading the Model to CFS or GooseFS

1. Place the SD base model in the models/Stable-diffusion path of CFS, and upload the metadata related to the model to the .cache directory together.

Note:

Due to software licensing and compliance issues related to Hugging Face, you need to download the WebUI models and plugins yourself before deploying them on TI-ONE.

2. An example of the initial minimal directory structure:

```
(base) [root@VM-0-16-tencentos webui]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.8G   0  1.8G   0% /dev
tmpfs           1.8G  24K  1.8G   1% /dev/shm
tmpfs           1.8G 258M  1.6G  15% /run
tmpfs           1.8G   0  1.8G   0% /sys/fs/cgroup
/dev/vda1       50G  8.8G   39G  19% /
192.168.0.14:/  80G  28G   53G  35% /data
(base) [root@VM-0-16-tencentos webui]# pwd
/data/webui
(base) [root@VM-0-16-tencentos webui]# ls -al
total 12
drwxr-xr-x  4 root root   50 Nov 24 14:28 .
drwxr-xr-x 14 1000 users 4096 Nov 24 14:28 ..
drwxr--r--  5 root root   53 Nov 24 14:28 .cache
-rw-r--r--  1 root root  420 Nov 24 14:28 config.json
drwxr-xr-x 14 root root 4096 Nov 24 14:29 models
```

3. models directory structure

```
(base) [root@VM-0-16-tencentos models]# pwd
/data/webui/models
(base) [root@VM-0-16-tencentos models]# tree -L 2
.
├── Stable-diffusion
│   └── pastelboys2D_v30.safetensors
└── 1 directory, 1 file
(base) [root@VM-0-16-tencentos models]#
```

4. .cache directory structure

```
(base) [root@VM-0-16-tencentos .cache]# pwd
/data/webui/.cache
(base) [root@VM-0-16-tencentos .cache]# tree -L 3
.
├── gdown
├── huggingface
│   ├── hub
│   │   ├── models--openai--clip-vit-large-patch14
│   │   └── version.txt
├── matplotlib
│   └── fontlist-v330.json

```

5 directories, 2 files

```
(base) [root@VM-0-16-tencentos .cache]#
```

Step 3: Deploying the Service

1. Log in to the [TI-ONE console](#), in the left sidebar, choose **Model Services > Online Services**, click **New Service**, and go to the creation page.
2. You can enter the service name, select **CFS or GooseFS** as the model source, choose the CFS or GooseFS instance in the current region, and set the mount path to **the custom model path** defined in Step 2 (such as webui). For the running environment, select **Built-in/STABLEDIFFUSION-WebUI(2.0.0)/sd-webui-1.6.0(gpu)**. Select CVM instance specifications **with GPU**, configure other required information, and click **Start Service**. When the service status becomes Running, the deployment is complete, and inference can be performed.

Step 4: Calling the Service

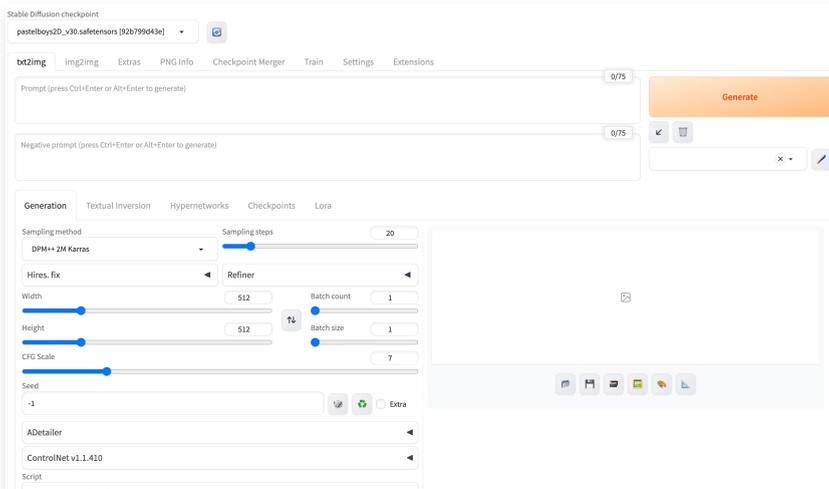
1. Click the name of the running Web-UI model service to go to the service details page, click the **Service Call** tab, and enter the Web-UI URL to the right of the API Call Address field in the API Information section, for example, text-to-image: /sdapi/v1/txt2img. For details, see [API Reference](#).

ⓘ Note:

To reduce access latency, see [Calling over Private Networks](#) and [Private Link](#) to obtain the private network address between your VPC network and the service, and access the service via the private network address.

Step 5: Using the SD-WebUI Page

1. In the Service Management list, an Open WebUI Page button will appear for running services. You can click it to open the WebUI page in a new window.



Step 6: Appendix Use Instructions

1. **Description of the complete directory structure on CFS or GooseFS after service deployment**
 - 1.1 extensions/sd-tiacc and extensions/sd-webui-controlnet are the platform's built-in accelerated sd-webui-controlnet plugin. If you have not stored these plugins on CFS or GooseFS, the platform's built-in plugin will be used.
 - 1.2 The directory structure and configuration files on CFS or GooseFS are consistent with the native WebUI.

```
(base) [root@VM-0-16-tencentos webui]# pwd
/data/webui
(base) [root@VM-0-16-tencentos webui]# tree -L 2
.
├── config.json
├── config_states
├── embeddings
├── extensions
│   ├── sd-tiacc
│   └── sd-webui-controlnet
├── models
│   ├── Codeformer
│   ├── ControlNet
│   ├── ESRGAN
│   ├── GFPGAN
│   ├── hypernetworks
│   ├── karlo
│   ├── LDSR
│   ├── Lora
│   ├── Stable-diffusion
│   ├── SwinIR
│   └── VAE-approx
├── scripts
├── styles.csv
└── ui-config.json

18 directories, 3 files
(base) [root@VM-0-16-tencentos webui]#
```

2. Environment variable description

2.1 The environment variable **CLI_ARGS** represents the WebUI startup parameter, which is "--allow-code --enable-insecure-extension-access --api" by default.

2.2 When environment variables such as **COS_UPLOAD_ENBALE**, **COS_REGION**, **COS_BUCKET**, and **COS_FILE_PREFIX** are configured simultaneously, images generated by calling the WebUI's text-to-image or image-to-image APIs will be automatically saved to Cloud Object Storage (COS). The API returns a temporary link (valid for 10 minutes by default) to the images on COS. **COS_SECRET_ID** and **COS_SECRET_KEY** are permanent keys and are optional. If not configured, the platform uses temporary keys.

Case 2: SD + ControlNet Model Inference Acceleration

Step 1: Prerequisites—Purchasing CVM and CFS/GooseFS

1. Log in to the CFS or GooseFS product console, select the required storage type, and complete the corresponding configuration.

Note:

1. For the region of CFS, GooseFS, and CVM, select a region (Shanghai, Guangzhou, or Nanjing) supported by TI-ONE.
2. The networks for CFS, GooseFS, and CVM must be consistent.

2. Mount CFS or GooseFS on CVM.

Step 2: Description of Acceleration Parameters in config.json

1. Below is an example of the config.json content:

```
{
  "use_tiacc": false,
  "use_tiacc_unet": false,
  "control_net_max_models_num": 3,
  "control_net_model_cache_size": 3,
}
```

2. Description of acceleration parameters in config.json:

2.1 When `use_tiacc` is set to `false`, it means the Angel tune acceleration method is disabled, and the non-tune acceleration method is adopted. (For large sizes greater than or equal to 1024, the performance of Angel tune and non-tune acceleration is comparable.)

2.2 `use_tiacc_unet` currently needs to be set to `false` and is not available (currently, only controlnet tune is supported).

2.3 `control_net_model_cache_size` represents the number of ControlNet models resident in GPU memory for acceleration. If you frequently use multiple ControlNet models, you need to increase this parameter; otherwise, conversion overhead will occur. (You can perform load testing on this parameter to determine the maximum value that the GPU memory supports.)

Step 3: Uploading ControlNet Models to CFS or GooseFS

1. Upload the ControlNet models you need to the `models/ControlNet` directory.
2. When `use_tiacc` is set to `true`, it is necessary to upload the cache models accelerated by TI-ONE to the `/models/cache` directory.

Note:

You can download cache models according to your needs.

File Description	ControlNet Version	Download URL	Required

Original model file	controlnet1.0	Click to download.	Required for all
Original model file	controlnet1.1	Click to download.	Required for all
A10-accelerated model file	controlnet1.0	Click to download.	Required for A10 CVM instances
A10-accelerated model file	controlnet1.1	Click to download.	Required for A10 CVM instances
A100-accelerated model file	controlnet1.0	Click to download.	Required for A100 CVM instances
A100-accelerated model file	controlnet1.1	Click to download.	Required for A100 CVM instances

Step 4: Deploying the Service

1. Log in to the [TI-ONE console](#), in the left sidebar, choose **Model Services > Online Services**, click **New Service**, and go to the creation page.
2. You can enter the service name, select **CFS or GooseFS** as the model source, choose the CFS or GooseFS instance in the current region, and set the mount path to **the custom model path** defined in Step 2 (such as webui). For the running environment, select **Built-in/STABLEDIFFUSION-WebUI(2.0.0)/sd-webui-1.6.0(gpu)**. Select CVM instance specifications **with GPU**, configure other required information, and click **Start Service**. When the service status becomes Running, the deployment is complete, and inference can be performed.

Step 5: Calling the Service

1. Click the name of the running Web-UI model service to go to the service details page, click the **Service Call** tab, and enter the Web-UI URL to the right of the API Call Address field in the API Information section, for example, text-to-image: /sdapi/v1/txt2img. For details, see [API Reference](#).

Note:

To reduce access latency, see [Private Link](#) to obtain the private network address between your VPC network and the service, and access the service via the private network address.

Importing and Deploying the SD ComfyUI Service

Last updated: 2026-03-10 10:19:49

Overview

ComfyUI is a node-based graphical user interface (GUI) specifically designed for Stable Diffusion (SD). It enables users to build complex image generation workflows by linking different blocks (nodes). Compared to SD-WebUI, ComfyUI provides exceptional freedom and flexibility, supporting high levels of customization and workflow reuse.

This document describes how to build a custom ComfyUI image based on the platform base image, import SD models, and deploy the ComfyUI service for online inference.

Note:

- The default model root directory in the container on the platform is the fixed path `/data/model`.
- The default port for the inference service on the platform is 8501.

Prerequisites

You need to first activate [Cloud File Storage \(CFS\)](#) , and [Cloud Virtual Machine \(CVM\)](#) . Subsequent models need to be stored in the file system, and uploading relies on CVM.

1. Building a Custom Image

This section introduces how to build a custom image based on the platform base image, upload it to a custom image repository, and use it for service deployment.

If you want to try it out quickly, you can directly upload the platform base image to [Tencent Cloud > Image Repository](#) . See the [Service Deployment](#) section for deployment and experience.

Platform Base Images

1. Built-in features are as follows:

- Built-in image-to-video plugin: ComfyUI-VideoHelperSuite
- Built-in SD model: v1-5-pruned-emaonly.ckpt

2. Obtaining the image:

- Log in to the [Tencent Cloud > Image Repository](#) console and find [Platform Images > Public Images](#) .
- Base image:

```
cc.ccs.tencentyun.com/tione-public-images/ti-cloud-stable-diffusion-  
webui:comfyui-v0.0.1-062483
```

○ Pull to local:

```
docker pull ccr.ccs.tencentyun.com/tione-public-images/ti-cloud-  
stable-diffusion-webui:comfyui-v0.0.1-062483
```

ⓘ Note:

1. For the Comfyui community edition, click [here](#).
2. The commit hash used in the examples of this document:
062483823738ed610d8d074ba63910c90e9d45b7.

Building a Custom Image

1. Take the custom image `ccr.ccs.tencentyun.com/test/stable-diffusion-webui:comfyui-1.0.1` as an example:

○ Custom image Dockerfile.

```
FROM ccr.ccs.tencentyun.com/tione-public-images/ti-cloud-stable-  
diffusion-webui:comfyui-v0.0.1-062483  
  
ENV DEBIAN_FRONTEND=noninteractive PIP_PREFER_BINARY=1  
ENV ROOT=/stable-diffusion  
  
##### ----- [Replace as needed: Begin] -----  
# already exit ComfyUI codes in base image  
RUN rm -rf ${ROOT}  
  
RUN --mount=type=cache,target=/root/.cache/pip \  
  git clone https://github.com/comfyanonymous/ComfyUI.git ${ROOT} && \  
  \  
  cd ${ROOT} && \  
  git checkout master && \  
  git reset --hard 062483823738ed610d8d074ba63910c90e9d45b7 && \  
  pip install -r requirements.txt  
  
##### ----- [Replace as needed: End] -----
```

```
WORKDIR ${ROOT}

RUN chmod u+x /docker/entrypoint.sh && cp
/docker/extra_model_paths.yaml ${ROOT}

ENV NVIDIA_VISIBLE_DEVICES=all PYTHONPATH="${PYTHONPATH}:${PWD}"
CLI_ARGS=""
EXPOSE 7860
ENTRYPOINT ["/docker/entrypoint.sh"]
CMD python -u main.py --listen --port 8501 ${CLI_ARGS}
```

- Build a custom image.

You need to replace `ccr.ccs.tencentyun.com/test/stable-diffusion-webui:comfyui-1.0.1` with your own image repository name.

```
docker build -f Dockerfile -t ccr.ccs.tencentyun.com/test/stable-
diffusion-webui:comfyui-1.0.1 .
```

- Push the custom image to the image repository.

2. Model Import

Step 1: Prerequisites—Purchasing CVM and CFS/GooseFSx

1. Log in to the [CFS console](#) or [GooseFSx console](#), select **File System** in the left sidebar, click **Create**, select the storage type you need, and complete the corresponding configuration. (This step can be skipped for GooseFSx.)
2. Log in to the [CFS console](#) or [GooseFSx console](#), select **Instance** in the left sidebar, click **New**, and proceed with the purchase.

Note:

1. For the region of CFS, GooseFS, and CVM, select a region (Beijing, Shanghai, or Guangzhou) supported by TI-ONE.
2. The networks for CFS, GooseFS, and CVM must be consistent.

3. Mount CFS on CVM. For details, see [Using CFS Instance on Windows Clients](#).
4. Mount GooseFSx on CVM. For details, see [Managing GooseFSx Clients](#).

Example:

```
sudo mount -t nfs -o vers=4.0,noresvport 192.168.0.14:/ /data
```

Step 2: Uploading the Model to CFS or GooseFSx

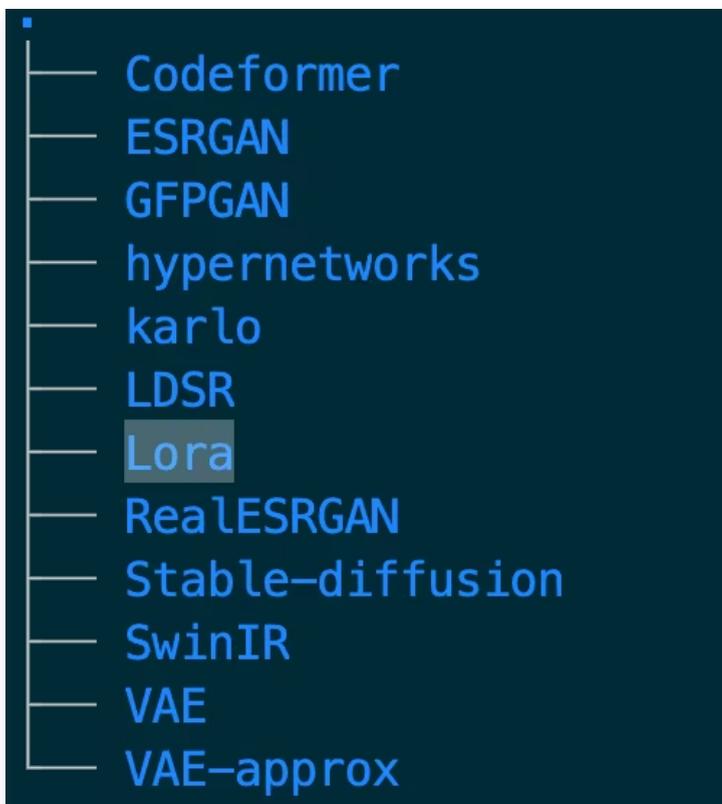
Assuming `#{CFS_DIR}` is the mount directory for CFS or GooseFSx, for example, `#{DATA_DIR} = /data/comfyui`:

1. Base models are stored in `#{DATA_DIR}/models/Stable-diffusion`. When there are no models in this path, the platform will use the built-in model `v1-5-pruned-emaonly.ckpt`.
2. Plugins are stored in `#{DATA_DIR}/config/model/config/comfy/custom_nodes`, and the platform image has a built-in plugin `ComfyUI-VideoHelperSuite`.
3. Model metadata is stored in `#{DATA_DIR}/.cache`, and the platform image has built-in `hugging face models--openai--clip-vit-large-patch14`.

Note:

Due to software licensing and compliance issues related to Hugging Face, public networks such as Hugging Face cannot be accessed on the platform. Thus, the metadata related to models needs to be manually stored here.

4. The directory structures for LoRa and others can be stored under `#{DATA_DIR}/models/` in the same way as `SD-WebUI`.



3. Service Deployment

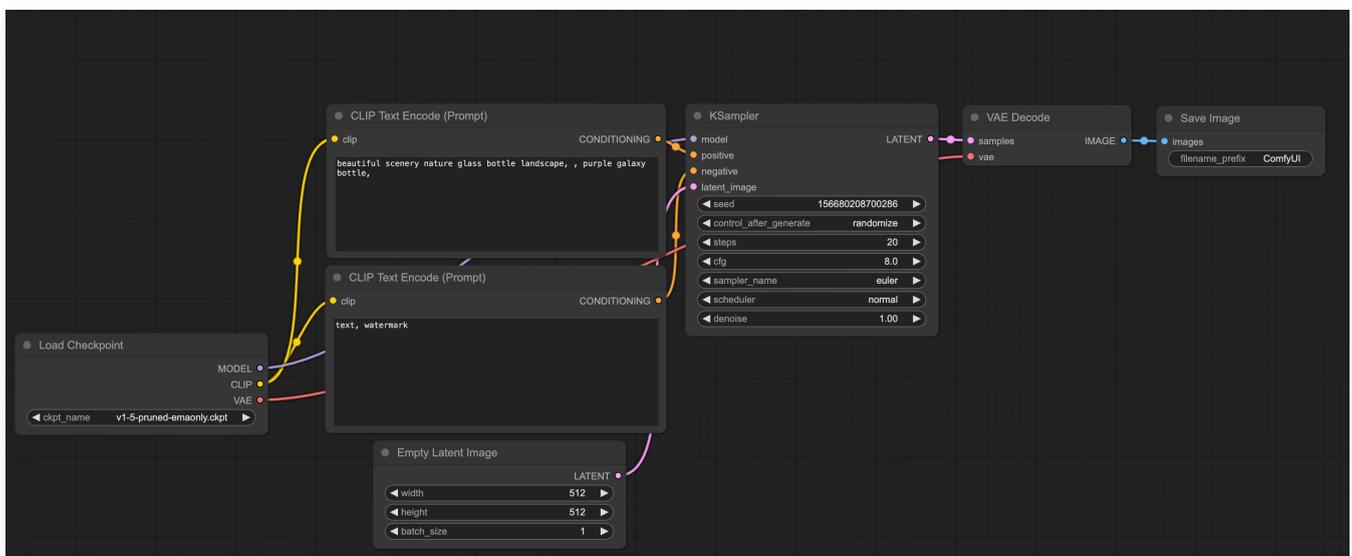
1. Log in to [Tencent Cloud TI-ONE Platform \(TI-ONE\) console](#), in the left sidebar, choose **Model Services > Online Services**, click **New Service**, and go to the creation page.
2. You can enter the service name, select **CFS or GooseFS** as the model source, choose the CFS or GooseFS instance in the current region, and set the mount path to **the custom model path** (such as `comfyui`). For the running environment, select **Custom**. Select the computing power, configure any other necessary information, and click **Start Service**. When the service status becomes **Running**, the deployment is complete, and inference can be performed.

4. WebUI Page Usage

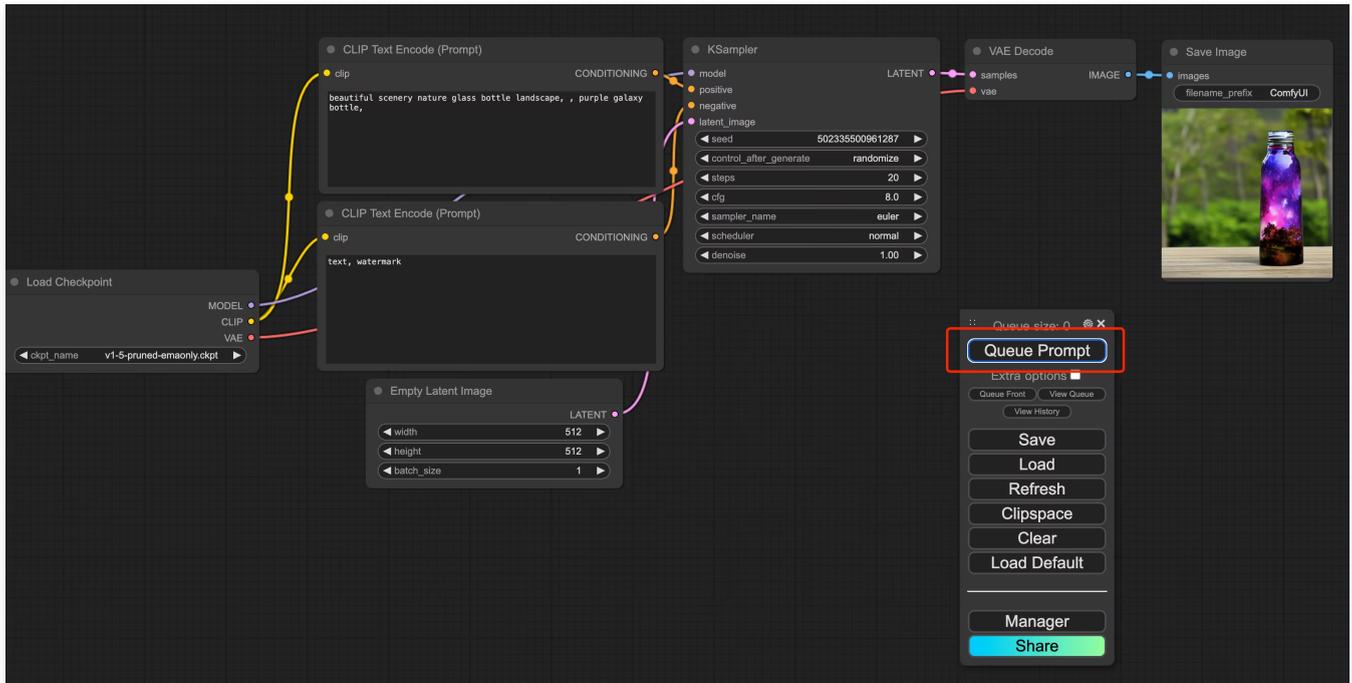
After the service is deployed, an **Open WebUI Page** button will appear in the Operation column under the **Service Management** tab on the service list page.

Case 1: Text-to-Image

1. Below is a ComfyUI example:

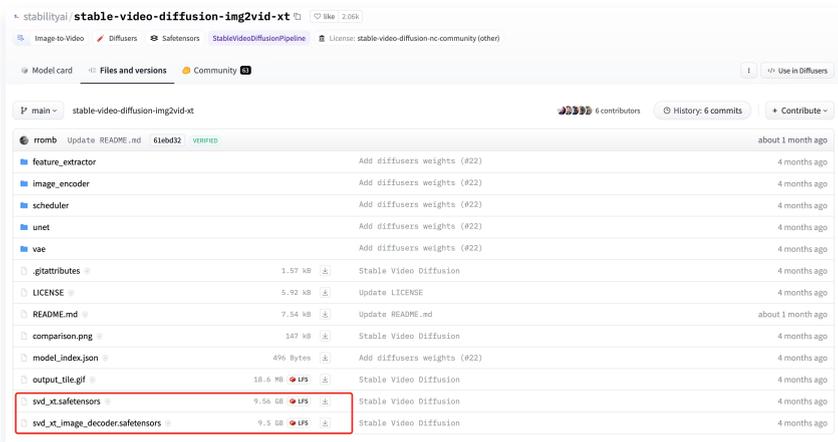


2. Click Queue Prompt to generate images, as shown in the following figure:



Case 2: Image-to-Video

1. Upload the Stable Video Diffusion (SVD) model file to the CFS path: `${CFS_DIR}/models/Stable-diffusion.`



Note:

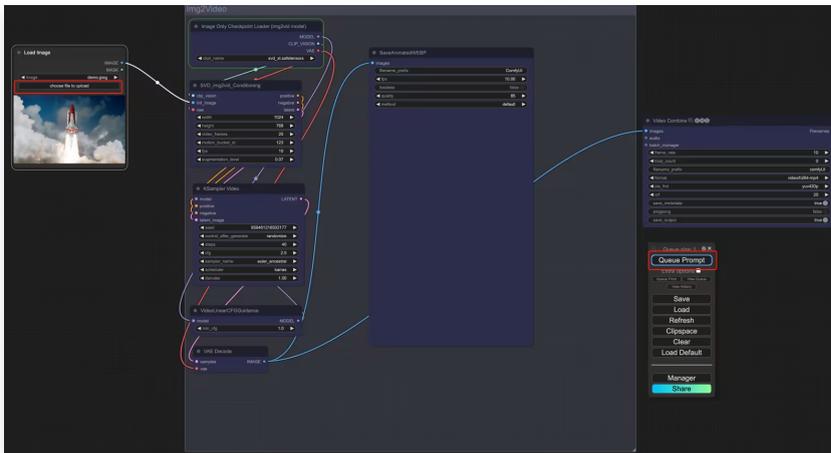
You can [click here to download](#) the SVD model file.

2. As described in [Step 2 of Model Import](#), place the SVD model file under the /models/Stable-

```
models/Stable-diffusion
├── sd-v1-5-inpainting.ckpt
├── svd_xt_image_decoder.safetensors
├── svd_xt.safetensors
└── v1-5-pruned-emaonly.ckpt
```

diffusion path.

3. Open the interface and upload the workflow. Below is an example of an image-to-video

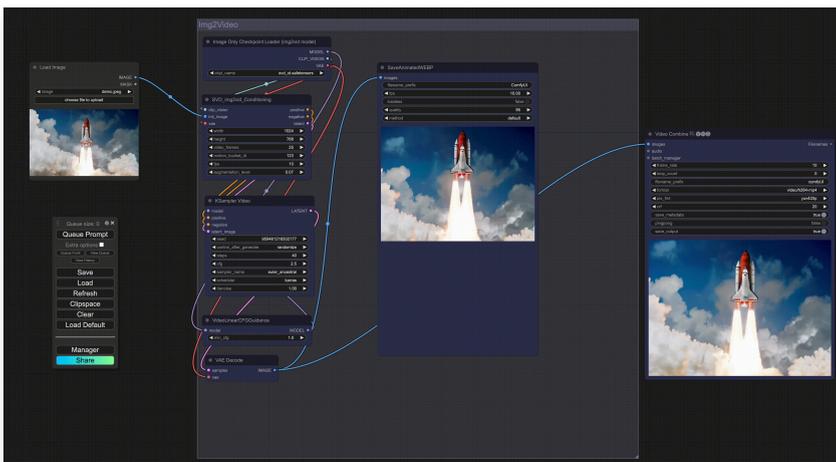


workflow:

Note:

You can [click here to download](#) the workflow.json file.

4. Click **Queue Prompt** to perform inference and obtain the video, as shown below:



Built-in Training Image List

Last updated: 2026-02-26 15:39:30

Introduction

TI-ONE features built-in mainstream deep learning and machine learning frameworks such as TensorFlow, PyTorch, and PySpark, while also introducing its self-developed training acceleration framework Angel. Among these, tilearn-llm is a training acceleration component customized for LLMs, which has been integrated into the platform's built-in images and supports both dev machines and task-based modeling. Additionally, TI-ONE includes built-in images for the autonomous driving industry and embodied intelligence industry, enabling relevant industry users to rapidly build required models using these pre-configured images.

Built-in Generic Images

Frame work	Image Name	Supported Training Mode	Remarks
PyTorch	verl0.6.1-vllm0.11.0-torch2.8-py312-cuda12.8-gpu	DDP,MPI,Ray	<ul style="list-style-type: none"> • Supported core libraries: Python 3.12.11, CUDA 12.8.93, jupyterlab 4.5.0, torch 2.8.0+cu128, transformers 4.57.1, verl 0.6.1, vllm 0.11.0, and ray 2.49.2 • supported GPU models: PNV6, HCCPNV6, H100, H800, L20, L40, A100, A800, A10, V100, T4 • Supported modules: Task-based Modeling and Dev Machines.
	tilearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu	DDP,MPI,Ray	<ul style="list-style-type: none"> • Supported core libraries: Python 3.10, CUDA 12.4, jupyterlab 2.3.2, torch 2.3.0a0+40ec155e58.nv24.3, transformers 4.39.3, deepspeed 0.13.4, tilearn-llm 0.9.9, tilearn.ops 0.2.2.175, angel-vllm 0.4.2, and ray 2.42.0. • Supported GPU models: PNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4.

		<ul style="list-style-type: none"> • Supported modules: Task-based Modeling and Dev Machines.
tilearn-llm0.9-torch2.1-py3.10-cuda12.1-gpu	DDP	<ul style="list-style-type: none"> • Supported core libraries: Python 3.10, CUDA 12.1, jupyterlab 2.3.2, torch 2.1.2, transformers 4.39.3, deepspeed 0.14.0, tilearn-llm 0.9.3.3, tilearn.ops 0.2.1.172, and angel-vllm 0.3.4. • Supported GPU models: H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
tilearn-llm0.8-torch2.1-py3.10-cuda12.1-gpu	DDP	<ul style="list-style-type: none"> • Supported core libraries: Python 3.10, CUDA 12.1, jupyterlab 2.3.2, torch 2.1.0a0+b5021ba, transformers 4.31.0, deepspeed 0.10.0, tilearn-llm 0.8.3, and tilearn.ops 0.2.0.1. • Supported GPU models: H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
ti-acc2.5-torch1.9-py3.8-cuda11.1-gpu	DDP	<ul style="list-style-type: none"> • Supported core libraries: Python 3.8, CUDA 11.1, torch 1.9.0+cu111, and tiacc-training.torch 2.5.1.dev13. • Supported GPU models: A100, A10, V100, and T4. • Supported module: Task-based Modeling.
torch1.9-py3.8-cuda11.1-gpu	DDP, MPI, and Horovod	<ul style="list-style-type: none"> • Supported core libraries: Python 3.8, CUDA 11.1, and torch 1.9.0+cu111.

			<ul style="list-style-type: none"> • Supported GPU models: A100, A10, V100, and T4. • Supported module: Task-based Modeling.
	ti-acc2.5-torch1.12-tf1.15-tf2.4-pyspark2.4.5-py3.8-cuda11.3-gpu	-	<ul style="list-style-type: none"> • Supported core libraries: jupyterlab 3.6.1 and multiple conda environments, including: <ul style="list-style-type: none"> ○ pyspark: Python 3.7 and spark-2.4.5-bin-hadoop2.7 ○ pyspark3: Python 3.8 and spark-3.3.1-bin-hadoop3 ○ pytorch_py3: Python 3.8, CUDA 11.1, and torch 1.9.0+cu111 ○ tiacc_pytorch_py3: Python 3.8, CUDA 11.1, torch 1.12.1+cu113, and tiacc-training.torch 2.5.1.dev10 ○ tf_py3: Python 3.7, CUDA 10.0, and tensorflow-gpu 1.15.0 – tiacc_tf_py3: Python 3.7, CUDA 10.0, and tensorflow-gpu 1.15.0 ○ tf2_py3: Python 3.8, CUDA 11.0, and tensorflow 2.4.0 • Supported GPU models: V100, T4, and A100/A10 (supported by certain kernels). • Supported module: Dev Machines.
Tensor Flow	ti-acc1.0-tf1.15-py3.6-cuda10.0-gpu	PS-Worker	-
	tf1.15-py3.7-cpu	PS-Worker, MPI, and Horovod	-
	tf1.15-py3.7-cuda10.0-gpu		
	tf2.4-py3.8-cpu		
	tf2.4-py3.8-cuda11.1-gpu		

Spark	spark2.4.5-cpu	Spark	-
PySpark	spark2.4.5-py3.6-cpu	Spark	-
Others	py3.8-cpu	-	<ul style="list-style-type: none"> Includes Miniforge3, jupyterlab 4.3, tensorboard 2.18.0, and Python 3.8 conda environment. Supported module: Dev Machines.
	py3.10-cpu	-	<ul style="list-style-type: none"> Includes Miniforge3, jupyterlab 4.3, tensorboard 2.18.0, and Python 3.10 conda environments. Supported module: Dev Machines.

Built-in Images for the Autonomous Driving Industry

Image Name	Supported Training Mode	Remarks
ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-bevformer	DDP,MPI	<ul style="list-style-type: none"> Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 conda environment: py38: python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, mmdet3d 0.17.1, and tilearn-llm 1.0.3.1. Applicable models: BEVFormer. Go to the /examples directory of images to obtain sample code. Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. Supported modules: Task-based Modeling and Dev Machines.
ubuntu22.04-miniforge3-cu124-py38-torch2.4-	DDP,MPI	<ul style="list-style-type: none"> Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0

<p>mmcv1.7.2-vrdma-bevformer</p>		<ul style="list-style-type: none"> • conda environment: py38: python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, mmdet3d 0.17.1, and tilearn-llm 1.0.3.1. • Applicable models: BEVFormer. Go to the /examples directory of images to obtain sample code. • Supported GPU models:HCCPNV6e. • Supported module: Task-based Modeling (only available for vrdma instance types).
<p>ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-bevfusion-fastbev-maptr</p>	<p>DDP,MPI</p>	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38: python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, tilearn-llm 1.0.3.1, torchpack 0.3.1, flash-attn 0.2.0, mpi4py 4.0.3, and av2 0.1. • Applicable models: FastBEV, BEVFusion, and MapTR. Go to the /examples directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
<p>ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-panoocc</p>	<p>DDP,MPI</p>	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38: python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, tilearn-llm 1.0.3.1, mmdet3d (0.17.1, modified version of PanoOcc), timm 1.0.15, einops 0.8.1, simplejson 3.20.1, ipdb 0.13.13, einops 0.8.1, torch-scatter+cu124, cumm (0.7.13 open-sourced on github + compiled and installed with cu124), spconv (2.3.8 open-sourced on github + compiled and installed with cu124), and ops (1.0, modified version of PanoOcc).

		<ul style="list-style-type: none"> • Applicable models: PanoOcc. Go to the /examples directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-sparse4d	DDP,MPI	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38 (general): python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, tilearn-llm 1.0.3.1, mmdet3d (0.17.2 open-sourced on github), numba 0.48.0, numpy 1.19.5, nuscenesc-devkit 1.1.9, motmetrics 1.1.3, tensorboard 2.6.0, torchmetrics 0.5.0, protobuf 3.20.*, and ops (0.0.0, modified version of Sparse4D). • Applicable models: Sparse4D. Go to the /examples directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-vrdma-sparse4d	DDP,MPI	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38 (general): python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, tilearn-llm 1.0.3.1, mmdet3d (0.17.2 open-sourced on github), numba 0.48.0, numpy 1.19.5, nuscenesc-devkit 1.1.9, motmetrics 1.1.3, tensorboard 2.6.0, torchmetrics 0.5.0, protobuf 3.20.*, and ops (0.0.0, modified version of Sparse4D).

		<ul style="list-style-type: none"> • Applicable models: Sparse4D. Go to the /examples directory of images to obtain sample code. • Supported GPU models:HCCPNV6e. • Supported module: Task-based Modeling (only available for vrdma instance types).
ubuntu22.04-miniforge3-cu124-py38-torch2.4-mmcv1.7.2-uniad	DDP,MPI	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38 (general): python3.8, torch 2.4.1+cu124, mmcv-full 1.7.2, mmdet 2.26.0, mmsegmentation 0.30.0, mmengine 0.10.3, tilearn-llm 1.0.3.1, mmdet3d (0.17.1 open-sourced on github), scipy 1.7.3, scikit-image 0.19.3, google-cloud-bigquery 3.30.0, motmetrics 1.1.3, einops 0.4.1, numpy 1.19.5, casadi 3.5.5, and pytorch-lightning 1.2.5. • Applicable models: UniAD. Go to the /examples directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.
ubuntu22.04-miniforge3-cu124-py38-torch2.4-hivt	DDP,MPI	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.4, miniforge3, jupyterlab 4.3.6, jupyterlab-tensorboard-pro 4.0.0 • conda environment: py38: python3.8, torch 2.4.1+cu124, pytorch-lightning 1.9.5, torch-geometric 2.6.1, argoverse 1.1.0, and tilearn-llm 1.0.3.1. • Applicable models: HiVT. Go to the /examples directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, H800, PNV5b, L40, A100, A800, A10, V100, and T4. • Supported modules: Task-based Modeling and Dev Machines.

Built-in Embodied Intelligence Industry Image

		<ul style="list-style-type: none"> • conda environment: cosmos:python3.12,torch 2.7.1+cu128,torchvision 0.22.1+cu128,apex 0.1,transformer_engine 1.13.0+cu128,flash_attn 2.6.3+cu128 • Applicable models: cosmos-transfer1. Go to the /workspace/cosmos-transfer1 directory of images to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, and H800 • Supported modules: Task-based Modeling and Dev Machines.
isaac-gr00t-n1.5	DDP	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.8, miniforge3, ffmpeg 7.0.1 • conda environment: gr00t:python3.11.14,torch 2.5.1,torchvision 0.20.1,transformer 4.51.3,flash_attn 2.7.1.post4,torchcodec 0.1.0 • Applicable models: GR00T-N1.5-3B. Go to the /workspace/Isaac-GR00T directory of the image to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, and H800 • Supported modules: Task-based Modeling and Dev Machines.
cosmos-predict2-cu128	DDP	<ul style="list-style-type: none"> • Supported core libraries: CUDA 12.8, uv, jupyterlab 4.4.9 • uv environment: cosmos-predict2:python3.12,torch 2.6.0+cu126,torchvision 0.21.0+cu126,diffusers==0.34.0,cosmos-guardrail==0.1.0,apex==0.1.0,flash-attn==2.6.3 • Applicable models: cosmos-predict2. Go to the /workspace/cosmos-predict2 directory of the image to obtain sample code. • Supported GPU models: PNV6, HCCPNV6, H100, and H800 • Supported modules: Task-based Modeling and Dev Machines.

Custom Training Image Specifications

Last updated: 2026-03-12 10:40:21

If the platform's built-in images do not meet your requirements, you can also use custom images to create training tasks and dev machine instances. Here is an example of a Dockerfile for custom images:

Basic Image Specifications

If you want a custom image to be able to start training tasks in **Task-based Modeling**, the image must have the `openssh-server` component installed. If you need to use Git storage, the image must also have the `git` component installed. Here is an example:

```
# Modify the base image.
FROM ubuntu:20.04

# Install openssh-server.
RUN apt-get update && apt-get install -y openssh-server git && apt-get
clean && mkdir -p /var/run/sshd
```

Note:

If the base image is a CentOS system, which uses `yum/dnf` for package management, adjust the installation command accordingly.

Specifications for dev machine Images

If you want a custom image to start an instance in dev machine , in addition to meeting the basic image specifications mentioned above, it is recommended to install the Python environment.

Here is an example:

```
# Modify the base image.
FROM ubuntu:20.04

# Install openssh-server.
RUN apt-get update && apt-get install -y openssh-server && apt-get clean
&& mkdir -p /var/run/sshd

# Install python3 and pip3.
```

```
RUN apt-get update && apt-get install -y python3.8 python3.8-distutils  
curl && \  
    curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py && \  
python3.8 get-pip.py && rm -f get-pip.py
```

Note:

- If the base image already contains the corresponding package, skip the corresponding installation command.
- Here, /home/tione/notebook is the default disk mount path for dev machine. Whether this path exists in the image does not affect its usage within the platform.

Complete Custom Image Dockerfile Example

The following is an example of a complete custom image based on NVIDIA's PyTorch image that supports creating training tasks and dev machine instances. It is recommended to create custom images based directly on this example:

```
# [Recommended] Use NVIDIA's PyTorch image as the base image to be  
compatible with newer open-source libraries and GPU card types.  
FROM nvcr.io/nvidia/pytorch:23.07-py3  
  
# [Recommended] Change the software repository. In Tencent Cloud, it is  
recommended to use a private network repository.  
# [Tencent public network software repository] mirrors.tencent.com  
# [Tencent Cloud private network software repository]  
mirrors.tencentyun.com  
ENV TENCENT_MIRRORS="mirrors.tencentyun.com"  
RUN sed -i "s/archive.ubuntu.com/${TENCENT_MIRRORS}/g"  
/etc/apt/sources.list && \  
    sed -i "s/security.ubuntu.com/${TENCENT_MIRRORS}/g"  
/etc/apt/sources.list && \  
    pip config set global.index-url http://${TENCENT_MIRRORS}/pypi/simple  
&& \  
    pip config set global.no-cache-dir true && \  
    pip config set global.trusted-host ${TENCENT_MIRRORS}  
  
# [Recommended] If you use NVIDIA's PyTorch image, it is recommended to  
delete the default NVIDIA repository to speed up pip package query and  
installation.
```

```
RUN rm /etc/xdg/pip/pip.conf /etc/pip.conf /root/.pip/pip.conf
/root/.config/pip/pip.conf && pip config unset global.extra-index-url

# [Basic image specifications] Install openssh-server. The SSH login
feature of Notebook and Task-based Modeling both rely on the openssh-
server component.
RUN apt-get update && apt-get install -y openssh-server git && apt-get
clean && mkdir -p /var/run/sshd

# [Recommended] Use tini as the entry point to facilitate the reclamation
of zombie processes.
RUN apt-get update && apt-get install -y tini && apt-get clean
ENTRYPOINT ["/usr/bin/tini", "-g", "--"]

# [Optional - Recommended for HCC-GPU instances] Optimize TCCL RDMA
communication.
# (If you use NVIDIA's PyTorch image, you need to delete the pre-installed
NCCL plugin in /opt/hpcx/nccl_rdma_sharp_plugin/lib.)
RUN wget https://taco-1251783334.cos.ap-shanghai.myqcloud.com/nccl/nccl-
rdma-sharp-plugins_1.2_amd64.deb && \
    dpkg -i nccl-rdma-sharp-plugins_1.2_amd64.deb && rm -f nccl-rdma-
sharp-plugins_1.2_amd64.deb && \
    rm -rf /opt/hpcx/nccl_rdma_sharp_plugin/lib/*

# [Optional] Install Tikit (excluding big data components).
RUN pip install tencentcloud-sdk-python==3.0.955 coscmd==1.8.6.31 && \
    pip install --no-dependencies -U tikit

# [Custom] Install required dependency libraries.
RUN pip3 install accelerate==0.21.0 bitsandbytes==0.40.2 datasets==2.14.1
deepspeed==0.10.0 evaluate==0.4.0 peft==0.4.0 protobuf==3.20.3
scipy==1.10.1 sentencepiece==0.1.99 transformers==4.31.0
```

Note:

- It is recommended to uniformly use Tencent Cloud's software repository for custom images to support faster installation speeds. The above example already includes the configuration method.
- If you need to conduct multi-machine training using HCC-GPU instance types, it is recommended to install the TCCL plugin in the image to optimize RDMA

communication on Tencent Cloud's StarPulse network. The above example already includes the configuration method.

- It is recommended to install Tikit in the dev machine image for convenient submission of training tasks. The above example already includes the minimal installation method. If you need to use big data components and require a full installation of Tikit, you can further see the Tikit documentation [Installation and Initialization](#).
- Currently, variables declared in bash configuration files, such as `.bashrc`, are not supported in training images.

Introduction to the Training Acceleration Feature of Angel

Last updated: 2026-02-26 15:27:38

Introduction to the Training Acceleration Feature

Tilearn-Angel, evolved from tiacc_training, provides LLM training acceleration capabilities compatible with the Hugging Face ecosystem. It supports computational optimization capabilities that combine custom CUDA operators with automatic compilation optimization, and 3D hybrid parallelism (Tensor Parallelism, Pipeline Parallelism, and Data Parallelism) compatible with the Hugging Face ecosystem. It also supports communication acceleration capabilities compatible with native Distributed Data Parallel (DDP). You can use these capabilities directly without modifying their original code or converting their models. Additionally, it supports general training acceleration capabilities such as optimizer fusion, CPU/GPU affinity optimization, and adaptive mixed precision, as well as model compression capabilities. You can enable these capabilities by adding just a few lines of code.

1. Tilearn-Angel LLM Training Acceleration Images

It is recommended to use the latest built-in image of the platform:

tilearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu

Update the latest tilearn.llm and tilearn.ops packages within the image (optional).

```
# tilearn-llm>=0.9.3
# tilearn.ops>=0.2.1.172
pip3 uninstall -y tilearn.llm tilearn.ops
pip3 install tilearn-llm==0.9.3 -i
https://pypi.tuna.tsinghua.edu.cn/simple
pip3 install tilearn.ops==0.2.1.172 -i https://g-bnvx3728-
pypi.pkg.coding.net/tione/tilearn/simple
wget https://tione-public-cos-1308945662.cos.ap-
shanghai.myqcloud.com/tilearn/hybrid_parallel/colossalai-0.3.4.1-cp310-
cp310-linux_x86_64.whl
pip3 install colossalai-0.3.4.1-cp310-cp310-
linux_x86_64.whl
```

When a custom image is used, it must meet one of the following conditions:

- The custom image is built based on pytorch/pytorch:2.1.2-cuda12.1-cudnn8-devel, with torch.version=='2.1.2'.

- The custom image is built based on the platform image `tilearn-llm1.0-torch2.1-angel-vllm1.0-py3.10-cuda12.1-gpu`.

For other cases, contact the acceleration team for support.

2. Tilearn-Angel Use Instructions

For more details, see [Tilearn.llm Use Instructions](#).

2.1. Tilearn-Angel Computation Acceleration

This section primarily introduces the computational optimization capabilities for LLMs. For more information about how to use general training acceleration capabilities (communication optimization, optimizer fusion, CPU/GPU affinity optimization, and adaptive mixed precision), see [Tilearn.llm Use Instructions](#).

Taking the Llama model as an example, the code modifications for using computational optimization are as follows:

```
### TILEARN.LLM
from tilearn.llm.transformers import LlamaForCausalLM

### The model API is consistent with the standard Hugging Face.
model = LlamaForCausalLM.from_pretrained(...)
```

Use the `AutoModelForCausalLM` API.

```
### TILEARN.LLM
from tilearn.llm.transformers import AutoModelForCausalLM

### The model API is consistent with the standard Hugging Face.
model = AutoModelForCausalLM.from_pretrained(...)
```

Note:

- Due to potential conflicts between Baichuan1-13B and Baichuan2-13B, `tilearn.llm.transformers.AutoModelForCausalLM` defaults to enabling Baichuan1-13B. If you need to use Baichuan2-13B, set the environment variable in your training startup script `export TILEARN_LLM_BAICHUAN_13B=2`.
- The following models support acceleration: Llama, Bloom, Baichuan1, and Baichuan2. For more details, see [Tilearn.llm Use Instructions](#).

2.2. Tilearn-Angel 3D Hybrid Parallelism Acceleration

Tilearn-Angel supports 3D hybrid parallelism (TensorParallel, PipelineParallel, and DateParallel) compatible with the Hugging Face ecosystem. It requires no model conversion. You can perform 3D parallel training directly using the Hugging Face Trainer. The usage method is as follows. For more information, see [3D Hybrid Parallelism Notebook Examples](#).

Environment variable configuration

```
export TILEARN_HYBRID_TP_SIZE=1
export TILEARN_HYBRID_PP_SIZE=2
```

Training code configuration

```
### Computational optimization.
from tilearn.llm.transformers import LlamaForCausalLM
from tilearn.llm.transformers import AutoModelForCausalLM
### 3D parallelism.
import tilearn.llm.hybrid_parallel

def main():
    ### The model API is consistent with the standard Hugging Face.
    model = AutoModelForCausalLM.from_pretrained(...)

    run_exp()
```

Recommended parameter configurations for LLM fine-tuning

- Llama-3-8B model (seqlength=4096)

```
# Default parameters for 8 x A100 40G.
GradientAccumulationSteps=64
BatchSize=1
GradientCheckPointing=False
TilearnHybridTPSize=2
TilearnHybridPPSize=2

# Default parameters for 8 x A800 80G.
GradientAccumulationSteps=32
BatchSize=1
GradientCheckPointing=False
TilearnHybridTPSize=1
TilearnHybridPPSize=2
```

```
TilearnHybridZeroStage=1
```

3. Tilearn-Angel Training Acceleration Performance

TILEARN-LLM LLM training acceleration metrics

Baichuan-13B-Chat 8*A800 80G	batch_size	sequence_length	gpu	每个step耗时(s/iter)	tokens / sec / gpu	speedup
deepspeed baseline	2	4096	8	7.69	1065	1
Megatron	tp2 mbs1 gbs32	4096	8	4.08	2021	1.90
tilearn-0.5.9	mbs1 gbs32 gac4	4096	8	3.48	2357	2.21
Baichuan-13B-Chat 8*H800 80G	batch_size	sequence_length	gpu	每个step耗时(s/iter)	tokens / sec / gpu	speedup
deepspeed baseline	2	4096	8	4.02	2038	1
Megatron	tp2 mbs1 gbs32	4096	8	2.02	4055	1.99
tilearn-0.5.6	2	4096	8	2.33	3516	1.73
tilearn-0.5.9	mbs1 gbs32 gac4	4096	8	1.73	4735	2.32

3. Appendix: Tiacc_training Legacy Training Acceleration

tilearn.llm 0.7.12 and later versions as well as tilearn.ops 0.2.0.1 and later versions support the legacy tiacc_training acceleration capabilities. For more information about general training acceleration capabilities, see [Section 2. Tilearn-Angel Use Instructions](#) in this document. If you use the latest platform image tilearn-llm0.4.2-torch2.1-deepspeed0.10.0-py3.10-cuda12.1-gpu, you can directly use legacy features.

Environment Setup

It is recommended to use the built-in PyTorch image and TensorFlow image of the platform:

PyTorch image: `ti-acc2.0-torch1.9-py3.8-cuda11.1-gpu`

TensorFlow image: `ti-acc1.0-tf1.15-py3.6-cuda10.0-gpu`

Using DDP Distributed Training Communication Optimization (PyTorch + DDP)

Start the training script in a way compatible with native DDP without modifying the training code.

Sample startup command:

```
python3 -u -m tiacc_training.distributed.launch --nproc_per_node
$GPUS_PER_NODE --nnodes $NNODES --node_rank $NODE_RANK --master_addr
$MASTER_ADDR --master_port $MASTER_PORT main.py
```

Measured performance of DDP distributed training communication optimization:

(Acceleration effects are noticeable only in multi-machine multi-GPU card scenarios. The performance is identical to native DDP in single-machine multi-GPU card scenarios.)

Hardware Environment	Model	Number of GPU Cards	Native DDP (examples/sec per V100)	TI-ACC Communication Optimization (examples/sec per V100)
Tencent Cloud GN10Xp.20XLA RGE320	resnext50_32x4d	1 (single-machine)	227	227
		8 (single-machine)	215	215
		16 (two-machine)	116	158.6

Using Adaptive Mixed Precision Optimization (PyTorch)

```
import torch.cuda.amp as amp
import tiacc_training.torch
scaler = amp.GradScaler()
# Instantiate an object of the adaptive mixed precision policy class.
policy =
tiacc_training.torch.tiacc_torch_warp.MixedPrecision_TrainingPolicy(policy
, start_step, hold_step, end_step, interval_time, interval_hold_time)
# Determine whether to enable adaptive mixed precision for the current
epoch based on the input parameters.
mixed_precision =
policy.enable_mixed_precision(epoch, lr=lr, loss=loss, scaler=scaler)
with amp.autocast(enabled=mixed_precision):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
scaler.scale(loss).backward()
scaler.step(optimizer)
scaler.update()
```

Measured performance of adaptive mixed precision optimization:

Hardware Environment	Model	Number of GPU Cards	Native PyTorch (examples/sec per V100)	TI-ACC Data I/O Optimization (examples/sec per V100)	TI-ACC Data I/O + Adaptive Mixed Precision Optimization (examples/sec per V100)
Tencent Cloud GN10Xp.20XLA RGE320	resnet50 mmcls	8 (single-machine)	70.8	350.5	379.2
	centernet mmdet	8 (single-machine)	26.4	28.6	30.6

Using Optimized Embedding Variable Construction (TensorFlow + PS)

```
# Start the container.
docker run -itd --name tiacc-rec-fm --network=host --ipc=host
ccr.ccs.tencentyun.com/ti-platform/tensorflow:1.15.5-py3-rec-0121
# Access the container.
docker exec -it tiacc-rec-fm bash
# Method to use native TensorFlow embedding.
cd wideanddeep && bash start_all.sh --fm
# Method to use TI-ACC lookup optimization.
cd wideanddeep && bash start_all.sh --tiacc --fm
```

Measured performance of optimized embedding variable construction and lookup computation:

Hardware Environment	Model	Number of GPU Cards	Native TensorFlow (global_steps/sec per V100)	TI-ACC After Optimization (global_steps/sec per V100)
Tencent Cloud GN10Xp.20XLA RGE320	DeepFM	16 (two-machine)	41.9 – 56	96.1 – 103.3

	Wide & Deep	16 (two-machine)	49.9 – 69	120 – 128
--	-------------	------------------	-----------	-----------

Acknowledgement

The Tilearn-Angel acceleration engine and related demo case studies benefit from [Deepspeed](#), [ColossalAI](#), [Transformers](#), [LLaMA-Factory](#), [flash-attention](#), and [PyTorch](#). We thank the authors of the above projects for their contributions.

Training Acceleration Class/Function Description

1. Tilearn.llm LLM Training Acceleration

For more information about how to use the training acceleration capabilities (computational optimization, communication optimization, optimizer fusion, CPU/GPU affinity optimization, and adaptive mixed precision), see [Tilearn.llm Use Instructions](#).

1.1. Computational Optimization Related APIs

The computational optimization API is fully compatible with Hugging Face. Taking the Llama model as an example, the usage is as follows:

```
### TILEARN.LLM
from tilearn.llm.transformers import LlamaForCausalLM

### The model API is consistent with the standard Hugging Face.
model = LlamaForCausalLM.from_pretrained(...)
```

1.2. 3D Hybrid Parallelism Optimization Related APIs

The 3D hybrid parallelism optimization feature is fully compatible with the Hugging Face ecosystem. You can use the Hugging Face Trainer without model conversion. Taking the Llama 3 model as an example, the usage is as follows:

Environment variable configuration

```
export TILEARN_HYBRID_TP_SIZE=1
export TILEARN_HYBRID_PP_SIZE=2
```

Training code configuration

```
### Computational optimization.
```

```
from tilearn.llm.transformers import LlamaForCausalLM
from tilearn.llm.transformers import AutoModelForCausalLM
### 3D parallelism.
import tilearn.llm.hybrid_parallel

def main():
    ### The model API is consistent with the standard Hugging Face.
    model = AutoModelForCausalLM.from_pretrained(...)

    run_exp()
```

Recommended parameter configurations for LLM fine-tuning

- Llama-3-8B model (seqlength=4096)

```
# Default parameters for 8 x A100 40G.
GradienAccumulationSteps=64
BatchSize=1
GradientCheckPointing=False
TilearnHybridTPSize=2
TilearnHybridPPSize=2
TilearnHybridZeroStage=1

# Default parameters for 8 x A800 80G.
GradienAccumulationSteps=32
BatchSize=1
GradientCheckPointing=False
TilearnHybridTPSize=1
TilearnHybridPPSize=2
TilearnHybridZeroStage=1
```

2. Tiacc_training Legacy Training Acceleration

tilearn.llm 0.7.12 and later versions as well as tilearn.ops 0.2.0.1 and later versions support the legacy tiacc_training acceleration capabilities. For more information about general training acceleration capabilities, see [Section 2. Tilearn-Angel Use Instructions](#) in this document. If you use the latest platform image tilearn-llm0.4.2-torch2.1-deepspeed0.10.0-py3.10-cuda12.1-gpu, you can directly use legacy features.

tiacc_training.distributed.launch Function

Initializes DDP communication acceleration optimization. The API is fully consistent with native `torch.distributed.launch`. It automatically adapts native DDP-related functions to call TI-ACC communication acceleration capabilities. The main native DDP-related modules/classes include `torch.distributed` and `torch.nn.parallel.DistributedDataParallel()`.

adaptfp16.MixedPrecision_TrainingPolicy Class

Instantiates an adaptive policy for adaptive mixed precision during training. Adaptive policies include time-based mixed precision, time-learning rate mixed precision policy, and loss function mixed precision policy.

Initialization parameters:

Parameter	Type	Required	Parameter Description	Example	Default Value
policy	INT	Yes	Adaptive mixed precision policy. 0: time-based mixed precision, suitable for general adaptive scenarios. 1: time-learning rate mixed precision policy, suitable for scenarios where abnormal loss fluctuations occur during a specific training phase. 2: loss function mixed precision policy, suitable for scenarios where the loss decreases too rapidly or too slowly during training.	0	None
start_time	INT	No	Start time for enabling adaptive mixed precision. Generally, it is recommended to set the start time to 10. This parameter is required when the policy value is set to 0 or 1, and is optional when the policy value is set to 2.	10	10

end_time	INT	No	End time for enabling adaptive mixed precision. Generally, it is recommended to set the end time to the last epoch time. This parameter is required when the policy value is set to 0 or 1, and is optional when the policy value is set to 2.	1000	None
hold_time	INT	No	Hold time for enabling policy 1. During the hold time, a unified policy is used: either enabled or disabled. Generally, it is recommended to set the hold time to the duration of abnormal loss fluctuations during training. This parameter is required when the policy value is set to 1, and is optional when the policy value is set to 0 or 2.	20	None
interval_time	INT	No	Interval for enabling policy 2. The default value is 1000, which indicates that policy 2 is enabled every 1,000 epochs. This parameter is required when the policy value is set to 2, and is optional when the policy value is set to 0 or 1.	1000	1000
interval_hold_time	INT	No	Hold time after policy 2 is enabled within the interval_time. The default value is 100. For example, if interval_time is 1000, policy 2 is enabled during 1000–1100, 2000–2100... This parameter is required when the policy value is set to 2, and is	100	100

			optional when the policy value is set to 0 or 1.		
--	--	--	--	--	--

Instantiated object:

Object	Type	Object Description
policy	MixedPrecision_TrainingPolicy class	An instantiated object of the adaptive policy for automatic mixed precision during training.

enable_mixed_precision Function Method

This function belongs to the MixedPrecision_TrainingPolicy class and determines whether to enable adaptive mixed precision for the current epoch based on the input parameters.

Input parameters:

Parameter	Type	Required	Parameter Description	Example	Default Value
epoch	INT	Yes	Current epoch.	20	None
scaler	torch.cuda.amp.GradScaler	Yes	Instantiated object for gradient scaling.	scaler	None
lr	float	No	lr is the learning rate for the current epoch.	0.01	None
loss	float	No	loss is the loss value from the previous epoch.	0.1	None

Output parameters:

Output Parameter	Type	Parameter Description
mixed_precision	BOOL	Determines whether to enable adaptive mixed precision for the current epoch based on the input parameters. Returns TRUE if enabling is required; otherwise, returns FALSE.

Implementing Resource Isolation Between Sub-users Based on Tags

Last updated: 2026-03-12 10:39:15

Overview

In enterprise multi-team collaboration scenarios, a root account needs to allocate differentiated resource access permissions to sub-users from different teams, allowing them to view and manage only specified resources on Tencent Cloud TI-ONE Platform (TI-ONE) and other associated cloud resources, such as Cloud File Storage (CFS) instances. To achieve effective resource isolation, you can categorize and authorize resources using tag-based authorization and configure corresponding policies in Cloud Access Management (CAM).

Scenario Description

Taking resource groups and online services as examples, assume that a root account has two teams using the TI-ONE product, with two resource groups, two online services, and two CFS instances. The following table shows the corresponding information.

Resource Group ID	Online Service	Tag	Organization
rsg-wwb2hgrs	llm_text	team:llm_text	Team llm_text
rsg-zbqrvlz	llm_picture	team:llm_picture	Team llm_picture

Expected Results

Resources such as resource groups and online services are isolated between sub-users.

1. Resource groups

When you use an administrator account to view the resource group list in the Shanghai region, all resource groups are displayed.

When you use the llm_text_user account to view the resource group list in the Shanghai region, only resource groups bound to the team:llm_text tag can be viewed.

2. Online services

When you use an administrator account to view the online service list in the Shanghai region, all online services can be viewed. When you use the llm_text_user account to view the online service list in the Shanghai region, only online services bound to the team:llm_text tag can be viewed.

Prerequisites

You Have Confirmed That the APIs Support Resource-Level Access Control

The prerequisite for implementing tag-based authorization is that the APIs must support resource-level access control. Most TI-ONE resources support resource-level authorization. For a list of all APIs that support resource-level authorization.

You Have Created Sub-users and Granted Permissions

Create and authorize CAM sub-users in advance: `llm_text_user` for the `llm_text` team and `llm_picture_user` for the `llm_picture` team.

Operation Steps

Step 1: Creating a Tag and Binding It to Resources

This step primarily prepares tag rules for subsequent permission configuration to achieve resource categorization.

1. Create a tag.
Go to the Tag List page. Click [Create Tag](#). Enter the tag key and value, and then click **OK**.
2. Bind the tag to a resource group.
On the Resource Group Management page, select the resource group to be tagged, and click [Edit](#). In the tag selection area, select the previously created team tag key, bind the corresponding tag value, and click **OK**.
3. Bind tags to an online service.
On the **Model Services > Online Services** page, select the service to be tagged, and click **Edit Tags**. In the tag selection area, select the previously created team tag key, bind the corresponding tag value, and click **OK**.

Step 2: Creating a Custom Policy

You can write custom policies to meet your permission requirements. This step involves creating custom policies to ensure sub-accounts of different teams can only access the specific resources they are authorized for.

APIs of TI-ONE vary in permission control granularity: some support tag-based resource-level authorization, while others only support API-level general access control. Furthermore, when executing AI tasks, the platform relies on other cloud services (such as querying CFS instances), requiring corresponding service authorization. Therefore, to achieve resource isolation for TI-ONE, you need to configure the following three policies:

Nu mb	Policy Name	Category	Policy Description
-------	-------------	----------	--------------------

er			
1	Policy_llm_text_tag	Custom policy	Policy used when TI-ONE APIs support tag-based resource-level authorization
2	Policy_tione_required	Custom policy	Policy used when TI-ONE features rely on the APIs of other cloud products
3	QcloudTIONEOperational Precondition	Preset policy	Policy used when TI-ONE APIs support API-level authorization

1. Create the custom policy **Policy_llm_text_tag**.

Go to the Policies page of the CAM console and click [Create Custom Policy](#).

Select **Authorize by Tag**.

On the **Edit Policy** page, select **Visual Policy Generator**.

- Service (required): Select the product to be authorized: **TI-ONE**.
- Action (required): Select the operations you want to authorize. By default, **All Operations (tione:*)** is selected.
- Select Tag (required): Select the **team** tag key and **llm_text** tag value created in Step 1.
- Select Condition Key: **resource_tag** is selected by default with the condition operator set to **OR**. **request_tag** is not selected by default.
- Whether to grant the permission "resource" (required): "*" for APIs that do not support tags: Select **No**.

Description

- All APIs of the service are included in the operations allowed. You can use **Whether Authorization by Tag** to filter APIs and check whether they support tag-based authorization.
 - Yes: APIs support authorization by tag and have the operation permissions on resources associated with the tags.
 - No: APIs do not support authorization by tag and have operation permissions on all resources.

Click **Next** to go to the Associate User/User Group/Role page.

The policy name is automatically generated by the console. The default name is **policygen**.

The suffix number is generated based on the creation date. In this example, change the **policy name** to **Policy_llm_text** and click **Complete**.

2. Create the custom policy **Policy_tione_required**.

Go to the Policies page of the CAM console and click [Create Custom Policy](#). Select **Create by Policy Syntax**.

Select a blank template and click **Next**.

Change the **policy name** to **Policy_tione_required**. Change the **policy content** as follows, and click **Complete**. This policy is modified based on the preset policy QcloudTIONEFullAccessContainMultiservice, removing "tione:*" in the action and the financial payment policy "finance:trade". Other actions can be adjusted as needed according to the table in the Appendix: API Description. By default, the policy grants permissions on all resources ("resource": "*"). To divide resource permissions by tag, use custom policies for authorization by tag.

```
{
  "statement": [
    {
      "action": [
        "cam:GetRole",
        "cam:ListAttachedRolePolicies",
        "vpc:DescribeVpcEx",
        "vpc:DescribeSubnetEx",
        "vpc:DescribeNetworkAcls",
        "vpc:DescribeSecurityGroupPolicies",
        "cls:DescribeLogsets",
        "cls:DescribeTopics",
        "tcr:DescribeInstances",
        "tcr:DescribeNamespaces",
        "tcr:DescribeRepositories",
        "tcr:DescribeRepositoryOwnerPersonal",
        "tcr:DescribeRepositoryFilterPersonal",
        "tcr:DescribeNamespacePersonal",
        "tcr:DescribeInternalEndpoints",
        "monitor:GetMonitorData",
        "cos:GetService",
        "cos:GetObject",
        "cos:GetBucket",
        "cos:HeadObject",
        "cos:OptionsObject",
        "cos:PutObject",
        "tag:DescribeTagKeys",
        "tag:DescribeTagValues",
```

```

        "tag:AttachResourcesTag",
        "tag:DetachResourcesTag",
        "tag:GetResources",
        "tag:DescribeEffectivePolicy",
        "cfs:DescribeCfsFileSystems",
        "emr:DescribeInstances",
        "cvm:DescribeAddresses",
        "cvm:DescribeInstances",
        "emr:DescribeInstancesList",
        "cfs:DescribeMountTargets",
        "goosefs:DescribeClusters",
        "goosefs:DescribeNamespaces",
        "goosefs:DescribeFileSystems",
        "cvm:DescribeSecurityGroupPolicys",
        "clb:DescribeListeners",
        "clb:DescribeLoadBalancers",
        "cam:ListMaskedSubAccounts",
        "monitor:DescribeAlarmPolicies",
        "vpc:DescribeSecurityGroupLimits",
        "cam:ListGroupForConsole"
    ],
    "effect": "allow",
    "resource": "*"
}
],
"version": "2.0"
}

```

Step 3: Associating the Policy with Sub-users

This step is primarily used to associate the custom policies and preset policy created in the preceding step with the sub-users.

On the User List page, find the sub-user [llm_text_user](#) to associate with policies, and click Authorize on the right.

Select [Policy_llm_text_tag](#), [QcloudTIONEOperationalPrecondition](#), and [Policy_tione_required](#), and then click Complete.

To associate policies with a user group, perform the following steps: On the User Group List page, find the target user group. Click the group name to go to the user group details page. Click Associate Policy in the lower part of the page. Select the above three policies, and click OK.

Note:

1. Ensure that the sub-user is not associated with other CAM policies for TI-ONE. Since CAM policies are combined into a union, other TI-ONE policies may cause resource isolation to fail.
2. To associate policies with a user group, see Authorization Management.

Step 4: Creating a Tag Policy

This step is primarily used to achieve automated tag management. Based on a user's tags, the system automatically associates the user's tag key-value pairs during product operations. Tag policies enable automatic association of tags when resources are created or modified. For specific configuration, see [Creating Tag Policies](#).

Note:

- Select the tag team:llm_text created in Step 1 of this section. Enter the tag key team.
- For Product Region, select All Products.
- Enable auto-assignment and enforce.
- After creation, select this policy and bind it to the corresponding sub-account.

Step 5: Verifying

This step is primarily used to verify whether the preceding steps have taken effect.

Log in to the TI-ONE console with the sub-user `llm_text_user` associated with the policies. The sub-user can only view and manage resources associated with the tag team:llm_text, which meets the expectations.

Authorization for the llm_picture team also follows the method described above.

Isolation of CFS Instances and Other Storage Resources

Two methods are available for CFS storage resource isolation. Select an isolation method based on your actual usage scenario.

Method 1: CFS Instance Isolation

When different teams require a higher level of data security and performance isolation, resource isolation can be implemented at the CFS instance level. The specific method is: create an independent CFS instance for each team, and then use the tag-based authorization feature to grant each team's sub-accounts the permissions to access only the CFS instance with specified tags. For specific operation steps, see [CAM > Creating Custom Polices Through Tag Authorization](#).

Method 2: Controlling CFS Mount Permissions at the Directory Level

When multiple teams need to share the same CFS instance but require data directory isolation, control can be implemented through directory-level permissions. The specific method is: create independent root directories for different teams within the same CFS instance. When you

configure data sources on TI-ONE, specify the mount directory that each team has permissions to access. For specific operation steps, see [Using the Data Source Feature to Implement Directory-Level Access Control for Storage](#).

Must-Knows

Note:

1. When a sub-account uses a custom policy, if an API permission error occurs for a specific feature on the platform, it is generally because product iteration has introduced new cloud product APIs. You can add the corresponding API permissions to the custom policy based on the error message. For account-related permissions, see CAM > User-Related Issues.
2. If a user is the resource creator, the user is not restricted by tag rules. For more information, see Creator Tag Overview.

Appendix: API Description

API description for the Policy_tione_required policy

Note: Pay special attention to the authorization scope for fields in bold and proceed with caution. Use tags to control the authorization scope when necessary.

Category	API Name	API Description
cam	GetRole	Queries user group details.
cam	ListAttachedRolePolicies	--
cam	ListMaskedSubAccounts	Queries a list of sub-accounts (with sensitive information masked).
cam	ListGroupsForConsole	Queries a list of user groups in the console.
cfs	DescribeCfsFileSystems	Queries file systems.
cfs	DescribeMountTargets	Queries file system mount points.
clb	DescribeListeners	Queries a list of Cloud Load Balancer (CLB) listeners.
clb	DescribeLoadBalancers	Queries a list of CLB instances.
cls	DescribeLogsets	Queries a list of logsets.
cls	DescribeTopics	Queries a list of log topics.

cos	GetService	Queries a list of buckets.
cos	GetObject	Downloads objects.
cos	GetBucket	Queries objects in a bucket.
cos	HeadObject	Checks whether an object exists and returns basic information if it does.
cos	OptionsObject	Sends a preflight request for cross-origin resource sharing (CORS).
cos	PutObject	Uploads objects.
cvm	DescribeAddresses	Queries elastic IP addresses (V3).
cvm	DescribeInstances	Queries a list of instances.
cvm	DescribeSecurityGroupPolicies	Queries security group rules.
emr	DescribeInstances	Queries Elastic MapReduce (EMR) instances.
emr	DescribeInstancesList	Queries a list of EMR cluster instances.
goosefs	DescribeClusters	Queries all Data Accelerator Goose FileSystem (GooseFS) clusters.
goosefs	DescribeNamespaces	Lists all namespaces.
goosefs	DescribeFileSystems	Lists all file systems.
monitor	GetMonitorData	Pulls the monitoring data.
monitor	DescribeAlarmPolicies	Queries a list of alarm policies (V2.0).
tag	DescribeTagKeys	Queries tag keys.
tag	DescribeTagValues	Queries tag values.
tag	AttachResourcesTag	Binds tags to resources in batches.
tag	DetachResourcesTag	Unbinds tags from resources in batches.
tag	GetResources	Queries a list of resources bound to tags.
tag	DescribeEffectivePolicy	Queries the effective policy of a target node.
tcr	DescribeInstances	Queries instance information.

tcr	DescribeNamespaces	Queries namespace information.
tcr	DescribeRepositories	Reads image repositories.
tcr	DescribeRepositoryOwnerPersonal	Queries user-created image repositories.
tcr	DescribeRepositoryFilterPersonal	Obtains personal image repositories that meet the specified search conditions.
tcr	DescribeNamespacePersonal	Queries personal namespace information.
tcr	DescribeInternalEndpoints	Queries the URLs for private network access to an instance.
vpc	DescribeVpcEx	Queries a list of Virtual Private Cloud (VPC) networks.
vpc	DescribeSubnetEx	Queries a list of subnets.
vpc	DescribeNetworkAcls	Queries a list of network ACLs.
vpc	DescribeSecurityGroupPolicies	Queries security group rules.
vpc	DescribeSecurityGroupLimits	Queries the security group quota.

Using the Data Source Feature to Implement Directory-Level Access Control for Storage

Last updated: 2026-02-26 15:44:24

1. Overview

During AI model training, evaluation, and inference, you may need to mount other cloud storage products to read, write, and manage relevant data. Within an enterprise, multiple users may need to access the same storage system/instance, thus requiring access permission management for storage.

This document describes how to implement access control for storage directories based on the data source management feature of Tencent Cloud TI-ONE Platform (TI-ONE), in conjunction with the Cloud Access Management (CAM) and Tag services.

- **Data source management**: You can register specified directories of storage instances (Cloud File Storage (CFS)/Data Accelerator Goose FileSystem extreme (GooseFSx)) as data sources on TI-ONE, enabling quick mounting and access control. Go to [the Data Source Management module in the TI-ONE console](#).
- **CAM**: helps you securely and conveniently manage access to Tencent Cloud services and resources. With CAM, you can create sub-users, user groups, and roles, and control their access scope through policies. CAM supports SSO capabilities for users and roles, allowing you to set up interoperability between enterprise users and Tencent Cloud based on specific management scenarios. [Log in to the CAM console](#).
- **Tag**: Tag is a cloud resource management tool provided by Tencent Cloud. You can categorize, search, and aggregate cloud resources with the same characteristics from different dimensions, thus easily managing cloud resources. [Log in to the Tag console](#).

2. Applicable Scenarios and Expected Results

In enterprise multi-user collaboration scenarios, platform administrators need to restrict the data scope that different team members can view and manage to prevent unauthorized access. For example, Team A members should only access Team A's storage instances and file directories, and Team B members should only access Team B's files.

1. Scenario example

- Assume an enterprise has two business teams: Team A is responsible for LLM training, with members including user_a1, user_a2, and so on. Team B is responsible for LLM inference, with members including user_b1, user_b2, and so on.

- Also, assume the enterprise has purchased a CFS instance and created dedicated storage directories /home/Group_A and /home/Group_B for Team A and Team B, respectively.

Team	Responsibility	Member	Dedicated Storage Directory
Team A (UserGroup_A)	LLM training	user_a1, user_a2, and so on	/home/Group_A
Team B (UserGroup_B)	LLM inference	user_b1, user_b2, and so on	/home/Group_B

2. Expected results

- Members of Team A can only access /home/Group_A.
- Members of Team B can only access /home/Group_B.

3. Prerequisites

1. You have purchased storage instances.

The data source management feature supports only the storage instances of CFS and GooseFSx. Purchase storage instances in the console in advance.

- **CFS**: CFS provides standard NFS and CIFS/SMB file system access protocols. It serves as a shared data source for multiple CVM instances or other computing services, supports elastic scaling of capacity and performance, and can be mounted and used by existing applications without modification. It is a highly available, highly reliable distributed file system suitable for scenarios such as big data analysis, media processing, and content management. [Log in to the CFS console](#).
- **GooseFSx**: GooseFSx accelerates access to Cloud Object Storage (COS), delivering extreme performance and ultra-low latency for high-performance computing workloads. It enables flexible management of hot and cold data, making it suitable for scenarios such as high-performance computing, autonomous driving, and machine learning. [Log in to the GooseFSx console](#).

2. You have created users and user groups.

Log in to the [CAM console](#). Create the following users and user groups and bind team members. For the specific operation steps, see [Creating Sub-users](#).

- Create user groups: UserGroup_A and UserGroup_B.
- Create users: user_a1, user_a2, user_b1, and user_b2.

4. Operation Steps

TI-ONE provides the capability to control data isolation between Tencent Cloud sub-accounts based on CAM and Tag. By adding specific tags to data sources and configuring corresponding

CAM policies, you can achieve data isolation between different Tencent Cloud sub-accounts, thereby implementing access control for storage directories. The specific operation steps are as follows:

Step 1: Creating a Tag

Go to the [Tag List](#) page. Create the tags team=UserGroup_A and team=UserGroup_B.

Step 2: Creating Data Sources and Adding Tags

1. Go to the [Data Source Management](#) page. Click **Create Data Source**.

Data Source Name	Storage Directory	Tag
Data source of Group_A	/home/Group_A	team:UserGroup_A
Data source of Group_B	/home/Group_B	team:UserGroup_B

2. Enable **mount control** for your storage instance. The following figure shows the configuration result.

Step 3: Creating Custom CAM Policies and Authorizing User Groups

1. Go to the [Policies](#) page of the CAM console. Click **Create Custom Policy** and select **Create by Policy Syntax**.

2. On the [Select Policy Template](#) page, select **Blank Template** and click **Next**.

3. On the policy editing page (see the image on the right side of the preceding step), configure the following parameters, and click **Complete**.

3.1 Policy Name (required): UserGroup_A_DataSource

3.2 Policy Content (required): as shown below. (Note: When you add other policies, replace the tag key-value in qcs:resource_tag as needed.)

```
{
  "statement": [
    {
      "action": [
        "tione:DescribeDataSources",
        "tione:DescribeDataSource",
        "tione>DeleteDataSource",
        "tione:UpdateDataSource",
        "tione:DescribeDataSourcePathStatus",
        "tione:DescribeDataSourcePathDetails",
```

```

        "tione:DescribeDataSourceRealPath",
        "tione:CreateDataSourcePath"
    ],
    "condition": {
        "for_any_value:string_equal": {
            "qcs:resource_tag": [
                "team&UserGroup_A"
            ]
        }
    },
    "effect": "allow",
    "resource": "*"
},
{
    "action": [
        "tione:CreateDataSource",
        "tione:CreateMountLimit",
        "tione:UpdateMountLimit",
        "tione>DeleteMountLimit",
        "tione:DescribeMountLimits",
        "tione:DescribeMountInstance",
        "tione:DescribeMountInstances"
    ],
    "effect": "allow",
    "resource": "*"
}
],
"version": "2.0"
}

```

4. Assign the custom policy **UserGroup_A_DataSource** to the user group **UserGroup_A**.
5. Similarly, follow the above process to create another custom policy **UserGroup_B_DataSource** and assign it to the user group **UserGroup_B**.

Step 4: Verifying the Result

1. Log in to the TI-ONE console with the sub-user account `user_a1` or `user_a2`. Go to the [Data Source Management](#) page. The list displays only the data source of `Group_A`, not the data source of `Group_B`.

2. Conversely, sub-users user_b1 and user_b2 can only view the data source of Group_B on this page.
3. Similarly, when data sources are mounted in business modules such as Task-based Modeling and Dev Machines, the drop-down list displays only data sources accessible to the corresponding team.

Guide on Using CFS Instances in Various Scenarios

Last updated: 2026-02-26 15:53:54

This document introduces how to use Cloud File Storage (CFS) instances in various scenarios and functional modules of Tencent Cloud TI-ONE Platform (TI-ONE).

1. CFS Introduction

CFS is a distributed file storage system provided by Tencent Cloud. It offers scalable shared file storage services and can be used in conjunction with other Tencent Cloud services, such as Cloud Virtual Machine (CVM). CFS supports the standard Network File System (NFS) protocol, serving as a shared data source for multiple CVM instances. It supports elastic scaling of capacity and performance. Existing applications can mount and use CFS instances without modification. It is a highly available, highly reliable distributed file system, suitable for scenarios such as big data analysis, media processing, and content management.

2. Basic Usage of CFS

CFS storage is used by mounting. CFS supports mounting on both Linux and Windows operating systems. Since TI-ONE primarily uses Linux, this document only covers the usage of CFS on Linux.

CFS instances are mounted on Linux using the mount command. For more information, see [Using CFS Instances on Linux Clients](#). The mount command requires specifying the source directory of the partition to be mounted and the mount path on the target system (such as a CVM instance or a container), enabling access to the partition's files on the target system.

On TI-ONE, you do not need to deal with the details of the mount command. You only need to specify the CFS path and the target mount path when using the Task-based Modeling or Dev Machines module.

3. Networking Modes for CFS

CFS storage specifications include the **General series** and the **Turbo series**. The Turbo series features high IOPS and is more suitable for high-performance computing scenarios in AI. For more information about the differences between CFS specifications, see [Storage Types and Specifications](#). You can purchase an appropriate instance based on your requirements and budget.

It is important to note that:

- **CFS General series supports only the Virtual Private Cloud (VPC) networking mode:** When creating a CVM instance, you need to specify the VPC network where the CFS instance resides. Subsequent CVM instances that access this CFS instance need to be within the VPC

network or interconnected with it through Cloud Connect Network (CCN). For more information, see [Cross-AZ and Cross-Network Guide](#).

- **CFS Turbo series supports both the VPC networking mode and the CCN networking mode.** For more information, see [Selecting a Network for Turbo CFS](#).
 - **VPC networking mode:** When you create a CFS instance, specify the VPC network to which the CFS instance belongs. Subsequent CVM instances that access this CFS instance need to be within the VPC network or interconnected with it through CCN.
 - **CCN networking mode:** When you create a CFS instance, specify a CCN instance. All VPC networks added to this CCN instance can access the CFS instance.

4. CFS Usage Patterns in TI-ONE Scenarios

When you use TI-ONE in scenarios such as deep learning or LLM training/fine-tuning, we recommend that you use CFS to store training/inference materials, including **code, training models, training data sets**, and so on.

TI-ONE supports the following scenarios:

- **Pay-as-you-go:** You use CVM instances provided by TI-ONE for dev machine development or task-based modeling, and you need to mount your own CFS storage to the CVM instances. In this mode, TI-ONE is responsible for establishing network connectivity between the CVM instances and your CFS storage.
- **Managing your CVM instances based on yearly/monthly subscription:** After purchasing CVM instances in the CVM console, you can manage these instances on TI-ONE for dev machine development or task-based modeling. You need to mount your own CFS storage to your CVM instances. In this mode, you must ensure network connectivity between your instance resources and CFS storage.

5. Differences in Using CFS with Different Networking Modes in TI-ONE Scenarios

Scenario 1: Pay-as-You-Go Mode for TI-ONE + VPC Networking Mode for CFS

CVM instances are under the TI-ONE account, while the CFS instances are under your own account. They are inherently in different VPC networks. Therefore, **TI-ONE automatically establishes network connectivity between its VPC network and the VPC network where your CFS instance resides.** You do not need to deal with the network connectivity details. In this scenario, **you only need to specify any VPC network under your account when creating your CFS instance.**

 **Note:**

If you need to mount multiple CFS instances within a single dev machine or a single task-based modeling task, you need to ensure these CFS instances are in the same VPC network or within the same CCN instance..

Scenario 2: Pay-As-You-Go Mode for TI-ONE + CCN Networking Mode for CFS

CVM instances are under the TI-ONE account, while the CFS instances are under your own account, and this CFS instance has been added to a CCN instance under your account. TI-ONE automatically establishes CCN network connectivity between its VPC network and your CFS instance. You do not need to deal with the network connectivity details.

Note:

- Due to certain limitations of CCN, failures may occur in this scenario. If you encounter issues, you can contact us for support and troubleshooting.
- If you need to mount multiple CFS instances within a single dev machine or a single task-based modeling task, you need to ensure these CFS instances are in the same VPC network or within the same CCN instance..

Scenario 3: Self-Managed Resource Management Mode for TI-ONE + VPC Networking Mode for CFS

Both the CVM instances and the CFS instance are under your own account. You need to ensure network connectivity between the CFS instances and the CVM instances. Connectivity can be achieved either by placing them in the same VPC network or by establishing connectivity through CCN. For more information, see [Cross-AZ and Cross-Network Access Guide](#) .

Note:

If you need to mount multiple CFS instances within a single dev machine or a single task-based modeling task, you need to ensure network connectivity between these CFS instances and the CVM instances you manage on TI-ONE.

Scenario 4: Self-Managed Resource Management Mode for TI-ONE + CCN Networking Mode for CFS

Both the CVM instances and the CFS instances are under your own account. You need to ensure network connectivity between the CFS instances and the CVM instances. You can achieve this by adding the VPC network containing your CVM instances to the CCN instances where your CFS instance resides. For more information about CCN, see CCN Overview.

Note:

If you need to mount multiple CFS instances within a single dev machine or a single task-based modeling task, you need to ensure network connectivity between these CFS instances and the CVM instances you manage on TI-ONE.

6. Different Usage Scenarios for Mounting CFS Instances

Scenario 1: Using a CFS Instance in a Dev Machine

After a CFS instance is created, the default root directory `/` is empty. You can mount the CFS instance to a CVM instance and then upload or create files, or you can mount and use the CFS instance in a dev machine.

When you create a dev machine, to ensure the dev machine can be used properly, the first CFS storage path you select is mounted to the default working directory `/home/tione/notebook` of the dev machine.

You can specify the root directory of the CFS instance or any directory you created as the source path. The contents in the specified directory are mounted to `/home/tione/notebook`. The following examples show the mappings between the mount path and the dev machine path.

- Example 1:

CFS Source Path /	Dev Machine Path /home/tione/notebook
<code>/</code>	<code>/home/tione/notebook</code>
<code>/data</code>	<code>/home/tione/notebook/data</code>
<code>/model</code>	<code>/home/tione/notebook/model</code>
<code>/code</code>	<code>/home/tione/notebook/code</code>

- Example 2:

CFS Source Path /user01	Dev Machine Path /home/tione/notebook
<code>/user01</code>	<code>/home/tione/notebook</code>
<code>/user01/data</code>	<code>/home/tione/notebook/data</code>
<code>/user02</code>	Inaccessible

Scenario 2: Using a CFS Instance in Data Preprocessing

Data Preprocessing Input

Assume you have created a dev machine for a data preprocessing pipeline in the single-turn Q&A format and mounted a CFS instance. The CFS source directory for mounting is `/`, and the target mount directory is `/home/tione/notebook`.

After accessing the dev machine, the system **creates script tools for data preprocessing and their corresponding directories** under your mounted source directory / (that is, the mount directory /home/tione/notebook of the dev machine). You can upload relevant data to this directory for processing.

The directory format of the script tool is mount path/<nb-id>/<ppl-type>/raw_dataset_files.

Where:

- **ppl-type is the selected data preprocessing pipeline type.** `single_round_qa_pipeline` indicates a single-turn Q&A pipeline. `multi_round_qa_pipeline` indicates a multi-turn Q&A pipeline. `unsupervised_pipeline` indicates an unsupervised pipeline.
- **nb-id is the ID of the current dev machine.** You can view the ID via the **URL**. **Do not be misled by the / directory displayed in the dev machine file bar.** The default directory for opening the dev machine is /home/tione/notebook, which is the source directory you mounted. You can open the terminal and enter the `pwd` command to view the current path.
- `single_round_qa_pipeline` is the script path for the single-turn Q&A pipeline. When you select a multi-turn or unsupervised pipeline, the path is different.
- After accessing the `ppl-type` directory, you can refer to the **quick_start.ipynb** guide to perform further data preprocessing operations.

Data Preprocessing Output

After you complete the data preprocessing operations, since the CFS instance is mounted, **the actual output files are already stored in your CFS storage.** Therefore, if you need to use them in Task-based Modeling, you need to map the CFS path.

- Example 1:

CFS Source Path	/
Dev Machine Output Path	/home/tione/notebook/nb-1023096325327389312/single_round_qa_pipeline/final_dataset_files/SelfMade_NL2SQLpilot_for_training.jsonl
Actual CFS Path	/nb-1023096325327389312/single_round_qa_pipeline/final_dataset_files/SelfMade_NL2SQLpilot_for_training.jsonl

- Example 2:

CFS Source Path	/user01
-----------------	---------

Dev Machine Output Path	/home/tione/notebook/nb-1023096325327389312/single_round_qa_pipeline/final_dataset_files/SelfMade_NL2SQLpilot_for_training.jsonl
Actual CFS Path	/user01/nb-1023096325327389312/single_round_qa_pipeline/final_dataset_files/SelfMade_NL2SQLpilot_for_training.jsonl

Scenario 3: Using a CFS Instance in Task-based Modeling

When using Task-based Modeling, you can store the corresponding code, model files, and data sets in a CFS instance and access them by mounting.

Note the mapping relationships of mount directories. Taking an LLM fine-tuning case as an example, you can see [Fine-Tuning a Custom LLM Using Task-based Modeling](#).

Assume that during the dev machine phase, after you debug the following code, model, and data, the final data resides under the CFS path `/data/custom`.

File Type	CFS Source Path <code>/data/custom</code>	Dev Machine Mount Path <code>/home/tione/notebook</code>
model	<code>/data/custom/model</code>	<code>/home/tione/notebook/model</code>
data	<code>/data/custom/llm/data</code>	<code>/home/tione/notebook/llm/data</code>
code	<code>/data/custom/llm/code</code>	<code>/home/tione/notebook/llm/code</code>

Then you proceed with task-based modeling. Since task-based modeling differs from dev machines, data is typically mounted under `/opt/ml` for processing.

File Type	CFS Source Path	Mount Path of Task-based Modeling
model	<code>/data/custom/model</code>	<code>/opt/ml/pretrain_model</code>
dataset	<code>/data/custom/llm/data</code>	<code>/opt/ml/input/data/train</code>
code	<code>/data/custom/llm/code</code>	<code>/opt/ml/input/data/code</code>
output	<code>/data/custom/llm/output</code>	<code>/opt/ml/output</code>

Note:

In the task-based modeling code, when you need to read data or models, use the mount path of task-based modeling. You can see [Fine-Tuning a Custom LLM Using Task-based Modeling](#), download the sample code according to the guide, and test it.

Scenario 4: Using a CFS Instance in Online Services

Loading Models Directly from a CFS Instance

In LLM scenarios, models are generally large. The recommended workflow is: use a dev machine mounted with a CFS instance to debug training code; use Task-based Modeling mounted with the CFS instance to execute distributed training tasks; finally, store the fine-tuned model on the CFS instance. The Online Services module for TI-ONE inference scenarios supports reading model files directly from the CFS instance. In this case, you only need to specify the absolute path of your model files in the CFS instance.

Note:

Some bare metal physical servers (such as HCC A100 and HCC V100 instance types) do not support elastic network interfaces and therefore cannot support mounting CFS Turbo instances.

Example:

After task-based modeling is completed, if we store the trained model to the path `/opt/ml/output/model/saved_model.pb`, then the corresponding CFS path is `/data/custom/llm/output/model/saved_model.pb`.

	CFS Source Path	Mount Path of Task-based Modeling
output	<code>/data/custom/llm/output</code>	<code>/opt/ml/output</code>

Using a CFS Instance as Temporary Storage When Importing Models from the Model Repository

After your model is registered in the model repository, you can import it from the repository to create an online service.

In the pay-as-you-go billing mode, the default total size of model packages supported by local disks is approximately 45 GB. In the yearly/monthly subscription billing mode, model packages are stored on the disks mounted to the resource group nodes.

When your model package size exceeds the limit, you need to configure an additional CFS instance. The platform automatically uses the CFS instance to store the model. In this case, you need to select a CFS instance and configure an **empty CFS path**. When the online service runs, this path is mounted, and the model files from the model repository are copied to this path for service startup loading.

When the online service is stopped or updated, the platform cleans up the files under this CFS source path by default. Ensure there is no other data in this path.

Note:

Some bare metal physical servers (such as HCC A100 and HCC V100 instance types) do not support elastic network interfaces and therefore cannot support mounting CFS Turbo instances.

7. How to Implement Directory Isolation Between Sub-accounts with CFS and CFS Turbo (Deprecated)

Note:

TI-ONE launched the [data source management](#) feature in July 2025. This feature supports registering directories of storage instances (CFS/GooseFSx) as data sources and configuring mount permissions for them.

After the data source management feature is launched, the solution described in this section, implementing directory isolation between sub-accounts based on tags, becomes deprecated. If you have configured directory permission policies according to this solution, see the data source management documentation for reconfiguration. If you encounter any questions during configuration, [submit a ticket](#) to contact us.

This section describes how to implement directory isolation between different sub-accounts within the same CFS or CFS Turbo instance by using tags.

Step 1: Creating a Tag

1. Log in to the [Tag console](#). In the left sidebar, select **Tag List**. Select the **Custom Tags** tab and click **Create Tag**.
2. Specify the tag keys and values according to the following rules:

	Tag Key	Tag Value	Rule Description
1	tione_cfs_directory_isolation_rulelists	RuleA:RuleB:RuleC	If a CFS or CFS Turbo instance has a tag with the key tione_cfs_directory_isolation_rulelists , it is considered that this instance requires directory isolation between sub-accounts. Otherwise, global shared permissions apply. The tag value can contain multiple rules separated by colons (:). Multiple rules are combined into a union . The specific path for a rule is defined by the tag in No.3.
2	tione_cfs_default_effective_rulelists	RuleA	If a CFS or CFS Turbo instance, on top of the condition in No.1, also has a tag

			<p>with the key <code>tionone_cfs_default_effective_rulelists</code>, then the rule (RuleA) specified in the tag value is considered the default effective rule. This rule takes effect even if a sub-account does not have the tag described in No.4. <u>Only one rule can be configured for this tag.</u> If multiple rules are configured, one will be selected at random, indicating a configuration error.</p>
3	RuleA	PathA:Path B:PathC	<p>This defines a specific isolation rule. The tag key corresponds to a rule name specified in the tag values of No.1, No.2, or No.4. The tag value is the file path on the CFS or CFS Turbo instance to be isolated. Multiple paths are combined into a union. Each path must start with a forward slash (/) and can only contain English letters, digits, underscores (_), hyphens (-), and periods (.). Folder names within the path can only start with English letters, digits, or underscores (_).</p> <p>Special wildcards:</p> <ol style="list-style-type: none"> 1. The path supports the wildcard <code><@subaccount></code>, which represents the sub-account UIN. For example, RuleA: /home/<@subaccount>. You only need to specify one path rule, and the UIN of the sub-account using the resource is automatically populated. 2. The path supports the wildcard <code><@subusername></code>, which represents the sub-account nickname. For example, RuleA: /home/<@subusername>. You only need to specify one tag path rule, and the nickname of the sub-account using the resource is automatically populated.
4	cfsidxx_cfs_directory_isolation	RuleA:Rule B	<p>If a sub-account has a tag with the key <code>cfsidxx_cfs_directory_isolation</code> (where cfsidxx is the ID of a specific CFS or</p>

CFS Turbo instance), then the paths accessible by this sub-account on the instance are the **intersection** of the rules specified in this tag value (No.4) and the rules from tags No.1 and No.2. If a sub-account does not have this tag (No.4), it has no access permissions on this CFS instance.

Step 2: Applying Tags to the CFS or CFS Turbo Instance

1. Log in to the [CFS console](#). In the left sidebar, select **File System**. Select the instance for which you need to configure directory isolation. Click **Edit Tags**. Select the tag keys and tag values created in [Step 1](#).
2. The CFS or CFS Turbo instance attached with the tag key `tionone_cfs_directory_isolation_rulelists` requires directory isolation between sub-accounts. Otherwise, global shared permissions apply.

Step 3: Applying Tags to Sub-Accounts

1. Log in to the [Cloud Access Management \(CAM\) console](#). In the left sidebar, select **Users/User List**. Select the sub-account for which you need to configure directory isolation. Click the **username** to go to the **user details page**. Click the **edit icon for the tag field**. Select the tag keys and tag values created in [Step 1](#).

Step 4: Using the Tagged CFS or CFS Turbo Instance for Training and Inference

1. Log in to the [TI-ONE console](#). In the left sidebar, select **Training Workshop/Task-based Modeling**. Click **Create Task**. Select your tagged CFS or CFS Turbo instance. After selection, the paths defined in the tags will appear. Select the desired path from the drop-down list.
2. Log in to the [TI-ONE console](#). In the left sidebar, select **Training Workshop/Dev Machine**. Click **Create Task**. Select your tagged CFS or CFS Turbo instance. After selection, the paths defined in the tags will appear. Select the desired path from the drop-down list.
3. Log in to the [TI-ONE console](#). In the left sidebar, select **Model Services/Online Services**. Click **Create Service**. Select your tagged CFS instance. After selection, the paths defined in the tags will appear. Select the desired path from the drop-down list.

 **Note:**

1. If a path specified in the tag value does not comply with the rules, it will be **grayed out and unavailable for selection in the console**. Contact your Tencent Cloud account administrator to adjust the path configuration in the relevant tags.
2. Path rules: Each path must start with a forward slash (/) and can only contain English letters, digits, underscores (_), hyphens (-), and periods (.). Folder names within the path can only start with English letters, digits, or underscores (_).

Guide on Data Writing Using TI-ONE Dev Machines

Last updated: 2026-03-12 10:34:28

1. Document Introduction

Document Purpose

This document aims to guide you in standardizing data storage paths when creating dev machines on Tencent Cloud TI-ONE Platform (TI-ONE). This helps prevent issues such as full disks and machine errors caused by writing data to the Cloud Virtual Machine (CVM) system disk, ensuring stable service operation.

Applicable Scope

1. Users who have purchased Tencent Cloud CVM instances and added them to the TI-ONE resource group.
2. Users who plan to use or are currently using TI-ONE to create dev machines.

Overview of the Relationship Between TI-ONE and Tencent Cloud CVM

1. After purchasing Tencent Cloud CVM instances, you need to add them to the TI-ONE resource group for unified management of resources. For details, see [Purchase Methods](#).
2. TI-ONE leverages the CVM instances in the resource group to provide you with features such as dev machine creation, online service deployment, and execution of containerization training tasks.

2. Risks and Issues of Writing Data to the System Disk

Risks at the System Resource Level

- **Host and Tencent Kubernetes Engine (TKE) exceptions caused by rapidly exhausted system disk space:**
 - Insufficient system disk capacity causes abnormal termination of the container process.
 - Node resource pressure triggers the Kubernetes eviction mechanism, causing a service interruption.
- **Overall node operation efficiency affected by the storage I/O performance bottleneck:**
 - The read/write bandwidth of the system disk is occupied by a large amount of data.
 - Increased OverlayFS layers cause degradation in container read-write performance.

Risks at the Data Management Level

- **Unguaranteed data persistence:**
 - The container restart or rebuild causes data loss in the /root directory.
- **Storage resource management difficulties:**
 - Monitoring and isolation of system disk space usage is difficult.
 - System file occupancy and business data occupancy cannot be distinguished.

Risks at the Ops Management Level

- **High troubleshooting complexity:**
 - It is required to perform troubleshooting of storage usage both within the container and on the host simultaneously.
 - Issue location involves a multi-layer storage architecture (container layer, image layer, and host layer).
- **Resource scheduling and cost control issues:**
 - System disk scale-out costs are higher than those of dedicated storage scale-out.
 - Storage resources are wasted and cannot be precisely measured or billed.

Risks at the Business Continuity Layer

- **Affected service availability:**
 - Storage issues in a single container may spread to the node level.
 - Business services may experience cascading failures due to storage issues.
- **Deployment and scale-out efficiency reduction:**
 - Container image bloat affects the deployment speed.
 - Node storage pressure limits the scale-out capability of businesses.

3. Introduction to the Persistent Storage Solution

CFS Solution

Cloud File Storage (CFS): provides a scalable shared file storage service that can be used in combination with Tencent Cloud services such as CVM. CFS provides a standard NFS access protocol, providing a shared data source for multiple CVM instances, and supporting elastic scale-out of capacity and performance. Existing applications can be mounted and used without modification. It is a highly available and reliable distributed file system, which is suitable for scenarios such as big data analysis, media processing, and content management. For details, see [CFS Introduction](#).

ⓘ Instructions on Service Adaptability Between CFS and TI-ONE:

1. CFS can be seamlessly integrated with the Dev Machines, Online Services, and Task-based Modeling modules of TI-ONE, providing a unified data storage feature.

2. After the CFS instance is mounted, the data read–write process is consistent with the local disk operation without the need to modify business code or operation habits.

Data Accelerator Goose FileSystem

Data Accelerator Goose FileSystem (GooseFS) is a highly reliable, available, and elastic data acceleration service launched by Tencent Cloud. Leveraging the cost strengths of Cloud Object Storage (COS) as the storage foundation for data lakes, it provides a unified data lake entry for compute applications within the data lake ecosystem, accelerating performance for large–scale data analysis, machine learning, AI intelligence, and other businesses accessing storage. GooseFS adopts a distributed cluster architecture featuring elasticity, high reliability, and high availability. It provides a unified namespace and access protocol for upper–layer compute applications, facilitating efficient data management and transfer across different storage systems.



Data Accelerator Goose FileSystem extreme

Data Accelerator Goose FileSystem extreme (GooseFSx) is a high–performance, POSIX–compliant data accelerator launched by Tencent Cloud. It can accelerate access to COS, delivering extremely high performance and ultra–low latency for high–performance computing workloads. It enables flexible management of cold and hot data, making it suitable for business scenarios such as high–performance computing, autonomous driving, and machine learning. As a fully managed service, GooseFSx is easy to use, postpaid by hour, and can be released after usage, with COS serving as persistent storage.

The following mainly explains storage specifications and must–knows using the scenarios of CFS.

4. Operation Steps for Mounting CFS and Data Writing in TI-ONE Dev Machine Scenarios

Mounting CFS and Standardizing Data Writing When Creating a Dev Machine

1. Prerequisites:

For example, the CFS instance has been created, and the permissions have been configured.

2. Operation steps for mounting CFS during the process of creating dev machines:

2.1 Go to the Dev Machines page of the TI-ONE console, and click **New**.

2.2 In the storage configuration section, find CFS and click **Add**.

2.3 Set the CFS mount path:

2.3.1 Source path: path in CFS.

2.3.2 Container Mount Path: the usage path within the dev machine container. The default path is `/home/tione/notebook`. To prevent system conflicts, avoid using critical system directories such as `/bin`, `/etc`, and `/usr` as mount paths. The path must start with a forward slash (`/`).

2.3.3 Note: If the source path does not exist, the platform prompts you to create it. You need to click **Create** to complete the directory creation.

2.4 Complete other configurations for the dev machine (image selection, specification settings, and so on) and submit the creation request.

3. Verify the mount result:

3.1 Start the dev machine.

3.2 Two WebIDEs are provided by default. Select one based on your personal preference. We use Visual Studio Code (VS Code) as an example:

3.3 View mount information in the dev machine:

`172.17.17.247@tcp:/d9fbf2a6/cfs 60T 53T 7.1T 89% /home/tione/notebook`, which means `cfs(172.17.17.247):/demo/` is mounted to the `/home/tione/notebook/` directory inside the container.

4. Directory interpretation:

File System	Mount Point	Detailed Explanation and User Operation Guide
overlay	/	Container root file system. It uses the Overlay2 driver and consists of a read-only base image layer and a writable container layer. Critical area: All data written directly to the root directory occupies this space, which may lead to disk exhaustion.
tmpfs	/dev	In-memory temporary file system that stores device files. It is automatically managed by the system, requiring no user action.
tmpfs	/sys/fs/cgroup	In-memory temporary file system that stores cgroup resource control information. It is automatically managed by the system.
tmpfs	/etc/linux_user_info	Special mount on TI-ONE. It is used to transmit user identity information to the container. A size of 1000T+ represents unlimited capacity. You should not store data here.

tmpfs	/dev/shm	System shared memory space, which is used for inter-process communication (IPC). It can store temporary data, but the data is lost upon restart.
/dev/mapper/d ata-lv_0	/run/ti	TI-ONE data volume. It may be used to store internal platform services, logs, or intermediate data. You should not store personal data in this path.
/dev/vda2	/usr/bin/tiops ctl	System disk partition of the host. It is mounted to specific paths within the container. It may contain TI-ONE platform management tools and binaries. This is a system-critical area where write operations are strictly prohibited.
172.17.17.247 @tcp:d9fbf2a 6/cfs	/home/tione/ notebook	CFS storage path (NFS protocol). <div style="border: 1px solid #00a0e3; padding: 10px; background-color: #e6f2ff;"> <p>Note:</p> <ul style="list-style-type: none"> This is the only secure data storage location. All code, data, model files, and logs must be stored in this directory to ensure persistence and sharing across multiple instances. </div>
tmpfs	/proc/acpi, /proc/scsi, /sys/firmwar e	Kernel information interface file system. It is part of container security isolation, requiring no user action.

Data Writing Requirements for Using Dev Machines

- Clearly define the types of data to be stored (such as data sets, model files, training logs, and temporary results).
- Example: Upload data sets to the `/home/tione/notebook/dataset/` directory, store model training outputs in the `/home/tione/notebook/checkpoint/` directory, and store code in the `/home/tione/notebook/code/` directory.
- Do not write data to default paths on the system disk (such as `/`, `/home`, `/root`, `/dev`, and `/etc`).

5. Data Management and Maintenance Recommendations After CFS Mounting

Data Classification and Storage Specifications

1. It is recommended to create corresponding directories in the CFS instance based on data types (such as data sets, model files, logs, and temporary files). Example:
 - 1.1 Assume that `cfs:/` is mounted to the container path `/home/tione/notebook/`:
 - `/home/tione/notebook/dataset`: stores various business data sets.
 - `/home/tione/notebook/model`: stores trained model files.
 - `/home/tione/notebook/logs`: stores service logs and training logs.
 - `/home/tione/notebook/temp`: stores temporarily generated data and is cleaned up periodically.
 - 1.2 This is equivalent to creating `cfs:/dataset/`, `cfs:/model/`, `cfs:/logs/`, and `cfs:/temp/`.
2. You can also create corresponding directories based on users, groups, and departments. Example:
 - 2.1 Assume that `cfs:/` is mounted to the container path `/home/tione/notebook/`:
 - `/home/tione/notebook/user1`: stores the data of user1.
 - `/home/tione/notebook/user2`: stores the data of user2.
 - `/home/tione/notebook/user3`: stores the data of user3.
 - 2.2 This is equivalent to creating `cfs:/user1/`, `cfs:/user2/`, and `cfs:/user3/`.
3. Avoid writing data directly to the CFS root directory to prevent disorganized files that are difficult to manage.
4. If you need to control CFS read/write permissions or have more complex management requirements, you may also consider using the data source management feature. For more details, see [Data Source Management](#).

CFS Storage Capacity Monitoring and Scale-Out

1. For more information about how to view the capacity usage of a CFS instance in the Tencent Cloud console, see CFS related documents.
2. Set capacity alarming thresholds to scale out the capacity promptly.

6. System Disk Monitoring and Alarm Configuration

Monitoring Configuration Steps

The following example shows how to configure disk monitoring alarms for a CVM instance.

1. Find the target CVM instance, expand the monitoring dashboard, find Disk Monitoring, and click the **alarm configuration icon**.
2. Configure an alarm rule:
 - 2.1 Policy Type: Select CVM/Disk Partition Monitoring.
 - 2.2 Trigger Condition: If the disk utilization exceeds 85% at a granularity of 1 minute for 5 consecutive data points, an alarm will be triggered every 6 hours. (Note: This alarm

condition is for example only. You should configure trigger conditions based on your actual usage.)

2.3 After the configuration is completed, click **Next: Configure Alarm Notification**.

3. Configure a notification template:

3.1 If a template already exists, you can directly add the existing template. If not, you can create a notification template.

3.2 Configure recipients, notification frequency, notification period, and receiving channels.

3.3 Configure custom callback interfaces as needed.

3.4 Click **Complete**.

After the configuration is completed, disk utilization alarms for the CVM instance are sent to the specified users through the specified channels.

7. Common Issues and Troubleshooting Solutions

CFS Mounting Failures

1. **Issue symptom 1: When a dev machine is created, the CFS mount option cannot be selected.**

1.1 Possible causes:

- The CFS instance and the CVM instance are not in the same region, and the network is not connected.
- Your account does not have the permissions to access CFS.
- The CFS instance does not exist.

1.2 Troubleshooting steps and solutions:

It is recommended to contact the TI-ONE or CFS support team for troubleshooting.

2. **Issue symptom 2: You are unable to read or write data after the CFS instance is mounted.**

2.1 Possible causes:

- CFS storage exceptions.
- Insufficient user permissions.

2.2 Troubleshooting steps and solutions:

It is recommended to contact the CFS support team for troubleshooting.

Data Writing Performance Issues

Issue symptom: After the CFS instance is mounted, the data read/write speed is slow, affecting service/task efficiency.

It is recommended to contact the CFS support team for troubleshooting.

Insufficient System Disk Space After Mounting

Issue symptom: After the CFS instance is mounted, the system disk of the CVM instance is still full.

- Possible causes:

Data is written to paths without persistent storage mounted inside the container. Example:

- Install software packages in the root directory / (for example, pip install).
- Download data to directories such as /tmp or /root.
- Download models from ModelScope without specifying a local folder path or without specifying the local path to persistent storage.

- ModelScope official documentation:

模型下载默认存放地址

无论是使用命令行还是ModelScope SDK，模型会下载到 `~/.cache/modelscope/hub` 默认路径下。如果需要修改 cache 目录，可以手动设置环境变量：`MODELSCOPE_CACHE`，完成设置后，模型将下载到该环境变量指定的目录中。

When you train models in a dev machine, download data from Hugging Face to the local cache.

Cache management

When you download a dataset from Hugging Face, the data are stored locally on your computer. Files from Hugging Face are stored as usual in the `huggingface_hub` cache, which is at `~/.cache/huggingface/hub` by default. See the [Hub cache documentation](#) for more details and how to change its location.

ⓘ Note:

The above operations add data to the container writable layer, resulting in reduced available space in the overlay and consuming system disk storage space.

8. Summary and Must-Knows

Core Operation Summary

1. It is recommended to prioritize configuring CFS mounting when you create TI-ONE services (dev machines, online services, and containerized training).
2. Data writing must be directed to the CFS mount path. Writing to critical directories on the system disk is strictly prohibited.
3. Regularly monitor the capacity and system disk usage of the CFS instance, and perform data backup and maintenance.

Additional Information About Data Disk Scenarios

1. To prevent users from directly writing data to the system disk of a CVM instance, which may cause the instance to become abnormal and unavailable, it is recommended to mount a cloud disk as a data disk for the instance.

2. By mounting a cloud disk as a data disk for a CVM instance, you can migrate container data storage from the system disk to the data disk, facilitating disk management and capacity expansion. When a dev machine created in this scenario writes data to non-persistent storage, by default, it writes to the data disk instead of the system disk.

reading.

Thank you for

Guide on Running DeepSpeed Distributed Training with Task-based Modeling

Last updated: 2026-03-12 10:33:35

Overview

This document takes [Llama-3.2-1B-Instruct](#) as an example to introduce how to use DeepSpeed for distributed training on Tencent Cloud TI-ONE Platform (TI-ONE).

This document mainly uses two modules of TI-ONE: **Dev Machines** and **Task-based Modeling**. The **Dev Machines** module is used to download models and training frameworks, and configure related scripts. The **Task-based Modeling** module is used to initiate training tasks. The training framework of this guide adopts the open-source LLaMA-Factory, which is a simple, easy-to-use, and efficient platform for training and fine-tuning LLMs. It provides examples of DeepSpeed configuration files for different stages (such as ZeRO-2/3), and comes preset with several data sets to support an out-of-the-box training process.

Note that DeepSpeed supports configuring multi-node computing resources using hostfiles in the OpenMPI format. The preset image for Task-based Modeling, `tilearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu`, supports MPI communication mode. Therefore, this document takes this image as an example to introduce how to perform distributed training with DeepSpeed.

Prerequisites

Application for Cloud File Storage (CFS) or Data Accelerator Goose FileSystem extreme (GooseFSx): In the process of fine-tuning custom LLMs, the storage used for model files and training code is CFS or GooseFSx. Therefore, it is required to first apply for CFS or GooseFSx. For details, see [CFS > Creating a File System and a Mount Point](#).

Operation Steps

Material Preparations

Note: Currently, TI-ONE does not support direct data upload through the console. To resolve this issue, you need to create an Ops dev machine that mounts CFS or GooseFSx instances and use the dev machine service to upload or download LLMs, training code, and other files.

Choose **Training Workshop > Dev Machines** and click **Create** to create a dev machine for debugging training code.

- **Image:** Select any built-in image, as this dev machine is only used for downloading model files and training code.
- **CVM Instance Source:** Select purchasing pay-as-you-go CVM instances from TI-ONE or select from CVM instances.
- **Billing Mode:** Select either pay-as-you-go or yearly/monthly subscription billing mode. For billing rules supported by TI-ONE, see [Billing Overview](#).
- **Storage Path Configuration:** Select CFS or GooseFSx. The name should be the CFS or GooseFSx instance you applied for and configured in the Prerequisites section. The path is the root directory / by default and is used to specify the storage location for your custom LLM.
- **Other Settings:** Not required by default.

Model Files

After successful creation, choose **Dev Machines** > Python3(ipynkernel) to download the required model through the script. You can search for the required LLM in [ModelScope](#) or [Hugging Face](#), download models through Python scripts in ModelScope, and save them to CFS or GooseFSx. This document takes the Llama-3.2-1B-Instruct model as an example, and the code for download is as follows:

```
!pip install modelscope

from modelscope import snapshot_download
#LLM-Research/Llama-3.2-1B-Instruct is the name of the model to be
downloaded, cache_dir is the address where the downloaded model is saved,
and ./ indicates saving the downloaded model in the root directory of the
mount directory of CFS.
model_dir = snapshot_download('LLM-Research/Llama-3.2-1B-Instruct',
cache_dir='./')
```

Note: The specified `cache_dir` address for the downloaded model is the location in Task-based Modeling, namely, the mounted dev machine's path + `cache_dir`. For example, if the mounted dev machine's path is `/dir1` and `cache_dir` is `/dir2`, then the file location in CFS or GooseFSx will be `/dir1/dir2`.

Copy the above download script, replace relevant content with the model to be downloaded, paste the modified script into the new `.ipynb` file, and click **Run** to start downloading the model.

```

root@nb-127046966108 x | start.sh x | pyproject.toml x | prepare.sh x | ds_0_config.json x | train.py x | 未命名.ipynb
[3]: pip install modelscope
from modelscope import snapshot_download
#open/ OwenZ-0.50-Instruct为模型名称，cache_dir为本地缓存地址，这里'/'表示将下载模型保存在CFS挂载目录的根目录中
model_dir = snapshot_download('LLM-Research/Llama-3.2-1B-Instruct', cache_dir='/')

Looking in indexes: https://mirrors.tencentyun.com/pypi/simple
Requirement already satisfied: modelscope in /usr/local/lib/python3.10/dist-packages (1.23.2)
Requirement already satisfied: requests==2.25 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.32.3)
Requirement already satisfied: tqdm==4.64.0 in /usr/local/lib/python3.10/dist-packages (from modelscope) (4.66.2)
Requirement already satisfied: urllib3==1.26 in /usr/local/lib/python3.10/dist-packages (from modelscope) (2.2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests==2.25->modelscope) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.25->modelscope) (3.6)
Requirement already satisfied: certifi==2021.4.11 in /usr/local/lib/python3.10/dist-packages (from requests==2.25->modelscope) (2024.2.2)
DEPRECATION: mlx-tools-5.2.0 has a non-standard version number. pip 24.1 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of mlx-tools or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at https://github.com/pypa/pip/issues/12063 class="ansi-yellow-fg">
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv class="ansi-yellow-fg">

[notice] A new release of pip is available: 24.0 -> 25.0.1
[notice] To update, run: python -m pip install --upgrade pip
Downloading Model to directory: ./LLM-Research/Llama-3.2-1B-Instruct
2025-03-10 18:40:02,212 - modelscope - INFO - Got 13 files, start to download ...
Processing 13 items: 0% | 0.00/13.0 [00:00<, 71t/s]
Downloading [LICENSE.txt]: 0% | 0.00/7.53k [00:00<, 7B/s]
Downloading [original/consolidated.00.pth]: 0% | 0.00/2.30G [00:00<, 7B/s]
Downloading [model.safetensors]: 0% | 0.00/2.30G [00:00<, 7B/s]
Downloading [original/params.json]: 0% | 0.00/220 [00:00<, 7B/s]
Downloading [generation_config.json]: 0% | 0.00/189 [00:00<, 7B/s]
Downloading [config.json]: 0% | 0.00/877 [00:00<, 7B/s]
Downloading [configuration.json]: 0% | 0.00/2.00 [00:00<, 7B/s]
Downloading [README.md]: 0% | 0.00/40.8k [00:00<, 7B/s]
Downloading [special_tokens_map.json]: 0% | 0.00/296 [00:00<, 7B/s]
Downloading [tokenizer.json]: 0% | 0.00/8.60M [00:00<, 7B/s]
Downloading [original/tokenizer.model]: 0% | 0.00/2.00M [00:00<, 7B/s]
Downloading [tokenizer_config.json]: 0% | 0.00/53.2k [00:00<, 7B/s]
Downloading [tokenizer.model]: 0% | 0.00/5.80k [00:00<, 7B/s]
2025-03-10 18:45:49,872 - modelscope - INFO - Download model 'LLM-Research/Llama-3.2-1B-Instruct' successfully.
2025-03-10 18:45:49,875 - modelscope - INFO - Creating symbolic link [./LLM-Research/Llama-3.2-1B-Instruct].
[1]:

```

Training Code

Downloading LLaMA-Factory Code

Directly download the code to CFS via git clone by clicking Terminal to enter the following command.

```
git clone https://github.com/hiyouga/LLaMA-Factory.git
```

Alternatively, download the open-source code locally first as [main.zip](#), and then upload the LLaMA-Factory source code to the dev machine and unzip it.

If you enable SSH Connection during creation, you can use SCP to upload local materials to the dev machine.

```
scp -r -P <port> Llama-3.2-1B-Instruct
root@host:/home/tione/notebook/workspace
```

Tips: If your file is downloaded via Git, you can try to delete the .git directory under the directory to reduce the number of files to be uploaded and improve the upload speed.

Creating a Dependency Installation Script

Using DeepSpeed requires a specific version of the transformer. Therefore, pip is directly used to install the latest DeepSpeed and the corresponding transformer.

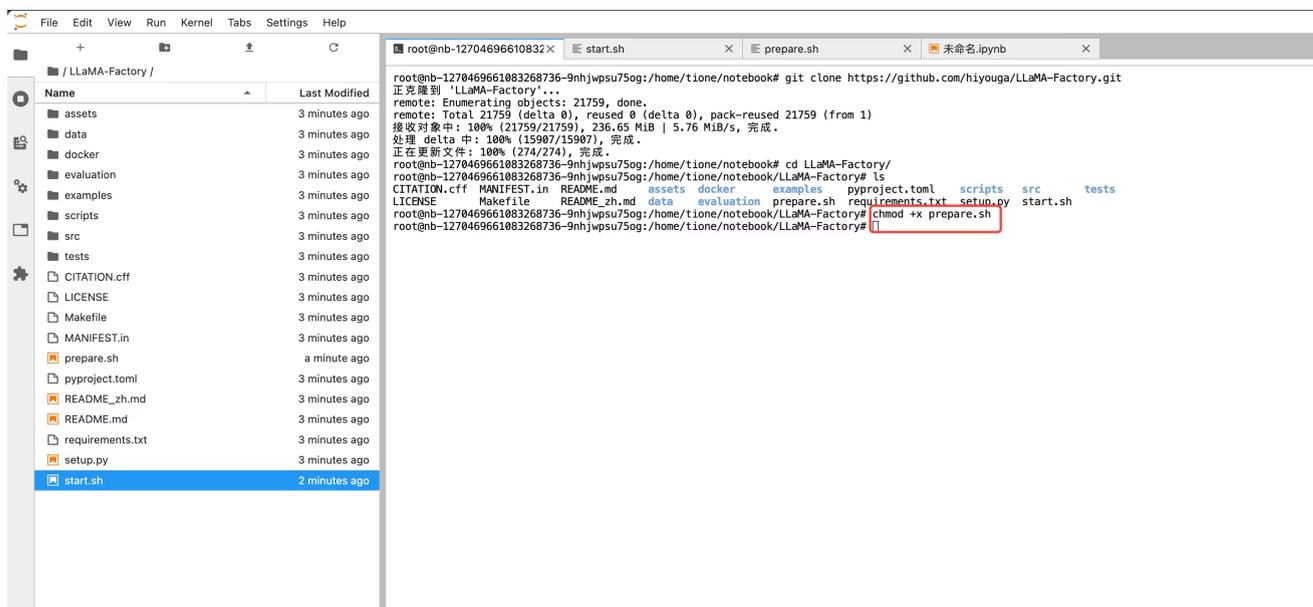
In the multi-machine scenario, it is required to install or update dependencies on each machine, which can be done through mpirun. mpirun is a command line interface of OpenMPI, which provides a simple method to start applications in parallel.

Create a `prepare.sh` file in the LLaMA-Factory directory, which is used as a dependency installation script with the following content:

```
#!/bin/bash

pip3 install -U accelerate==1.0.1 transformers deepspeed -i
https://mirrors.tencent.com/pypi/simple
```

Grant execution permissions via `chmod` in the terminal.



Creating a Startup Script

Create a `start.sh` file in the LLaMA-Factory directory, which is used as a training startup script with the following content:

```
#!/bin/bash

# Install dependencies.
# pip3 install -U accelerate==1.0.1 transformers deepspeed -i
https://mirrors.tencent.com/pypi/simple

# Start DeepSpeed distributed training in the MPI mode.
# Visit https://llamafactory.readthedocs.io/zh-cn/latest/advanced/distributed.html#id18.
deepspeed --num_gpus $GPU_NUM_PER_NODE --num_nodes 2 --hostfile
/etc/mpi/hostfile src/train.py \
--deepspeed examples/deepspeed/ds_z0_config.json \
--stage sft \
```

```
--model_name_or_path /opt/ml/pretrain_model \  
--do_train \  
--dataset identity \  
--template llama3 \  
--finetuning_type full \  
--output_dir /llm-private/output \  
--overwrite_cache \  
--per_device_train_batch_size 1 \  
--gradient_accumulation_steps 8 \  
--lr_scheduler_type cosine \  
--logging_steps 10 \  
--save_steps 500 \  
--learning_rate 1e-4 \  
--num_train_epochs 10 \  
--plot_loss \  
--bf16
```

Specifically, the value of `num_nodes` can be adjusted based on the actual node number. `$GPU_NUM_PER_NODE` is an environment variable injected for Task-based Modeling, indicating the number of GPU cards for each node.

`/etc/mpi/hostfile` will be created through Task-based Modeling.

`model_name_or_path` refers to the model directory and needs to be mounted accordingly when training is started.

The value of `template` should be modified according to the corresponding model.

Launching a Distributed Fine-Tuning Task in Task-based Modeling

Creating a Training Task

Choose **Training Workshop > Task-based Modeling** and click **Create a Task**. Set the training image to the following built-in image, set the training mode to MPI, and select the resource configurations as needed. In this guide, a 2-node distributed training will be performed.

- **Training Image:** Select built-in image/PyTorch/tilearn-llm0.9-torch2.3-py3.10-cuda12.4-gpu. This image has been configured with the runtime environment for LLM training by default.
- **CVM Instance Source:** See the configuration of [Dev Machines > Computing Resources](#).
- **CVM Instance Specifications:** Select training resources appropriately. Use 1 GPU card per node (allocated as 0.5 GPU cards but functioning as a full GPU card) and use 2 nodes.
- **Storage Path Configuration:** The **training code**, **base model**, and **training output path** need to be mounted. The container mount path corresponds to the relevant parameters in the training configuration file mentioned above. You can also place the training code, base

model, and training output path in the same large directory, mount the root directory to the container, and modify the startup command to run the `cd` command to access LLaMA-Factory.

- **Startup Command:** Run the dependency installation script via `mpirun`, and then start training.

```
cd /opt/ml/code
mpirun --allow-run-as-root -hostfile /etc/mpi/hostfile --map-by
ppr:1:node /opt/ml/code/prepare.sh
bash start.sh
```

Viewing Training Status

After the task is started, you can click **Logs** to view training logs and other information such as dependency installation and training.

After training is completed, the final checkpoint will be saved to the mounted output directory `output`, which is used for subsequent deployment of the fine-tuned model.

TensorBoard

You can choose **Task List** and click **TensorBoard** in the **Operation** column to configure a TensorBoard task. The path `/project/output` is the output path for model training in CFS. After confirmation, you can view the status of the training process, such as loss and gradient.

