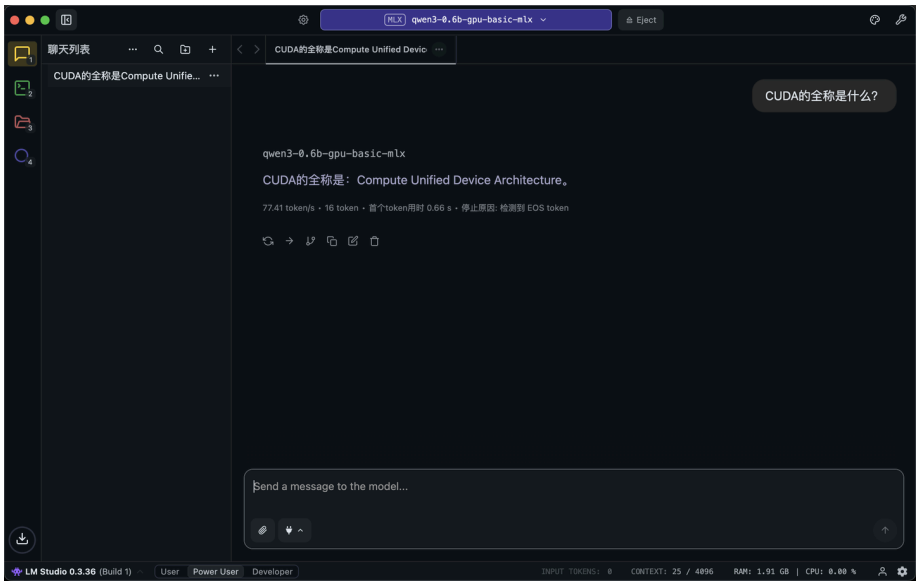


国科大GPU大作业二：训练-推理-部署一条龙服务

1234567 关注他

15 人赞同了该文章 ›

前言：我选择的并行科技赛道，使用Megatron训练框架微调Qwen2.5-0.5B+模型（比Qwen3可以小0.1B hh），vllm推理框架在线推理实测精度和吞吐都非常高，我仅仅是开关一些vllm的参数就可以实现非常高的吞吐（没有任何个人的优化），不过在此之前也想到了vllm作为开源的活跃社区本身优化的已经非常高了（我能力比较菜也写不出更高性能的优化），因为没啥技术结果还打上了比较高的榜，特此撰写此篇文章，一方面帮助完成作业有困难的同学提高作业完成度，另一方面让一些大神可以结合一下我的（调参）方法打出更好的成绩，话不多说，下面是我的排名（时间截取自2025.12.31），其实还有非常大的提升空间的

题型: 基础题 加分题

排名	昵称	校区	赛道	题型	准确率	速度 (tokens/s)
1	[illegible]	[illegible]	***	基础题	0.4292	79970.86
2	[illegible]	[illegible]	***	基础题	0.3851	27901.28
3	[illegible]	[illegible]	***	基础题	0.4022	27174.39
4	123321	玉泉路	***	基础题	0.4144	25869.30
5	[illegible]	[illegible]	***	基础题	0.3945	25037.51

题型: 基础题 加分题

排名	昵称	校区	赛道	题型	准确率	速度 (tokens/s)
1	[illegible]	[illegible]	***	加分题	0.3690	45716.96
2	123321	玉泉路	***	加分题	0.3920	30767.54
3	[illegible]	[illegible]	***	加分题	0.3590	25493.00
4	[illegible]	[illegible]	***	加分题	0.4019	22973.52
5	[illegible]	[illegible]	***	加分题	0.3572	21245.26

一. Megatron模型微调

关于作者

1234567

回答 0 文章 1 关注者 3

关注他 发私信

大家都在搜 换一换

2026新年贺词 347 万 热
央视跨年晚会 346 万 热
你好2026 322 万 热
2026 年国补政策来了 298 万 热
Meta 数十亿美元收购 Manus 277 万
再见2025 266 万
王楚钦拟记大功 252 万 热
2026 年元旦升国旗仪式举行 241 万
土耳其宣布对中国公民免签 237 万 热
元旦跨年朋友圈发什么 232 万

因为之前接触过megatron模型训练，所以在模型微调的时候直接选择了这个训练框架，发现效果非常好，轻轻松松训练一个0.5b的模型并且准确率比较高（由于模型非常小，我提交几次并没有遇到超时问题），官方仓库：Megatron。但是这个训练框架稍微麻烦的地方是，需要先将hf格式的模型转换成框架要求的mcore格式的模型，然后训练完成再转回hf，并且不同模型的参数不一样，官方并没有支持所有的模型（需要自行根据训练的模型填充合适的参数），所以这里我选择另一个阿里的开源框架Pai-Megatron-Patch。优点是底层仍然是megatron，但是上层封装比较好，各种流程一条龙服务，而且自身的Qwen模型支持度比较高。下面介绍该训练框架的安装和训练流程：

2. 安装

这里以并行科技提供的5090机器为例，使用支持的vllm镜像：

```
# 使用一个轻量的Python基础镜像

# for GPU
FROM swr.cn-north-4.myhuaweicloud.com/ddn-k8s/docker.io/vllm/vllm-openai:v0.11.0

# 清除原有的ENTRYPOINT，非常重要，避免CMD被覆盖
ENTRYPOINT []

# 设置工作目录
WORKDIR /app

COPY download_model.py .
RUN python3 download_model.py

# 复制依赖文件并安装依赖
# 这一步单独做可以利用Docker的层缓存机制，如果requirements.txt不变，则不会重新安装
COPY requirements.txt .
RUN pip config set global.index-url https://mirrors.tuna.tsinghua.edu.cn/pypi/web,
RUN pip install --no-cache-dir -r requirements.txt

# 声明容器对外暴露的端口
EXPOSE 8000

COPY . .

# 容器启动时运行的命令
# 使用uvicorn启动FastAPI服务，并监听所有网络接口的8000端口
CMD ["uvicorn", "serve:app", "--host", "0.0.0.0", "--port", "8000"]
```

这里我们先将上面的Dockerfile构建（注意要在这个文件所在目录下执行下面命令）：

```
docker build --network=host -f Dockerfile -t vllm:latest .
# 最后出现Successfully字样表示构建成功
```

然后我们把docker run起来（创建了一个名称为vllm_docker的容器）：

```
docker run --ipc=host --net host --gpus all --privileged=true -p 8000:8000 --name
```

此时会直接运行serve.py的代码，如果出现下面开始监听端口的输出，docker就没问题了。到这里其实也是之后验证代码和模型的操作，之后我们还需要用到

```
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000
```

此时我们开启另一个终端，然后进入到启动的容器中开始环境配置（注意之前的开始监听的这个终端不要关闭，否则可能容器也关闭了）如果使用vscode类的编辑器可以直接进入到容器中操作更方便：

```
docker exec -it vllm_test_docker bash
```

1. 检查环境中torch是否已经存在：

```
pip list | grep torch
# 得到下面的输出就是已经有了torch环境了（如果你也是使用的这个镜像的话肯定是已经存在的）
# torch                             2.8.0+cu128
# torchaudio                        2.8.0+cu128
# torchvision                       0.23.0+cu128
```

然后我们先执行下面命令将其余进程都先kill掉，否则可能影响后续操作（容易出现OOM的情况）

```
nvidia-smi
# 因为之前我们还留了一个终端，如果serve代码里面有推理的相关操作，模型会残留在cuda中，我们
# 0   N/A  N/A              70      C   VLLM::EngineCore                      315
# 比如上面我的这个进程号是70，则执行：
kill -9 70
```

现在重新执行nvidia-smi看是否还有残留进程

然后先把pai-megatron+的仓库clone下来（仓库有一些子模块必须要下载，所以比较大，可以先下着然后进行后续的安装）

我这边测试网速似乎非常慢，可以使用这个网盘链接：
pan.baidu.com/s/1VjCEkH... 提取码：qexz

```
git clone --recurse-submodules https://github.com/alibaba/Pai-Megatron-Patch.git
```

2. 安装一些杂七杂八的库

```
pip install pybind11 regex setuptools importlib-metadata psutil einops pydantic b
```

3. 安装apex+（速度可能有点慢，需要源码编译安装）

```
git clone https://github.com/NVIDIA/apex.git
cd apex
git checkout 25.05
pip install -v --disable-pip-version-check --no-cache-dir --no-build-isolation --
cd ..
```

4. 安装flash-attention+（下载编译好的whl安装）

5. 安装transformer-engine+

```
# 1. 获取 pip 安装的 cuDNN 路径
export CUDNN_PATH=/usr/local/lib/python3.12/dist-packages/nvidia/cudnn

# 2. 将 cuDNN 的 include 目录添加到 CPATH (用于查找头文件)
export CPATH=$CUDNN_PATH/include:$CPATH

# 3. 将 cuDNN 的 lib 目录添加到 LD_LIBRARY_PATH (用于链接库)
export LD_LIBRARY_PATH=$CUDNN_PATH/lib:$LD_LIBRARY_PATH

pip install --no-build-isolation transformer_engine[pytorch]==2.5.0
```

6. 最后再安装杂七杂八的包

```
pip install megatron-energon megatron-energon[av_decode]
pip install pyarrow_hotfix retry pillow accelerate evaluate jieba sentencepiece op
# 此时有可能出现s3fs和fsspec版本不一致的情况，需要重新安装成一致的版本
pip install s3fs==2025.10.0
pip install nvidia-modelopt nvidia-modelopt-core
```

至此安装过程结束

2. 训练微调

大致需要使用上面的项目经历这么几个步骤：数据集转换---模型hf格式转换mcore---训练微调---模型mcore格式转换hf

1. 数据集转换

我们需要先生成下面类似格式的数据集json文件

```
[
    {"instruction": "CUDA的全称是？", "input": "", "output": "CUDA的全称是：Compute
    {"instruction": "什么是数据并行性？", "input": "", "output": "数据并行性是指将计算
    ......
]
```

如果现在容器中还无法直接使用python（似乎只能python3），需要先创建软链接

```
mkdir -p ~/bin
ln -sf "$(which python3)" ~/bin/python
echo 'export PATH="$HOME/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc
```

然后执行命令

```
cd /app/Pai-Megatron-Patch/toolkits/sft_data_preprocessing

bash run_build_idxmap_sft_dataset.sh \
/app/qwen-zh-basic.json \
Qwen3Tokenizer \
```

```
# /app/qwen-zh-basic.json 是你的数据集路径
# /app/qwen-zh-basic 是格式转换后的数据集路径
# /app/Qwen3-0.6B 是实现下载的模型路径
# 可以：modelscope download --model Qwen/Qwen3-0.6B --local_dir /app/Qwen3-0.6B

# 输出如下就执行完成了：
python build_idxmap_sft_dataset.py --input /app/qwen-zh-basic.json --output-prefi
Opening /app/qwen-zh-basic.json
Time to startup: 0.4274251461029053
After pre-tokenizing, the idxmap dataset has 103 samples
0 min 8 sec
```

2. 模型hf格式转换mcore

```
cd /app/Pai-Megatron-Patch/toolkits/distributed_checkpoints_convertor

export KUBERNETES_CONTAINER_RESOURCE_GPU=1
export MODEL_PARALLEL_ARGS="--tensor-model-parallel-size 1 --pipeline-model-paral

bash scripts/qwen3/run_8xH20.sh \
0.6B \
/app/Qwen3-0.6B \
/app/Qwen3-0.6B/mcore-tp1-pp1  \
false \
true \
bf16

# /app/Qwen3-0.6B 是通过modelscope下载的hf模型路径
# /app/Qwen3-0.6B/mcore-tp1-pp1 是将要转换成mcore格式的模型路径
# 其余参数的含义大家自行查找吧

# 输出如下表示成功了（一些警告之类的信息可以不用关注）：
# successfully saved checkpoint from iteration       1 to /app/Qwen3-0.6B/mcore-t
```

3. 模型微调

```
cd /app/Pai-Megatron-Patch/examples/qwen3

bash run_mcore_qwen3.sh  \
dsw  \
0.6B   \
1    \
2 \
1e-5   \
1e-6   \
256  \
256  \
bf16  \
1   \
1  \
1 \
1 \
1 \
true \
true   \
true \
```

```
100  \
/app/qwen-zh-basic_text_document   \
/app/qwen-zh-basic_text_document   \
/app/Qwen3-0.6B/mcore-tp1-pp1  \
500  \
50   \
./output_mcore_qwen3

# /app/qwen-zh-basic_text_document 是转换后数据集的路径（测试集和验证集一样的，没有后缀
# /app/Qwen3-0.6B/mcore-tp1-pp1 是mcore格式的模型路径
# ./output_mcore_qwen3 输出模型checkpoints以及其他日志的路径
# 其他参数对应的含义可以自行查看脚本，500iter其实是有点过拟合的

# 如果跟到这一步理论上应该没啥问题，然后就可以看到500次的训练输出了
# 由于训练轮次比较少，可以在run_mcore_qwen3.sh文件中搜索--eval-interval参数改成100，这
```

4. 模型mcore格式转换回hf格式

这里需要注意，我们生成的checkpoints在上面给定的 `output_mcore_qwen3/checkpoints/finetune...../iter_xxxxxxx/` ，需要把这个iter_xxxxxxx的文件夹转移到一个特定文件夹下面（否则要么报错，要么模型出来乱码输出）：

```
# 复制之前转换好的/app/Qwen3-0.6B/mcore-tp1-pp1一份，改名成mcore-trained
# 删除mcore-trained下面的release文件夹
# 复制iter_xxxxxxx文件夹，比如iter_0000500到mcore-trained路径下
# 将iter_0000500改名成release（关键）

cd /app/Pai-Megatron-Patch/toolkits/distributed_checkpoints_convertor

bash scripts/qwen3/run_8xH20.sh \
0.6B \
/app/Qwen3-0.6B/mcore-trained \
/app/Qwen3-0.6B/hf  \
true \
true \
bf16 \
/app/Qwen3-0.6B

# /app/Qwen3-0.6B/mcore-trained 表示带有mcore训练好的模型路径
# /app/Qwen3-0.6B/hf 是最终转换成hf模型文件的路径，也就是最终可以直接推理的模型了
# /app/Qwen3-0.6B 表示带有原始hf模型路径（需要相关的tokenizer之类的文件）
```

二. Vllm模型推理

1. 简介

现在我们就可以使用上面训练好的模型来进行推理了，同时我也用助教给的100条基础数据集训练了Qwen3-0.6B上传到了modelscope，供大家下载使用：mxy333333/Qwen3-0.6B-GPU-basic（仅用了100条数据集，目前还没有测试能有多少准确率，但是估计应该不差的），同时模型里面的一个qwen-zh-basic.json文件是使用的训练数据集，如果需要更多数据集或者其他模型比如Qwen2.5-0.5B，可以按照上面步骤类似训练，Pai-Megatron-Patch这个项目已经写明了Qwen2.5如何训练

（偷偷告诉你，并行科技的智算云平台
好像会额外送50rmb的额度哦）

这里我以batch推理为例，首先模型需要初始化一些参数，同时我们先进行预热以及前缀缓存

```
# 本地模型路径
LOCAL_MODEL_PATH = "./local-model/mxy333333/Qwen3-0.6B-GPU-basic"
DATASET_PATH = "./local-model/mxy333333/Qwen3-0.6B-GPU-basic/qwen-zh-basic.json

@lru_cache(maxsize=10000)
def format_prompt(tokenizer, msg:  str) -> str:
    message = [{"role": "user", "content": msg}]
    return tokenizer.apply_chat_template(
        message,
        tokenize=False,
        add_generation_prompt=True,
        enable_thinking=False
    )
################################### 初始化部分 ###################################
# 1. 设置路径
model_dir = LOCAL_MODEL_PATH

tokenizer = AutoTokenizer.from_pretrained(model_dir, trust_remote_code=True)

# 2. 准备 Prompt 预热推理
json_path = DATASET_PATH
with open(json_path, 'r', encoding='utf-8') as f:
    data = json.load(f)
input_text = [item["instruction"] for item in data]
prompt_list = [format_prompt(tokenizer, item) for item in input_text]
warmup_prompt_list = prompt_list * 3


# 3. 配置采样参数 (SamplingParams)
sampling_params = SamplingParams(
    temperature=0,
    top_k=1,
    # top_p=0.8,
    max_tokens=256,  # 对应 max_new_tokens
    stop=["\n\n", "<|endoftext|>", "<|im_end|>"],
    stop_token_ids=[tokenizer.eos_token_id],
)

# 4. 初始化 vLLM 引擎
llm = LLM(
    model=model_dir,
    dtype="bfloat16",
    quantization="fp8",
    # kv_cache_dtype="fp8",
    trust_remote_code=True,
    tensor_parallel_size=1,
    gpu_memory_utilization=0.95,
    enforce_eager=False,
    max_model_len=1024,
    max_num_seqs=1024,
    enable_prefix_caching=True,
    disable_log_stats=True,
)

# 5. 执行预热推理
```

1. 由于我们使用megatron训练模型时有一个步骤需要将json数据集转换成模型训练需要的数据集格式，在这个过程中实际上将输入输出按照模型的chat_template格式处理过了，所以我们在推理的时候也用template处理一下，并且使用lru_cache将message缓存（其实用处不是很大），如果是Qwen3模型支持thinking，我们关掉来提高吞吐

```
@lru_cache(maxsize=10000)
def format_prompt(tokenizer, msg:  str) -> str:
    message = [{"role": "user", "content": msg}]
    return tokenizer.apply_chat_template(
        message,
        tokenize=False,
        add_generation_prompt=True,
        enable_thinking=False
    )
```

2. 我们将训练的数据集作为预热数据，在还没有进行测试的时候先跑一遍，一方面预热设备使后续测试的时候吞吐量更高，另一方面我们开启了前缀缓存，如果后续测试的时候输入和前面输入有一致的前缀则不再计算，因为我们知道数据集中至少100条是测试中用到的，所以这部分在预热阶段已经缓存了，到时候测试的时候可以减少一定的时间

```
# 2. 准备 Prompt 预热推理
json_path = DATASET_PATH
with open(json_path, 'r', encoding='utf-8') as f:
    data = json.load(f)
input_text = [item["instruction"] for item in data]
prompt_list = [format_prompt(tokenizer, item) for item in input_text]
warmup_prompt_list = prompt_list * 3
```

3. 配置采样参数，这里我们不需要temperature之类的参数，这样可以进一步提高吞吐，增加stop token可以帮助模型提早结束本条内容的输出（但是实测好像没太大用）。max_tokens是模型允许的最大token长度，这里我们设置了256，目前基础数据集256应该是完全够的，甚至你也可以调整更小（我估计64也行），这里我没有再继续尝试，进一步减小应该也是可以的，感兴趣同学可以尝试刷榜，这个参数提高吞吐还是挺明显的

```
# 3. 配置采样参数 (SamplingParams)
sampling_params = SamplingParams(
    temperature=0,
    top_k=1,
    # top_p=0.8,
    max_tokens=256,
    stop=["\n\n", "<|endoftext|>", "<|im_end|>"],
    stop_token_ids=[tokenizer.eos_token_id],
)
```

4. Vllm引擎初始化，首先我们对quantization开启fp8可以直接使用fp8数据格式参与推理，充分利用5090硬件特性提高吞吐量，并且准确率几乎不减；kv_cache也可以开启fp8，但是实测没啥用还有可能降低吞吐，所以关闭了；enforce_eager可以在预热阶段compile，保存下来计算图，提高测试过程的速度；max_num_seqs表示同时处理批次的数量，我们测试就300条左右数据，再大其实也没用；enable_prefix_caching表示开启前缀缓存

另外我也尝试过其他量化格式的模型速度，发现好像直接fp8就是最快的了，感兴趣可以参考下面两个项目：

Model-Optimizer

```
# 4. 初始化 vLLM 引擎
llm = LLM(
    model=model_dir,
    dtype="bfloat16",
    quantization="fp8",
    # kv_cache_dtype="fp8",
    trust_remote_code=True,
    tensor_parallel_size=1,
    gpu_memory_utilization=0.95,
    enforce_eager=False,
    max_model_len=1024,
    max_num_seqs=1024,
    enable_prefix_caching=True,
    disable_log_stats=True,
)
```

5. 最后调用generate执行一次推理预热

6. 正式测试就比较简单了，只是注意我们要使用batch模式来推理

```
################################## 正式推理部分 ##################################
if isinstance(request.prompt, str):
    real_input_list = [request.prompt]
else:
    real_input_list = request.prompt

final_prompt_texts = [format_prompt(tokenizer,msg) for msg in real_input_list]
# vLLM 自动进行高效的批量推理
outputs = llm.generate(final_prompt_texts, sampling_params)

generated = [output.outputs[0].text for output in outputs]
print(f"成功生成 {len(generated)} 条结果")
```

下面是完整的serve代码

```
import sys
import os
os.environ["TRANSFORMERS_OFFLINE"] = "1"
os.environ["TORCH_CUDA_ARCH_LIST"] = "12.0"
import torch
from fastapi import FastAPI
from pydantic import BaseModel
from transformers import pipeline, set_seed
from transformers import AutoModelForCausalLM, AutoTokenizer
import socket
import json
from vllm import LLM, SamplingParams
from typing import Union, List
from functools import lru_cache

def check_internet(host="8.8.8.8", port=53, timeout=3):
    try:
        socket.setdefaulttimeout(timeout)
        socket.socket(socket.AF_INET, socket.SOCK_STREAM).connect((host, port))
        return True
    except Exception:
        return False

@lru_cache(maxsize=10000)
```

```
        message,
        tokenize=False,
        add_generation_prompt=True,
        enable_thinking=False
    )
# 本地模型路径
LOCAL_MODEL_PATH = "./local-model/mxy333333/Qwen3-0.6B-GPU-basic"
DATASET_PATH = "./local-model/mxy333333/Qwen3-0.6B-GPU-basic/qwen-zh-basic.json"

# --- 网络连通性测试 ---
internet_ok = check_internet()
print("【Internet Connectivity Test】:",
      "CONNECTED" if internet_ok else "OFFLINE / BLOCKED")

# --- 模型加载（从本地加载，无需网络）---
print(f"从本地加载模型: {LOCAL_MODEL_PATH}")
################################### 初始化部分 ###################################
# 1. 设置路径
model_dir = LOCAL_MODEL_PATH

tokenizer = AutoTokenizer.from_pretrained(model_dir, trust_remote_code=True)

# 2. 准备 Prompt 预热推理
json_path = DATASET_PATH
with open(json_path, 'r', encoding='utf-8') as f:
    data = json.load(f)
input_text = [item["instruction"] for item in data]
prompt_list = [format_prompt(tokenizer, item) for item in input_text]
warmup_prompt_list = prompt_list * 3


# 3. 配置采样参数 (SamplingParams)
sampling_params = SamplingParams(
    temperature=0,
    top_k=1,
    # top_p=0.8,
    max_tokens=256,
    stop=["\n\n", "<|endoftext|>", "<|im_end|>"],
    stop_token_ids=[tokenizer.eos_token_id],
)

# 4. 初始化 vLLM 引擎
llm = LLM(
    model=model_dir,
    dtype="bfloat16",
    quantization="fp8",
    # kv_cache_dtype="fp8",
    trust_remote_code=True,
    tensor_parallel_size=1,
    gpu_memory_utilization=0.95,
    enforce_eager=False,
    max_model_len=1024,
    max_num_seqs=1024,
    enable_prefix_caching=True,
    disable_log_stats=True,
)

# 5. 执行预热推理

print(f"开始预热推理")
outputs = llm.generate(warmup_pror
```

```
# 创建FastAPI应用实例
app = FastAPI(
    title="Simple Inference Server",
    description="A simple API to run a small language model."
)

# 定义API请求的数据模型
class PromptRequest(BaseModel):
    prompt: Union[str, List[str]]

# 定义API响应的数据模型
class PredictResponse(BaseModel):
    response: Union[str, List[str]]

# --- API 端点 ---
@app.post("/predict", response_model=PredictResponse)
async def predict(request: PromptRequest):
    """
    接收一个prompt，使用加载的模型进行推理，并返回结果。
    """

    ################################## 正式推理部分 ##############################
    if isinstance(request.prompt, str):
        real_input_list = [request.prompt]
    else:
        real_input_list = request.prompt

    final_prompt_texts = [format_prompt(tokenizer,msg) for msg in real_input_list]
    # vLLM 自动进行高效的批量推理
    outputs = llm.generate(final_prompt_texts, sampling_params)

    generated = [output.outputs[0].text for output in outputs]
    print(f"成功生成 {len(generated)} 条结果")
    # print(generated)


    return PredictResponse(response=generated)


@app.get("/")
def health_check():
    """
    健康检查端点，用于确认服务是否启动成功。
    """
    return {"status": "batch"}
```

下面是down_load模型代码

```
#验证 ModelScope token
from modelscope.hub.api import HubApi
api = HubApi()
api.login('YOUR-MODELSCOPE-TOKEN')

model_name = 'mxy333333/Qwen3-0.6B-GPU-basic'
local_model_path = './local-model'

#模型下载
from modelscope import snapshot_download
model_dir = snapshot_download(
```

```
)
```

这样修改完这两个代码之后，我们返回最初的docker build过程检测是否成功

三. LM-Studio部署模型

由于本课程期末考试可以使用没有网络的电脑，这里我们可以选择将模型通过LM-Studio部署，我们的模型只有0.6B比较小，所以也还算跑的起来，这里我展示mac下模型部署的过程

首先下载LM-Studio：LM Studio - Local AI on your computer

Apple有一个专门为Apple Silicon优化的推理框架MLX，所以我们可以将模型转换成mlx格式，这在LM-Studio是同样支持的

```
# 先安装mlx相关库，这里我是在5090上执行的，mlx同样有cuda版本，但是要求12.9的cuda会和当
pip install mlx[cpu]
pip install mlx-lm==0.29.1
```

然后将模型转换成mlx格式，好像也可以使用量化参数，但是我们模型已经比较小了，我这里没有量化

```
from mlx_lm import convert

# 定义模型路径
hf_path = "/app/Qwen3-0.6B/hf"
mlx_path = "/app/Qwen3-0.6B/mlx"

# 执行转换
convert(
    hf_path=hf_path,
    mlx_path=mlx_path,
    # quantize=True,
    # q_bits=4,
    # q_group_size=64
)

print(f"模型已转换并保存至：{mlx_path}")
```

模型上传到modelscope或者huggingface

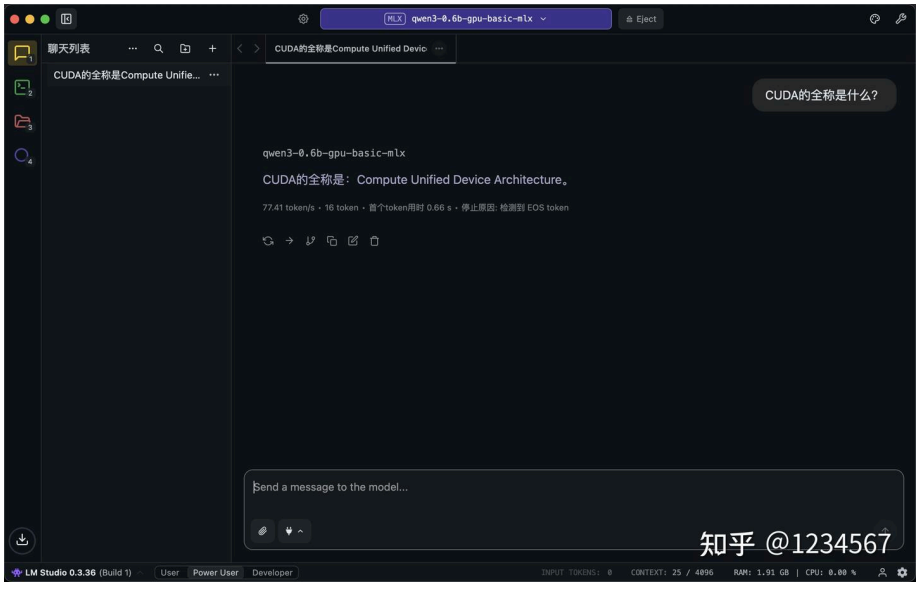
然后在mac端，我们下载相应的模型，这里我也将对应的MLX模型开源了供大家下载使用：mxy333333/Qwen3-0.6B-GPU-basic-MLX（同样只是用了100条数据集训练的）

```
modelscope download --model mxy333333/Qwen3-0.6B-GPU-basic-MLX --local_dir ~/.lmst
```

注意比较关键的是LM-Studio有一个规定的模型存放路径~/.lmstudio/models，同时模型文件Qwen3-0.6B-GPU-basic-MLX前面也必须要有一个Qwen路径，这样才能正常识别到

如果下载本地成功，点击左侧边栏的文件夹图标会出现你的模型

现在就可以加载模型进行推理了，点击左侧边栏聊天图标，上方可以点击模型选择，如下，我们可以看到有MLX标志，选择合适的模型加载，就可以开始推理了



对于LM-Studio的其他功能，以及在Windows平台的使用，大家可以自行探索

编辑于 2025-12-31 21:16 · 日本

理性发言，友善互动

7 条评论 默认 最新

海绵A梦
请问大佬试自己设计了加分题的推理prompt了嘛？只是做了微调，别的技术都没用嘛~还想请教一下，加分题微调到loss多少可以停下来呀，呜呜呜呜
2 分钟前 · 北京 回复 喜欢

cccc
0.5b就可以完成加分题准确率要求了吗🧐
3 小时前 · 北京 回复 喜欢

1234567 作者
对的，我用qwen2.5–0.5b就可以达到0.39的准确率了，所以还有将近0.04的准确率可以浪费来提升吞吐。估计用更小的模型也是可以的，另外建议加分题数据集加上基础题的数据集会更好🫣
2 小时前 · 北京 回复 喜欢

你好
😲
2025-12-31 · 北京 回复 喜欢

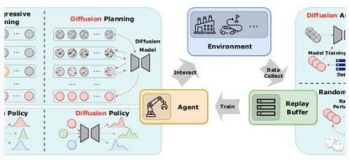
1234567 作者
🥵
2 小时前 · 北京 回复 喜欢

知乎用户Cc
感谢大佬
2025-12-31 · 北京 回复 喜欢

1234567 作者
😘
2 小时前 · 北京 回复 喜欢

推荐阅读

大模型实习笔记 之 强化学习(前沿篇)
本文主要探索这条从经典强化学习到大模型对齐前沿技术的路径，需要你有基本的强化学习知识与大模型背景。内容包括： 基础篇：直觉与推导（REINFORCE 算法与策略梯度核心）进化篇：稳定与效...
iTheresaApocalypse



强化学习+扩散模型的综述
泳鱼

推荐系统时代前沿（9）：强个性化的非梯度场景——强化...
水哥 发表于从零单排推...

用基于规则的强化学习训练模型自主调用搜索引擎，从而增加...
Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning 论文地址：https://arxiv.org/abs/2503.09516 研究背景 研究问题：这篇文章要...
生锅