Easy Model Deployer

None

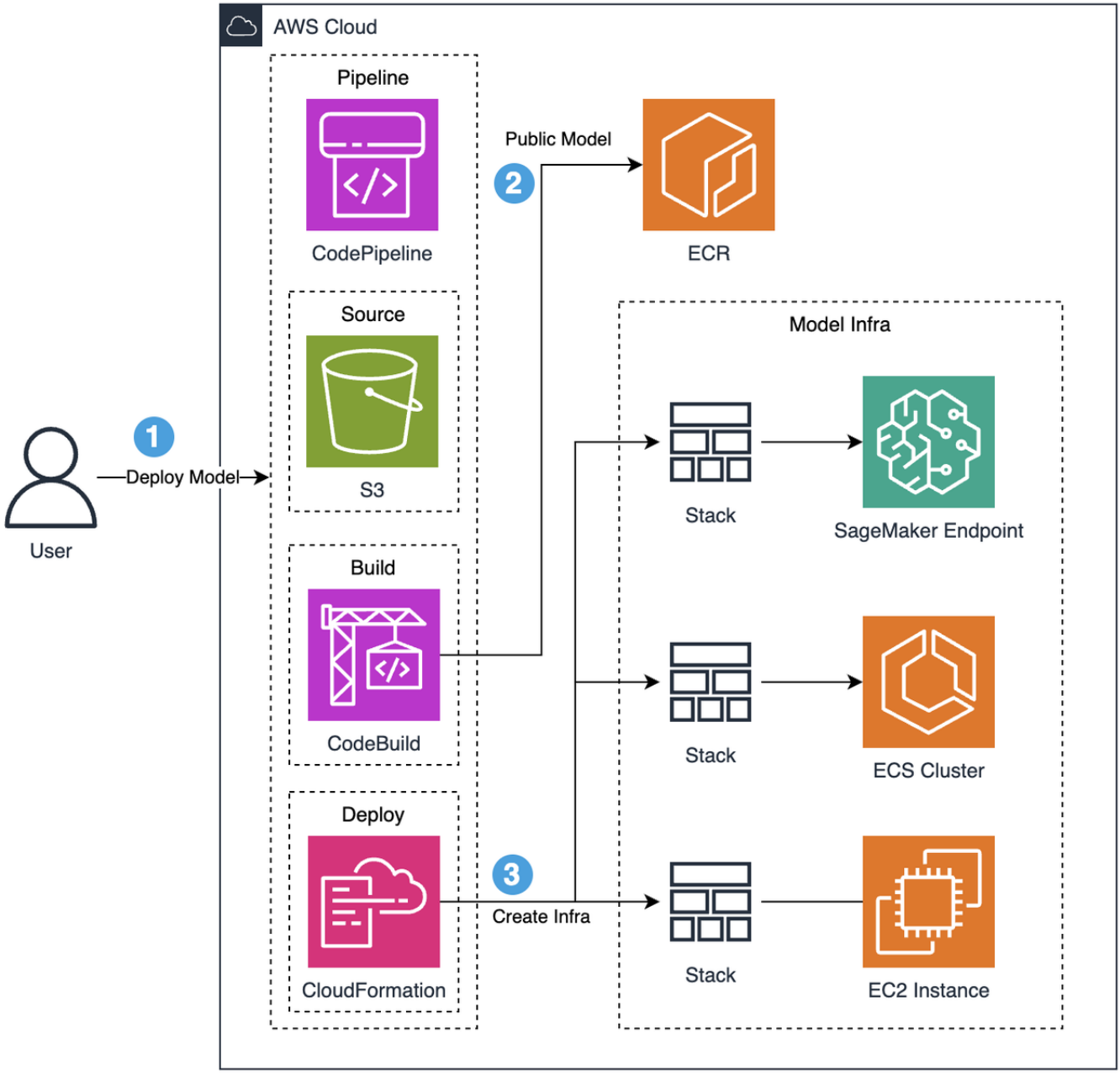
None

None

Table of contents

1. Architecture

Deploy models to the cloud with EMD will use the following components in Amazon Web Services:



1. User/Client initiates model deployment task, triggering pipeline to start model building.
2. AWS CodeBuild constructs the large model using predefined configuration and publishes it to Amazon ECR.
3. AWS CloudFormation creates a model infrastructure stack based on user selection and deploys the model from ECR to AWS services (Amazon SageMaker, EC2, ECS).

2. Installation

2.1 Installation Guide

2.1.1 Prerequisites

- Python 3.9 or higher
- pip (Python package installer)

2.1.2 Setting up the Environment

1. Create a virtual environment:

```
python -m venv emd-env
```

2. Activate the virtual environment:

```
source emd-env/bin/activate
```

3. Install the required packages:

```
pip install https://github.com/aws-samples/easy-model-deployer/releases/download/main/emd-0.6.0-py3-none-any.whl
```

2.2 Deployment parameters

2.2.1 --force-update-env-stack

No additional `emd bootstrap` required for deployment. Because of other commands, status/destroy etc. require pre-bootstrapping. Therefore, it is recommended to run `emd bootstrap` separately after each upgrade.

2.2.2 --extra-params

Extra parameters passed to the model deployment. extra-params should be a Json object of dictionary format as follows:

```
{

  "model_params": {
  },
  "service_params":{
  },
  "instance_params":{
  },
  "engine_params":{
      "cli_args": "<command line arguments of current engine>",
      "api_key":"<api key>"
  },
  "framework_params":{
      "uvicorn_log_level":"info",
      "limit_concurrency":200
  }
}
```

To learn some practice examples, please refer to the Best Deployment Practices.

2.3 Local deployment on the ec2 instance

This is suitable for deploying models using local GPU resources.

2.3.1 Pre-requisites

Start and connect to EC2 instance

It is recommended to launch the instance using the AMI "**Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.6 (Ubuntu 22.04)**".

2.3.2 Deploy model using EMD

```
emd deploy --allow-local-deploy
```

There some EMD configuration sample settings for model deployment in the following two sections: Non-reasoning Model deployment configuration and Reasoning Model deployment configuration. Wait for the model deployment to complete.

Non-reasoning Model deployment configuration

QWEN2.5-72B-INSTRUCT-AWQ

```
? Select the model series: qwen2.5
? Select the model name: Qwen2.5-72B-Instruct-AWQ
? Select the service for deployment: Local
? input the local gpu ids to deploy the model (e.g. 0,1,2): 0,1,2,3
? Select the inference engine to use: tgi
? (Optional) Additional deployment parameters (JSON string or local file path), you can skip by pressing Enter: {"engine_params":{"api_key":"<YOUR_API_KEY>",
"default_cli_args": "--max-total-tokens 30000 --max-concurrent-requests 30"}}
```

LLAMA-3.3-70B-INSTRUCT-AWQ

```
? Select the model series: llama
? Select the model name: llama-3.3-70b-instruct-awq
? Select the service for deployment: Local
? input the local gpu ids to deploy the model (e.g. 0,1,2): 0,1,2,3
engine type: tgi
framework type: fastapi
? (Optional) Additional deployment parameters (JSON string or local file path), you can skip by pressing Enter: {"engine_params":{"api_key":"<YOUR_API_KEY>",
"default_cli_args": "--max-total-tokens 30000 --max-concurrent-requests 30"}}
```

Reasoning Model deployment configuration

DEEPSEEK-R1-DISTILL-QWEN-32B

```
? Select the model series: deepseek reasoning model
? Select the model name: DeepSeek-R1-Distill-Qwen-32B
? Select the service for deployment: Local
? input the local gpu ids to deploy the model (e.g. 0,1,2): 0,1,2,3
engine type: vllm
framework type: fastapi
? (Optional) Additional deployment parameters (JSON string or local file path), you can skip by pressing Enter: {"engine_params":{"api_key":"<YOUR_API_KEY>",
"default_cli_args": "--enable-reasoning --reasoning-parser deepseek_r1 --max_model_len 16000 --disable-log-stats --chat-template emd/models/chat_templates/
deepseek_r1_distill.jinja --max_num_seq 20 --gpu_memory_utilization 0.9"}}
```

DEEPSEEK-R1-DISTILL-LLAMA-70B-AWQ

```
? Select the model series: deepseek reasoning model
? Select the model name: deepseek-r1-distill-llama-70b-awq
? Select the service for deployment: Local
? input the local gpu ids to deploy the model (e.g. 0,1,2): 0,1,2,3
? Select the inference engine to use: tgi
framework type: fastapi
? (Optional) Additional deployment parameters (JSON string or local file path), you can skip by pressing Enter: {"engine_params":{"api_key":"<YOUR_API_KEY>",
"default_cli_args": "--max-total-tokens 30000 --max-concurrent-requests 30"}}
```

2.4 Examples

3. Best Deployment Practices

This document provides examples of best practices for deploying models using EMD for various use cases.

3.1 Deploying to Specific GPU Types

Choosing the right GPU type is critical for optimal performance and cost-efficiency. Use the `--instance-type` parameter to specify the GPU instance.

3.1.1 Example: Deploying Qwen2.5-7B on g5.2xlarge

```
emd deploy --model-id Qwen2.5-7B-Instruct --instance-type g5.2xlarge --engine-type vllm --service-type sagemaker
```

3.2 Achieving Longer Context Windows

To enable longer context windows, use the `--extra-params` option with engine-specific parameters.

3.2.1 Example: Deploying model with 16k context window

```
emd deploy --model-id Qwen2.5-7B-Instruct --instance-type g5.4xlarge --engine-type vllm --service-type sagemaker --extra-params '{
  "engine_params": {
    "vllm_cli_args": "--max_model_len 16000 --max_num_seqs 4"
  }
}'
```

3.2.2 Example: Deploying model on G4dn instance

```
emd deploy --model-id Qwen2.5-14B-Instruct-AWQ --instance-type g4dn.2xlarge --engine-type vllm --service-type sagemaker --extra-params '{
  "engine_params": {
    "environment_variables": "export VLLM_ATTENTION_BACKEND=XFORMERS && export PYTORCH_CUDA_ALLOC_CONF=expandable_segments:True",
    "default_cli_args": " --chat-template emd/models/chat_templates/qwen_2d5_add_prefill_chat_template.jinja --max_model_len 12000 --max_num_seqs 10  --
gpu_memory_utilization 0.95 --disable-log-stats --enable-auto-tool-choice --tool-call-parser hermes"
  }
}'
```

3.3 Common Troubleshooting

If your deployment fails due to out-of-memory issues, try:

- Using a larger instance type
- Reducing max_model_len and max_num_seqs in the engine parameters
- Setting a lower gpu_memory_utilization value (e.g., 0.8 instead of the default)

4. Usse EMD client to invoke deployed models

```
emd invoke MODEL_ID MODEL_TAG (Optional)
```

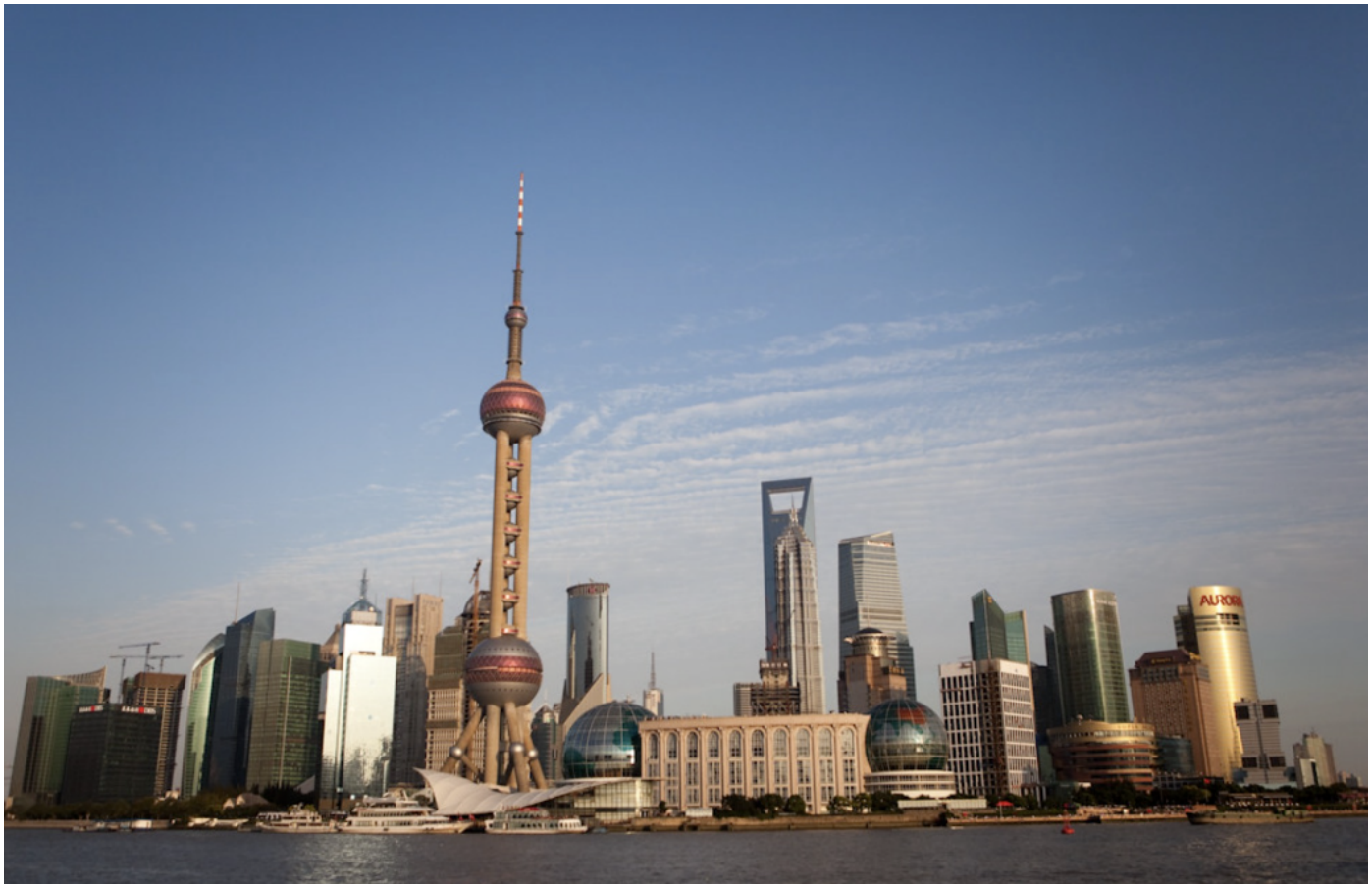
4.1 LLM models

```
emd invoke  DeepSeek-R1-Distill-Qwen-7B
...
Invoking model DeepSeek-R1-Distill-Qwen-7B with tag dev
Write a prompt, press Enter to generate a response (Ctrl+C to abort),
User: how to solve the problem of making more profit
Assistant:<think>

Okay, so I need to figure out how to make more profit. Profit is basically the money left after subtracting costs from revenue, right? So, increasing profit
means either making more money from sales or reducing the
expenses. Let me think about how I can approach this.
...
```

4.2 VLM models

1. upload image to a s3 path



```
aws s3 cp image.jpg s3://your-bucket/image.jpg
```

2. invoke the model

```
emd invoke  Qwen2-VL-7B-Instruct
...
Invoking model Qwen2-VL-7B-Instruct with tag dev
Enter image path(local or s3 file): s3://your-bucket/image.jpg
```

```
Enter prompt: What's in this image?
...
```

4.2.1 Video(Txt2Video) models

1. input prompt for video generation

```
emd invoke txt2video-LTX
...
Invoking model txt2video-LTX with tag dev
Write a prompt, press Enter to generate a response (Ctrl+C to abort),
User: Two police officers in dark blue uniforms and matching hats enter a dimly lit room through a doorway on the left side of the frame. The first officer,
with short brown hair and a mustache, steps inside first, followed by his partner, who has a shaved head and a goatee. Both officers have serious expressions
and maintain a steady pace as they move deeper into the room. The camera remains stationary, capturing them from a slightly low angle as they enter. The room
has exposed brick walls and a corrugated metal ceiling, with a barred window visible in the background. The lighting is low-key, casting shadows on the
officers' faces and emphasizing the grim atmosphere. The scene appears to be from a film or television show.
...
```

2. download generated video from **output_path**

4.3 Embedding models

```
emd invoke bge-base-en-v1.5
...
Invoking model bge-base-en-v1.5 with tag dev
Enter the sentence: hello
...
```

4.4 Rerank models

```
emd invoke bge-reranker-v2-m3
...
Enter the text_a (string): What is the capital of France?
Enter the text_b (string): The capital of France is Paris.
...
```

4.5 ASR models(whisper)

1. upload audio to a s3 path

```
aws s3 cp xx.wav s3://your-bucket/xx.wav
```

2. invoke the model

```
emd invoke whisper
...
Enter the s3 path to the audio file: s3://your-bucket/xx.wav
Enter model [large-v3-turbo/large-v3]: large-v3-turbo
...
```

5. Usse Langchain interface to invoke deployed models

5.1 LLM models

```
from emd.integrations.langchain_clients import SageMakerVllmChatModel
from langchain_core.output_parsers import StrOutputParser
from langchain_core.messages import HumanMessage,AIMessage,SystemMessage
from langchain.tools.base import StructuredTool
from langchain_core.utils.function_calling import (
    convert_to_openai_function,
    convert_to_openai_tool
)
chat_model = SageMakerVllmChatModel(
    model_id="Qwen2.5-7B-Instruct",
    model_kwargs={
        "temperature":0.5,
    }
)
chain = chat_model | StrOutputParser()
messages = [
        HumanMessage(content="9.11 9.9        "),
    ]
print(chain.invoke(messages))
```

5.2 VLM models

1. upload image to a s3 path

```
aws s3 cp image.jpg s3://your-bucket/image.jpg
```

2. invoke the model

```
emd invoke  Qwen2-VL-7B-Instruct
...
Invoking model Qwen2-VL-7B-Instruct with tag dev
Enter image path(local or s3 file): s3://your-bucket/image.jpg
Enter prompt: What's in this image?
...
```

5.2.1 Video(Txt2Video) models

Not supported

5.3 Embedding models

```
import time
from emd.integrations.langchain_clients import SageMakerVllmEmbeddings
from emd.integrations.langchain_clients import SageMakerVllmRerank
embedding_model = SageMakerVllmEmbeddings(
    model_id="bge-m3",
)
text = 'The giant panda (Ailuropoda melanoleuca), sometimes called a panda bear or simply panda, is a bear species endemic to China.'
t0 = time.time()
r1 = embedding_model.embed_query(text)
t1 = time.time()
embedding_model.embed_documents([text]*1000)
t2 = time.time()
print(f"embed_query: {t1-t0}")
print(f"embed_documents: {t2-t1}")
```

5.4 Rerank models

```
import time
from emd.integrations.langchain_clients import SageMakerVllmRerank
docs = ["hi",'The giant panda (Ailuropoda melanoleuca), sometimes called a panda bear or simply panda, is a bear species endemic to China.']
query = 'what is panda?'
rerank_model = SageMakerVllmRerank(
    model_id="bge-reranker-v2-m3"
```

```
)
print(rerank_model.rerank(query=query,documents=docs))
```

6. Test OpenAI compatible interface

6.0.1 Sample Code

```python
import openai
# Change the api_key here to the parameter you passed in via extra-parameter
api_key = "your_openai_api_key"
def chat_with_openai_stream(prompt):
    client = openai.OpenAI(
        api_key=api_key,
        base_url="http://ec2-54-189-171-204.us-west-2.compute.amazonaws.com:8080/v1"
    )
    response = client.chat.completions.create(
        model="Qwen2.5-72B-Instruct-AWQ",
        # model="Qwen2.5-1.5B-Instruct",
        messages=[
        {"role": "user", "content": prompt}
        ],
        stream=True,
        temperature=0.6
    )
    print("AI: ", end="", flush=True)
    print(response)
    for chunk in response:
        content = chunk.choices[0].delta.content
        think = getattr(chunk.choices[0].delta,"reasoning_content",None)
        if think is not None:
            print(think,end="",flush=True)
        else:
            print(content, end="", flush=True)
    print("\n")

def chat_with_openai(prompt):
    client = openai.OpenAI(
        api_key=api_key,
        base_url="http://127.0.0.1:9000/v1"
    )
    response = client.chat.completions.create(
        model="DeepSeek-R1-Distill-Qwen-1.5B",
        messages=[{"role": "system", "content": "You are a helpful assistant."},
                  {"role": "user", "content": prompt}],
        stream=False
    )
    print(response)

# Test the stream and non-stream interface
chat_with_openai_stream("What is the capital of France?")
chat_with_openai("What is the capital of France?")
```

7. Supported Model

ModelId	ModelSeries	ModelType	Supported Engines	Supported Instances
glm-4-9b-chat	glm4	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
internlm2_5-20b-chat-4bit-awq	internlm2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
internlm2_5-20b-chat	internlm2.5	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
internlm2_5-7b-chat	internlm2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
internlm2_5-7b-chat-4bit	internlm2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
internlm2_5-1_8b-chat	internlm2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
Qwen2.5-7B-Instruct	qwen2.5	llm	vllm,tgi	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
Qwen2.5-72B-Instruct-AWQ	qwen2.5	llm	vllm,tgi	g5.12xlarge,g5.24xlarge,g5.48xlarge,inf2.
Qwen2.5-72B-Instruct-AWQ-inf2	qwen2.5	llm	tgi	inf2.24xlarge
Qwen2.5-72B-Instruct	qwen2.5	llm	vllm	g5.48xlarge
Qwen2.5-72B-Instruct-AWQ-128k	qwen2.5	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
Qwen2.5-32B-Instruct	qwen2.5	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
Qwen2.5-32B-Instruct-inf2	qwen2.5	llm	tgi	inf2.24xlarge
Qwen2.5-0.5B-Instruct	qwen2.5	llm	vllm,tgi	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
Qwen2.5-1.5B-Instruct	qwen2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
Qwen2.5-3B-Instruct	qwen2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
Qwen2.5-14B-Instruct-AWQ	qwen2.5	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
Qwen2.5-14B-Instruct	qwen2.5	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
QwQ-32B-Preview	qwen reasoning model	llm	huggingface,vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
llama-3.3-70b-instruct-awq	llama	llm	tgi	g5.12xlarge,g5.24xlarge,g5.48xlarge
DeepSeek-R1-Distill-Qwen-32B	deepseek reasoning model	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
DeepSeek-R1-Distill-Qwen-14B	deepseek reasoning model	llm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
DeepSeek-R1-Distill-Qwen-7B	deepseek reasoning model	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
DeepSeek-R1-Distill-Qwen-1.5B		llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla

ModeId	ModelSeries	ModelType	Supported Engines	Supported Instances
	deepseek reasoning model			
DeepSeek-R1-Distill-Qwen-1.5B_ollama	deepseek reasoning model	llm	ollama	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
DeepSeek-R1-Distill-Qwen-1.5B-GGUF	deepseek reasoning model	llm	llama.cpp	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
DeepSeek-R1-Distill-Llama-8B	deepseek reasoning model	llm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.16xla
deepseek-r1-distill-llama-70b-awq	deepseek reasoning model	llm	vllm,tgi	g5.12xlarge,g5.24xlarge,g5.48xlarge
deepseek-r1-671b-1.58bit_ollama	deepseek reasoning model	llm	ollama	g5.48xlarge
deepseek-r1-671b-1.58bit_gguf	deepseek reasoning model	llm	llama.cpp	g5.48xlarge
deepseek-v3-UD-IQ1_M_ollama	deepseek v3	llm	ollama	g5.48xlarge
Baichuan-M1-14B-Instruct	baichuan	llm	vllm,huggingface	g5.12xlarge,g5.24xlarge,g5.48xlarge
Qwen2-VL-72B-Instruct-AWQ	qwen2vl	vlm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
QVQ-72B-Preview-AWQ	qwen reasoning model	vlm	vllm	g5.12xlarge,g5.24xlarge,g5.48xlarge
Qwen2-VL-7B-Instruct	qwen2vl	vlm	vllm	g5.2xlarge,g5.4xlarge,g5.8xlarge,g5.12xla
InternVL2_5-78B-AWQ	internvl2.5	vlm	lmdeploy	g5.12xlarge,g5.24xlarge,g5.48xlarge
txt2video-LTX	comfyui	video	comfyui	g5.4xlarge,g5.8xlarge,g6e.2xlarge
whisper	whisper	whisper	huggingface	g5.xlarge,g5.2xlarge,g5.4xlarge,g5.8xlarg
bge-base-en-v1.5	bge	embedding	vllm	g5.xlarge,g5.2xlarge,g5.4xlarge,g5.8xlarg
bge-m3	bge	embedding	vllm	g5.xlarge,g5.2xlarge,g5.4xlarge,g5.8xlarg
bge-reranker-v2-m3	bge	rerank	vllm	g5.xlarge,g5.2xlarge,g5.4xlarge,g5.8xlarg