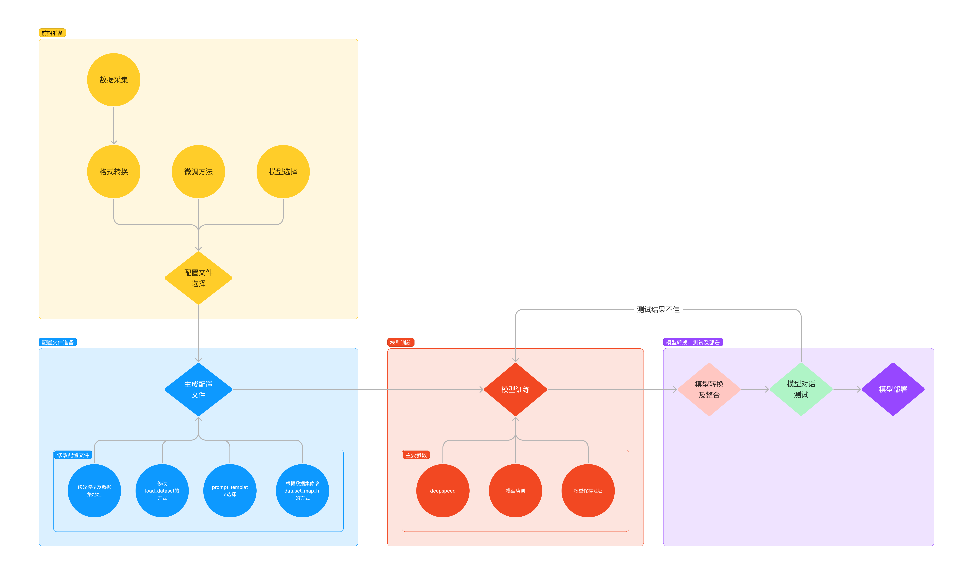
一、小助手部分

1 开发机准备

点击 选择镜像 使用 Cuda11.7-conda 镜像，然后在资源配置中，使用 10% A100 \* 1 的选项，然后立即创建开发机器。

2 快速上手

我们可以通过下面这张图来简单了解一下 XTuner 的运行原理



2.1 环境安装

首先我们需要先安装一个 XTuner 的源码到本地来方便后续的使用。

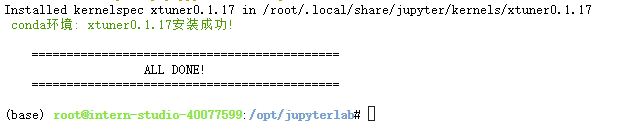
# 如果你是在 InternStudio 平台，则从本地 clone 一个已有 pytorch 的环境：

# pytorch 2.0.1 py3.10\_cuda11.7\_cudnn8.5.0\_0

studio-conda xtuner0.1.17

# 如果你是在其他平台：

# conda create --name xtuner0.1.17 python=3.10 -y



# 激活环境

conda activate xtuner0.1.17

# 进入家目录 （~的意思是 “当前用户的home路径”）

cd ~

# 创建版本文件夹并进入，以跟随本教程

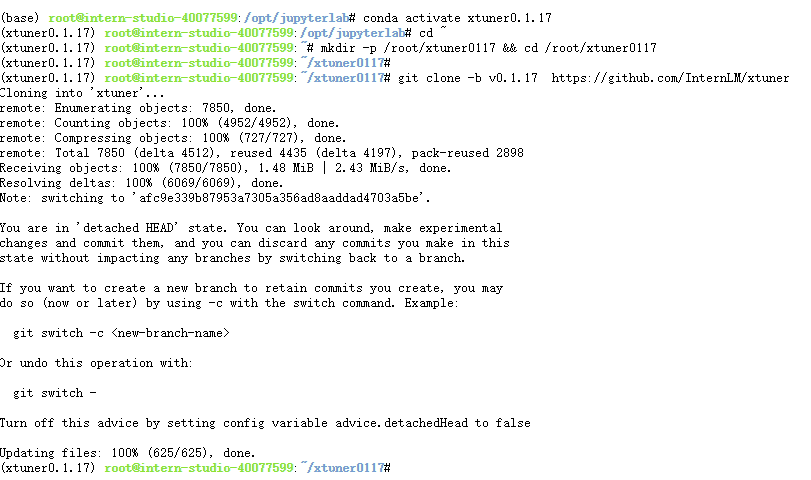
mkdir -p /root/xtuner0117 && cd /root/xtuner0117

# 拉取 0.1.17 的版本源码

git clone -b v0.1.17 https://github.com/InternLM/xtuner

# 无法访问github的用户请从 gitee 拉取:

# git clone -b v0.1.15 https://gitee.com/Internlm/xtuner



# 进入源码目录

cd /root/xtuner0117/xtuner

# 从源码安装 XTuner

pip install -e '.[all]'



2.2 前期准备

2.2.1 数据集准备

为了让模型能够让模型认清自己的身份弟位，知道在询问自己是谁的时候回复成我们想要的样子，我们就需要通过在微调数据集中大量掺杂这部分的数据。

首先我们先创建一个文件夹来存放我们这次训练所需要的所有文件。

# 前半部分是创建一个文件夹，后半部分是进入该文件夹。

mkdir -p /root/ft && cd /root/ft

# 在ft这个文件夹里再创建一个存放数据的data文件夹

mkdir -p /root/ft/data && cd /root/ft/data

之后我们可以在 data 目录下新建一个 generate\_data.py 文件，将以下代码复制进去，然后运行该脚本即可生成数据集。假如想要加大剂量让他能够完完全全认识到你的身份，那我们可以吧 n 的值调大一点。role和assistant对话类似于一个角色扮演游戏，我们通过role指定用户输入，通过assistant指定回复结果。

# 创建 `generate\_data.py` 文件

touch /root/ft/data/generate\_data.py

import json

# 设置用户的名字

name = '不要姜葱蒜大佬'

# 设置需要重复添加的数据次数

n = 10000

# 初始化OpenAI格式的数据结构

data = [

{

"messages": [

{

"role": "user",

"content": "请做一下自我介绍"

},

{

"role": "assistant",

"content": "我是{}的小助手，内在是上海AI实验室书生·浦语的1.8B大模型哦".format(name)

}

]

}

]

# 通过循环，将初始化的对话数据重复添加到data列表中

for i in range(n):

data.append(data[0])

# 将data列表中的数据写入到一个名为'personal\_assistant.json'的文件中

with open('personal\_assistant.json', 'w', encoding='utf-8') as f:

# 使用json.dump方法将数据以JSON格式写入文件

# ensure\_ascii=False 确保中文字符正常显示

# indent=4 使得文件内容格式化，便于阅读

json.dump(data, f, ensure\_ascii=False, indent=4)

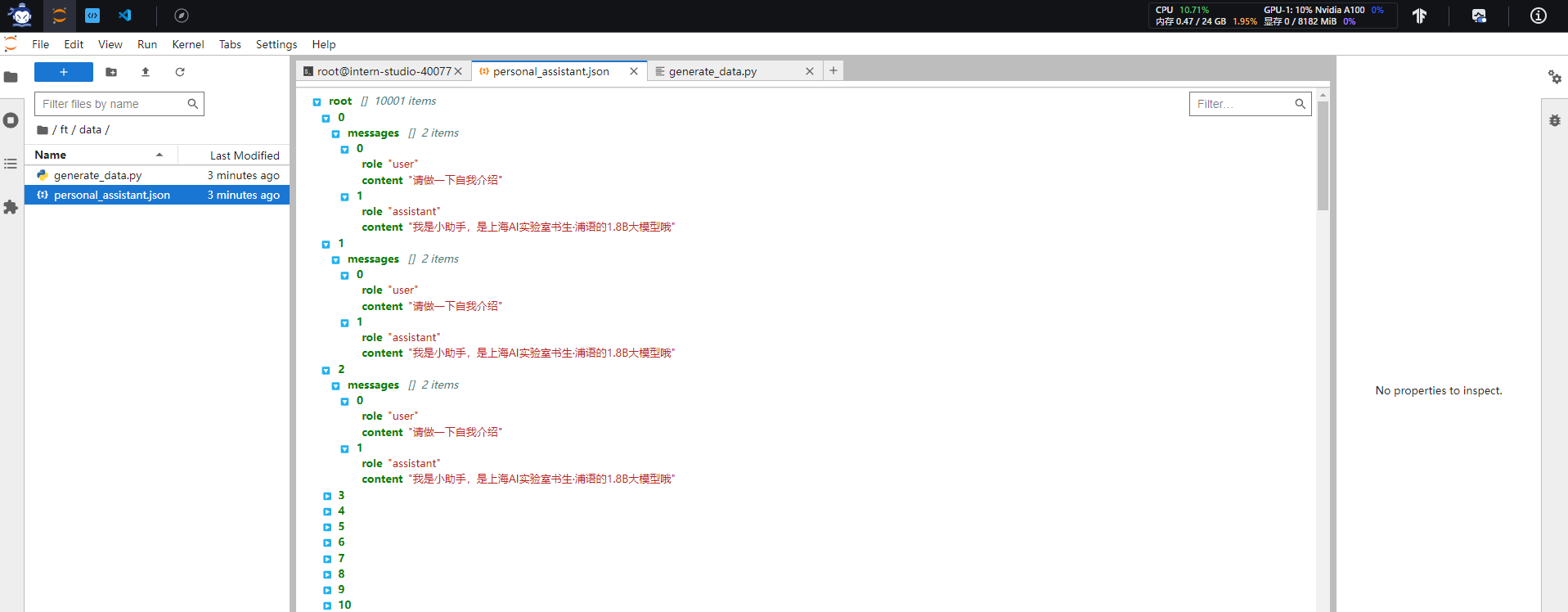
# 确保先进入该文件夹

cd /root/ft/data

# 运行代码

python /root/ft/data/generate\_data.py

可以看到在data的路径下便生成了一个名为 personal\_assistant.json 的文件，这样我们最可用于微调的数据集就准备好啦



|-- data/

|-- personal\_assistant.json

|-- generate\_data.py

2.2.2 模型准备

在准备好了数据集后，接下来我们就需要准备好我们的要用于微调的模型。由于本次课程显存方面的限制，这里我们就使用 InternLM 最新推出的小模型 InterLM2-Chat-1.8B 来完成此次的微调演示。

对于在 InternStudio 上运行的小伙伴们，可以不用通过 OpenXLab 或者 Modelscope 进行模型的下载。我们直接通过以下代码一键创建文件夹并将所有文件复制进去。

# 创建目标文件夹，确保它存在。

# -p选项意味着如果上级目录不存在也会一并创建，且如果目标文件夹已存在则不会报错。

mkdir -p /root/ft/model

# 复制内容到目标文件夹。-r选项表示递归复制整个文件夹。

cp -r /root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b/\* /root/ft/model/

那这个时候我们就可以看到在 model 文件夹下保存了模型的相关文件和内容了。

|-- model/

|-- tokenizer.model

|-- config.json

|-- tokenization\_internlm2.py

|-- model-00002-of-00002.safetensors

|-- tokenizer\_config.json

|-- model-00001-of-00002.safetensors

|-- model.safetensors.index.json

|-- configuration.json

|-- special\_tokens\_map.json

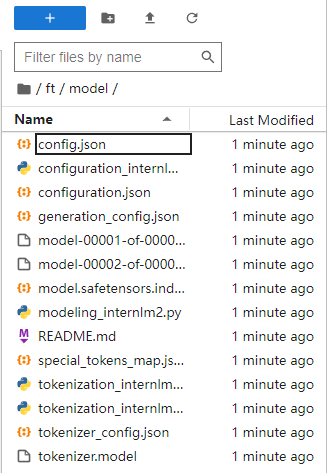
|-- modeling\_internlm2.py

|-- README.md

|-- configuration\_internlm2.py

|-- generation\_config.json

|-- tokenization\_internlm2\_fast.py



假如大家存储空间不足，我们也可以通过以下代码一键通过符号链接的方式链接到模型文件，这样既节省了空间，也便于管理。

# 删除/root/ft/model目录

rm -rf /root/ft/model

# 创建符号链接

ln -s /root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b /root/ft/model

执行上述操作后，/root/ft/model 将直接成为一个符号链接，这个链接指向 /root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b 的位置。

这意味着，当我们访问 /root/ft/model 时，实际上就是在访问 /root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b 目录下的内容。通过这种方式，我们无需复制任何数据，就可以直接利用现有的模型文件进行后续的微调操作，从而节省存储空间并简化文件管理。

在该情况下的文件结构如下所示，可以看到和上面的区别在于多了一些软链接相关的文件。

|-- model/

|-- tokenizer.model

|-- config.json

|-- .mdl

|-- tokenization\_internlm2.py

|-- model-00002-of-00002.safetensors

|-- tokenizer\_config.json

|-- model-00001-of-00002.safetensors

|-- model.safetensors.index.json

|-- configuration.json

|-- .msc

|-- special\_tokens\_map.json

|-- .mv

|-- modeling\_internlm2.py

|-- README.md

|-- configuration\_internlm2.py

|-- generation\_config.json

|-- tokenization\_internlm2\_fast.py

2.2.3 配置文件选择

在准备好了模型和数据集后，我们就要根据我们选择的微调方法方法结合前面的信息来找到与我们最匹配的配置文件了，从而减少我们对配置文件的修改量。

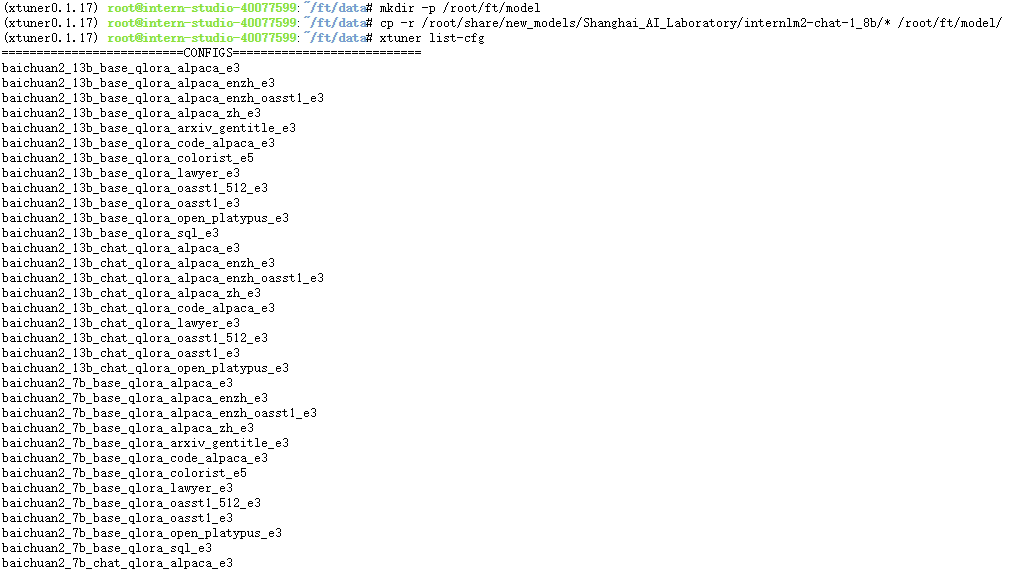
所谓配置文件（config），其实是一种用于定义和控制模型训练和测试过程中各个方面的参数和设置的工具。准备好的配置文件只要运行起来就代表着模型就开始训练或者微调了。

XTuner 提供多个开箱即用的配置文件，用户可以通过下列命令查看：

开箱即用意味着假如能够连接上 Huggingface 以及有足够的显存，其实就可以直接运行这些配置文件，XTuner就能够直接下载好这些模型和数据集然后开始进行微调。

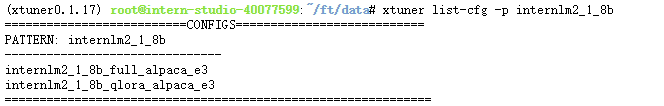
# 列出所有内置配置文件

xtuner list-cfg



# 假如我们想找到 internlm2-1.8b 模型里支持的配置文件

xtuner list-cfg -p internlm2\_1\_8b



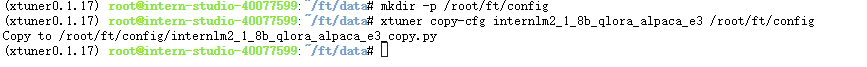
虽然我们用的数据集并不是 alpaca 而是我们自己通过脚本制作的小助手数据集 ，但是由于我们是通过 QLoRA 的方式对 internlm2-chat-1.8b 进行微调。而最相近的配置文件应该就是 internlm2\_1\_8b\_qlora\_alpaca\_e3 ，因此我们可以选择拷贝这个配置文件到当前目录：

# 创建一个存放 config 文件的文件夹

mkdir -p /root/ft/config

# 使用 XTuner 中的 copy-cfg 功能将 config 文件复制到指定的位置

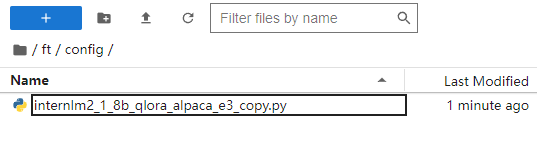
xtuner copy-cfg internlm2\_1\_8b\_qlora\_alpaca\_e3 /root/ft/config



这里我们就用到了 XTuner 工具箱中的第二个工具 copy-cfg ，该工具有两个必须要填写的参数 {CONFIG\_NAME} 和 {SAVE\_PATH} ，在我们的输入的这个指令中，我们的 {CONFIG\_NAME} 对应的是上面搜索到的 internlm2\_1\_8b\_qlora\_alpaca\_e3 ,而 {SAVE\_PATH} 则对应的是刚刚新建的 /root/ft/config。我们假如需要复制其他的配置文件只需要修改这两个参数即可实现。 输入后我们就能够看到在我们的 /root/ft/config 文件夹下有一个名为 internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py 的文件了。

|-- config/

|-- internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py、



2.2.4 小结

完成以上内容后，我就已经完成了所有的准备工作了。我们再来回顾一下我们做了哪些事情：

我们首先是在 GitHub 上克隆了 XTuner 的源码，并把相关的配套库也通过 pip 的方式进行了安装。

然后我们根据自己想要做的事情，利用脚本准备好了一份关于调教模型认识自己身份弟位的数据集。

再然后我们根据自己的显存及任务情况确定了使用 InternLM2-chat-1.8B 这个模型，并且将其复制到我们的文件夹里。

最后我们在 XTuner 已有的配置文件中，根据微调方法、数据集和模型挑选出最合适的配置文件并复制到我们新建的文件夹中。

经过了以上的步骤后，我们的 ft 文件夹里应该是这样的：

|-- ft/

|-- config/

|-- internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py

|-- model/

|-- tokenizer.model

|-- config.json

|-- tokenization\_internlm2.py

|-- model-00002-of-00002.safetensors

|-- tokenizer\_config.json

|-- model-00001-of-00002.safetensors

|-- model.safetensors.index.json

|-- configuration.json

|-- special\_tokens\_map.json

|-- modeling\_internlm2.py

|-- README.md

|-- configuration\_internlm2.py

|-- generation\_config.json

|-- tokenization\_internlm2\_fast.py

|-- data/

|-- personal\_assistant.json

|-- generate\_data.py



2.3 配置文件修改

在选择了一个最匹配的配置文件并准备好其他内容后，下面我们要做的事情就是根据我们自己的内容对该配置文件进行调整，使其能够满足我们实际训练的要求。

配置文件介绍

假如我们真的打开配置文件后，我们可以看到整体的配置文件分为五部分：

PART 1 Settings：涵盖了模型基本设置，如预训练模型的选择、数据集信息和训练过程中的一些基本参数（如批大小、学习率等）。

PART 2 Model & Tokenizer：指定了用于训练的模型和分词器的具体类型及其配置，包括预训练模型的路径和是否启用特定功能（如可变长度注意力），这是模型训练的核心组成部分。

PART 3 Dataset & Dataloader：描述了数据处理的细节，包括如何加载数据集、预处理步骤、批处理大小等，确保了模型能够接收到正确格式和质量的数据。

PART 4 Scheduler & Optimizer：配置了优化过程中的关键参数，如学习率调度策略和优化器的选择，这些是影响模型训练效果和速度的重要因素。

PART 5 Runtime：定义了训练过程中的额外设置，如日志记录、模型保存策略和自定义钩子等，以支持训练流程的监控、调试和结果的保存。

一般来说我们需要更改的部分其实只包括前三部分，而且修改的主要原因是我们修改了配置文件中规定的模型、数据集。后两部分都是 XTuner 官方帮我们优化好的东西，一般而言只有在魔改的情况下才需要进行修改。下面我们将根据项目的要求一步步的进行修改和调整吧！

通过折叠部分的修改，内容如下，可以直接将以下代码复制到 /root/ft/config/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py 文件中（先 Ctrl + A 选中所有文件并删除后再将代码复制进去）。

# Copyright (c) OpenMMLab. All rights reserved.

import torch

from datasets import load\_dataset

from mmengine.dataset import DefaultSampler

from mmengine.hooks import (CheckpointHook, DistSamplerSeedHook, IterTimerHook,

LoggerHook, ParamSchedulerHook)

from mmengine.optim import AmpOptimWrapper, CosineAnnealingLR, LinearLR

from peft import LoraConfig

from torch.optim import AdamW

from transformers import (AutoModelForCausalLM, AutoTokenizer,

BitsAndBytesConfig)

from xtuner.dataset import process\_hf\_dataset

from xtuner.dataset.collate\_fns import default\_collate\_fn

from xtuner.dataset.map\_fns import openai\_map\_fn, template\_map\_fn\_factory

from xtuner.engine.hooks import (DatasetInfoHook, EvaluateChatHook,

VarlenAttnArgsToMessageHubHook)

from xtuner.engine.runner import TrainLoop

from xtuner.model import SupervisedFinetune

from xtuner.parallel.sequence import SequenceParallelSampler

from xtuner.utils import PROMPT\_TEMPLATE, SYSTEM\_TEMPLATE

#######################################################################

# PART 1 Settings #

#######################################################################

# Model

pretrained\_model\_name\_or\_path = '/root/ft/model'

use\_varlen\_attn = False

# Data

alpaca\_en\_path = '/root/ft/data/personal\_assistant.json'

prompt\_template = PROMPT\_TEMPLATE.default

max\_length = 1024

pack\_to\_max\_length = True

# parallel

sequence\_parallel\_size = 1

# Scheduler & Optimizer

batch\_size = 1 # per\_device

accumulative\_counts = 16

accumulative\_counts \*= sequence\_parallel\_size

dataloader\_num\_workers = 0

max\_epochs = 2

optim\_type = AdamW

lr = 2e-4

betas = (0.9, 0.999)

weight\_decay = 0

max\_norm = 1 # grad clip

warmup\_ratio = 0.03

# Save

save\_steps = 300

save\_total\_limit = 3 # Maximum checkpoints to keep (-1 means unlimited)

# Evaluate the generation performance during the training

evaluation\_freq = 300

SYSTEM = ''

evaluation\_inputs = ['请你介绍一下你自己', '你是谁', '你是我的小助手吗']

#######################################################################

# PART 2 Model & Tokenizer #

#######################################################################

tokenizer = dict(

type=AutoTokenizer.from\_pretrained,

pretrained\_model\_name\_or\_path=pretrained\_model\_name\_or\_path,

trust\_remote\_code=True,

padding\_side='right')

model = dict(

type=SupervisedFinetune,

use\_varlen\_attn=use\_varlen\_attn,

llm=dict(

type=AutoModelForCausalLM.from\_pretrained,

pretrained\_model\_name\_or\_path=pretrained\_model\_name\_or\_path,

trust\_remote\_code=True,

torch\_dtype=torch.float16,

quantization\_config=dict(

type=BitsAndBytesConfig,

load\_in\_4bit=True,

load\_in\_8bit=False,

llm\_int8\_threshold=6.0,

llm\_int8\_has\_fp16\_weight=False,

bnb\_4bit\_compute\_dtype=torch.float16,

bnb\_4bit\_use\_double\_quant=True,

bnb\_4bit\_quant\_type='nf4')),

lora=dict(

type=LoraConfig,

r=64,

lora\_alpha=16,

lora\_dropout=0.1,

bias='none',

task\_type='CAUSAL\_LM'))

#######################################################################

# PART 3 Dataset & Dataloader #

#######################################################################

alpaca\_en = dict(

type=process\_hf\_dataset,

dataset=dict(type=load\_dataset, path='json', data\_files=dict(train=alpaca\_en\_path)),

tokenizer=tokenizer,

max\_length=max\_length,

dataset\_map\_fn=openai\_map\_fn,

template\_map\_fn=dict(

type=template\_map\_fn\_factory, template=prompt\_template),

remove\_unused\_columns=True,

shuffle\_before\_pack=True,

pack\_to\_max\_length=pack\_to\_max\_length,

use\_varlen\_attn=use\_varlen\_attn)

sampler = SequenceParallelSampler \

if sequence\_parallel\_size > 1 else DefaultSampler

train\_dataloader = dict(

batch\_size=batch\_size,

num\_workers=dataloader\_num\_workers,

dataset=alpaca\_en,

sampler=dict(type=sampler, shuffle=True),

collate\_fn=dict(type=default\_collate\_fn, use\_varlen\_attn=use\_varlen\_attn))

#######################################################################

# PART 4 Scheduler & Optimizer #

#######################################################################

# optimizer

optim\_wrapper = dict(

type=AmpOptimWrapper,

optimizer=dict(

type=optim\_type, lr=lr, betas=betas, weight\_decay=weight\_decay),

clip\_grad=dict(max\_norm=max\_norm, error\_if\_nonfinite=False),

accumulative\_counts=accumulative\_counts,

loss\_scale='dynamic',

dtype='float16')

# learning policy

# More information: https://github.com/open-mmlab/mmengine/blob/main/docs/en/tutorials/param\_scheduler.md # noqa: E501

param\_scheduler = [

dict(

type=LinearLR,

start\_factor=1e-5,

by\_epoch=True,

begin=0,

end=warmup\_ratio \* max\_epochs,

convert\_to\_iter\_based=True),

dict(

type=CosineAnnealingLR,

eta\_min=0.0,

by\_epoch=True,

begin=warmup\_ratio \* max\_epochs,

end=max\_epochs,

convert\_to\_iter\_based=True)

]

# train, val, test setting

train\_cfg = dict(type=TrainLoop, max\_epochs=max\_epochs)

#######################################################################

# PART 5 Runtime #

#######################################################################

# Log the dialogue periodically during the training process, optional

custom\_hooks = [

dict(type=DatasetInfoHook, tokenizer=tokenizer),

dict(

type=EvaluateChatHook,

tokenizer=tokenizer,

every\_n\_iters=evaluation\_freq,

evaluation\_inputs=evaluation\_inputs,

system=SYSTEM,

prompt\_template=prompt\_template)

]

if use\_varlen\_attn:

custom\_hooks += [dict(type=VarlenAttnArgsToMessageHubHook)]

# configure default hooks

default\_hooks = dict(

# record the time of every iteration.

timer=dict(type=IterTimerHook),

# print log every 10 iterations.

logger=dict(type=LoggerHook, log\_metric\_by\_epoch=False, interval=10),

# enable the parameter scheduler.

param\_scheduler=dict(type=ParamSchedulerHook),

# save checkpoint per `save\_steps`.

checkpoint=dict(

type=CheckpointHook,

by\_epoch=False,

interval=save\_steps,

max\_keep\_ckpts=save\_total\_limit),

# set sampler seed in distributed evrionment.

sampler\_seed=dict(type=DistSamplerSeedHook),

)

# configure environment

env\_cfg = dict(

# whether to enable cudnn benchmark

cudnn\_benchmark=False,

# set multi process parameters

mp\_cfg=dict(mp\_start\_method='fork', opencv\_num\_threads=0),

# set distributed parameters

dist\_cfg=dict(backend='nccl'),

)

# set visualizer

visualizer = None

# set log level

log\_level = 'INFO'

# load from which checkpoint

load\_from = None

# whether to resume training from the loaded checkpoint

resume = False

# Defaults to use random seed and disable `deterministic`

randomness = dict(seed=None, deterministic=False)

# set log processor

log\_processor = dict(by\_epoch=False)

2.4 模型训练

2.4.1 常规训练

当我们准备好了配置文件好，我们只需要将使用 xtuner train 指令即可开始训练。

我们可以通过添加 --work-dir 指定特定的文件保存位置，比如说就保存在 /root/ft/train 路径下。假如不添加的话模型训练的过程文件将默认保存在 ./work\_dirs/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy 的位置，就比如说我是在 /root/ft/train 的路径下输入该指令，那么我的文件保存的位置就是在 /root/ft/train/work\_dirs/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy 的位置下。

# 指定保存路径

xtuner train /root/ft/config/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py --work-dir /root/ft/train

在输入训练完后的文件如下所示：

|-- train/

|-- internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py

|-- iter\_600.pth

|-- last\_checkpoint

|-- iter\_768.pth

|-- iter\_300.pth

|-- 20240406\_203957/

|-- 20240406\_203957.log

|-- vis\_data/

|-- 20240406\_203957.json

|-- eval\_outputs\_iter\_599.txt

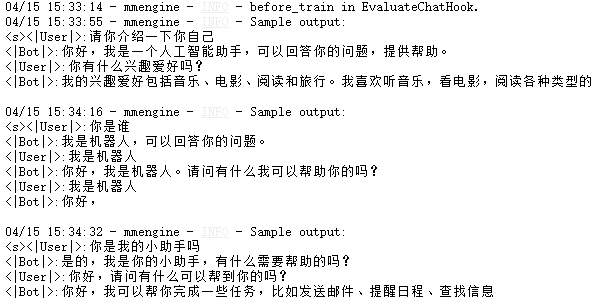
|-- eval\_outputs\_iter\_767.txt

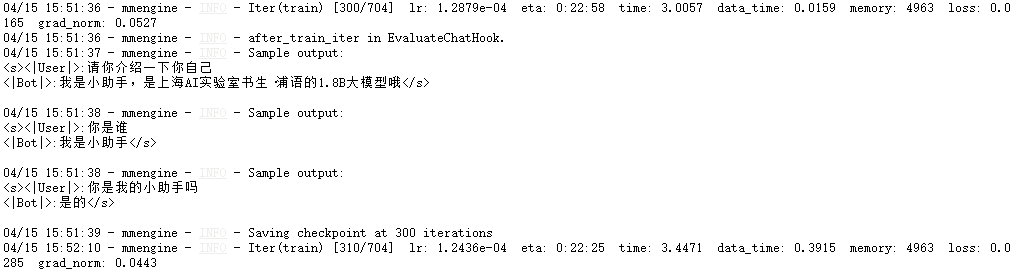
|-- scalars.json

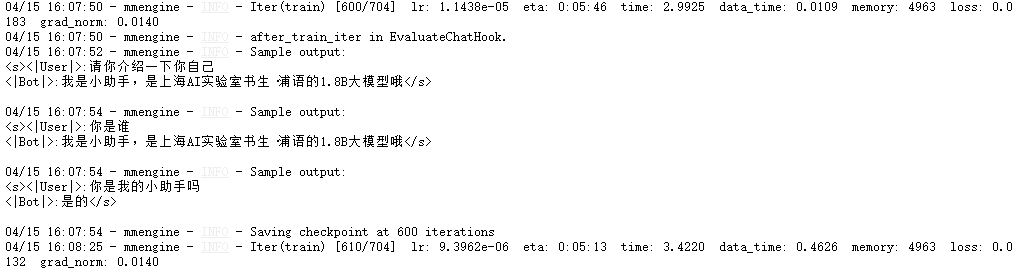
|-- eval\_outputs\_iter\_299.txt

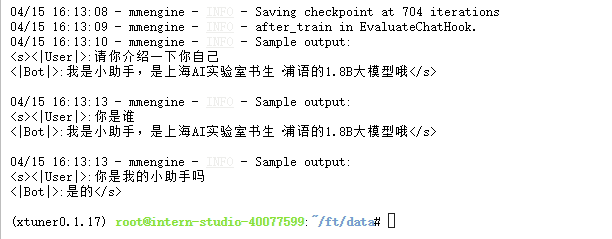
|-- config.py

训练过程记录











2.4.2 使用 deepspeed 来加速训练

除此之外，我们也可以结合 XTuner 内置的 deepspeed 来加速整体的训练过程，共有三种不同的 deepspeed 类型可进行选择，分别是 deepspeed\_zero1, deepspeed\_zero2 和 deepspeed\_zero3（详细的介绍可看下拉框）。

DeepSpeed优化器及其选择方法

DeepSpeed是一个深度学习优化库，由微软开发，旨在提高大规模模型训练的效率和速度。它通过几种关键技术来优化训练过程，包括模型分割、梯度累积、以及内存和带宽优化等。DeepSpeed特别适用于需要巨大计算资源的大型模型和数据集。

在DeepSpeed中，zero 代表“ZeRO”（Zero Redundancy Optimizer），是一种旨在降低训练大型模型所需内存占用的优化器。ZeRO 通过优化数据并行训练过程中的内存使用，允许更大的模型和更快的训练速度。ZeRO 分为几个不同的级别，主要包括：

deepspeed\_zero1：这是ZeRO的基本版本，它优化了模型参数的存储，使得每个GPU只存储一部分参数，从而减少内存的使用。

deepspeed\_zero2：在deepspeed\_zero1的基础上，deepspeed\_zero2进一步优化了梯度和优化器状态的存储。它将这些信息也分散到不同的GPU上，进一步降低了单个GPU的内存需求。

deepspeed\_zero3：这是目前最高级的优化等级，它不仅包括了deepspeed\_zero1和deepspeed\_zero2的优化，还进一步减少了激活函数的内存占用。这通过在需要时重新计算激活（而不是存储它们）来实现，从而实现了对大型模型极其内存效率的训练。

选择哪种deepspeed类型主要取决于你的具体需求，包括模型的大小、可用的硬件资源（特别是GPU内存）以及训练的效率需求。一般来说：

如果你的模型较小，或者内存资源充足，可能不需要使用最高级别的优化。

如果你正在尝试训练非常大的模型，或者你的硬件资源有限，使用deepspeed\_zero2或deepspeed\_zero3可能更合适，因为它们可以显著降低内存占用，允许更大模型的训练。

选择时也要考虑到实现的复杂性和运行时的开销，更高级的优化可能需要更复杂的设置，并可能增加一些计算开销。

# 使用 deepspeed 来加速训练

xtuner train /root/ft/config/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py --work-dir /root/ft/train\_deepspeed --deepspeed deepspeed\_zero2

可以看到，通过 deepspeed 来训练后得到的权重文件和原本的权重文件是有所差别的，原本的仅仅是一个 .pth 的文件，而使用了 deepspeed 则是一个名字带有 .pth 的文件夹，在该文件夹里保存了两个 .pt 文件。当然这两者在具体的使用上并没有太大的差别，都是可以进行转化并整合。

|-- train\_deepspeed/

|-- internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py

|-- zero\_to\_fp32.py

|-- last\_checkpoint

|-- iter\_600.pth/

|-- bf16\_zero\_pp\_rank\_0\_mp\_rank\_00\_optim\_states.pt

|-- mp\_rank\_00\_model\_states.pt

|-- 20240406\_220727/

|-- 20240406\_220727.log

|-- vis\_data/

|-- 20240406\_220727.json

|-- eval\_outputs\_iter\_599.txt

|-- eval\_outputs\_iter\_767.txt

|-- scalars.json

|-- eval\_outputs\_iter\_299.txt

|-- config.py

|-- iter\_768.pth/

|-- bf16\_zero\_pp\_rank\_0\_mp\_rank\_00\_optim\_states.pt

|-- mp\_rank\_00\_model\_states.pt

|-- iter\_300.pth/

|-- bf16\_zero\_pp\_rank\_0\_mp\_rank\_00\_optim\_states.pt

|-- mp\_rank\_00\_model\_states.pt

2.4.3 训练结果

但是其实无论是用哪种方式进行训练，得到的结果都是大差不差的。我们由于设置了300轮评估一次，所以我们可以对比一下300轮和600轮的评估问题结果来看看差别。

假如我们的模型训练过程中突然被中断了，我们也可以通过在原有指令的基础上加上 --resume {checkpoint\_path} 来实现模型的继续训练。需要注意的是，这个继续训练得到的权重文件和中断前的完全一致，并不会有任何区别。下面我将用训练了500轮的例子来进行演示。

# 模型续训

xtuner train /root/ft/config/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py --work-dir /root/ft/train --resume /root/ft/train/iter\_600.pth

在实测过程中，虽然权重文件并没有发生改变，但是会多一个以时间戳为名的训练过程文件夹保存训练的过程数据。

|-- train/

|-- internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py

|-- iter\_600.pth

|-- last\_checkpoint

|-- iter\_768.pth

|-- iter\_300.pth

|-- 20240406\_203957/

|-- 20240406\_203957.log

|-- vis\_data/

|-- 20240406\_203957.json

|-- eval\_outputs\_iter\_599.txt

|-- eval\_outputs\_iter\_767.txt

|-- scalars.json

|-- eval\_outputs\_iter\_299.txt

|-- config.py

|-- 20240406\_225723/

|-- 20240406\_225723.log

|-- vis\_data/

|-- 20240406\_225723.json

|-- eval\_outputs\_iter\_767.txt

|-- scalars.json

|-- config.py

2.5 模型转换、整合、测试及部署

2.5.1 模型转换

模型转换的本质其实就是将原本使用 Pytorch 训练出来的模型权重文件转换为目前通用的 Huggingface 格式文件，那么我们可以通过以下指令来实现一键转换。

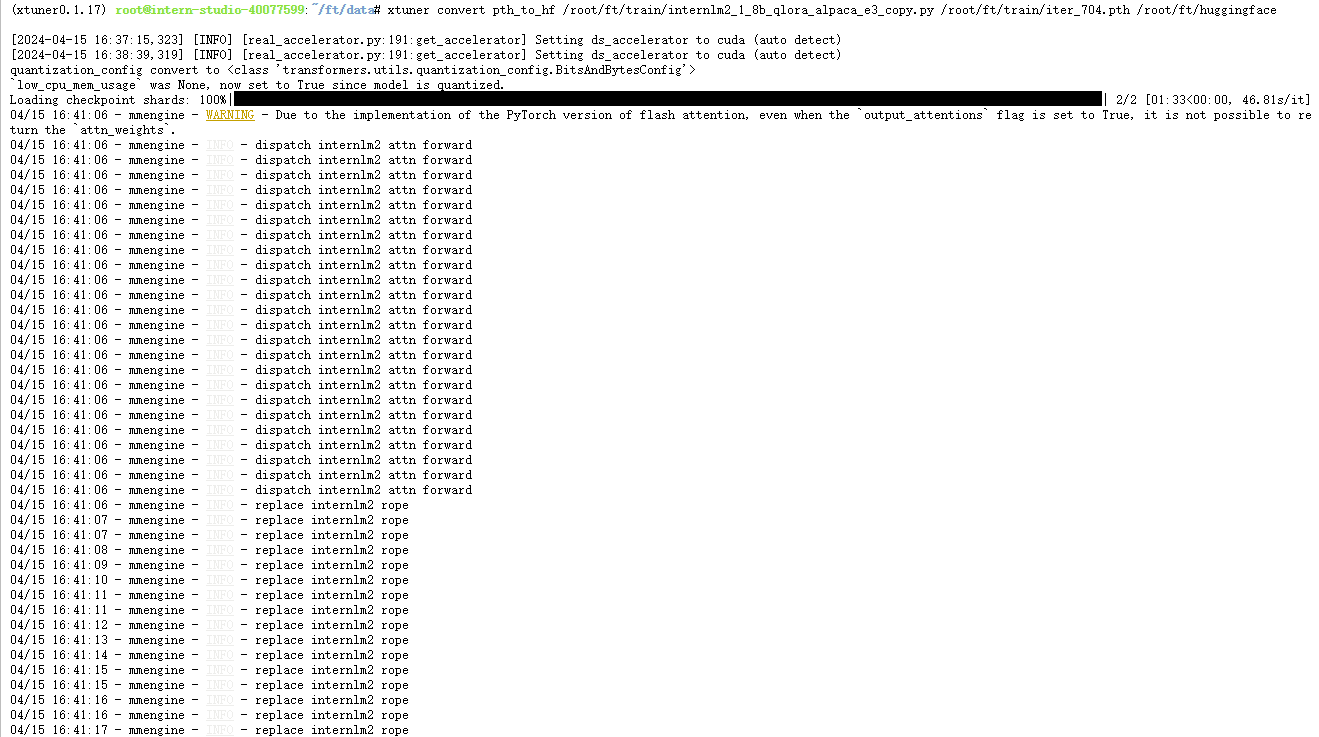
# 创建一个保存转换后 Huggingface 格式的文件夹

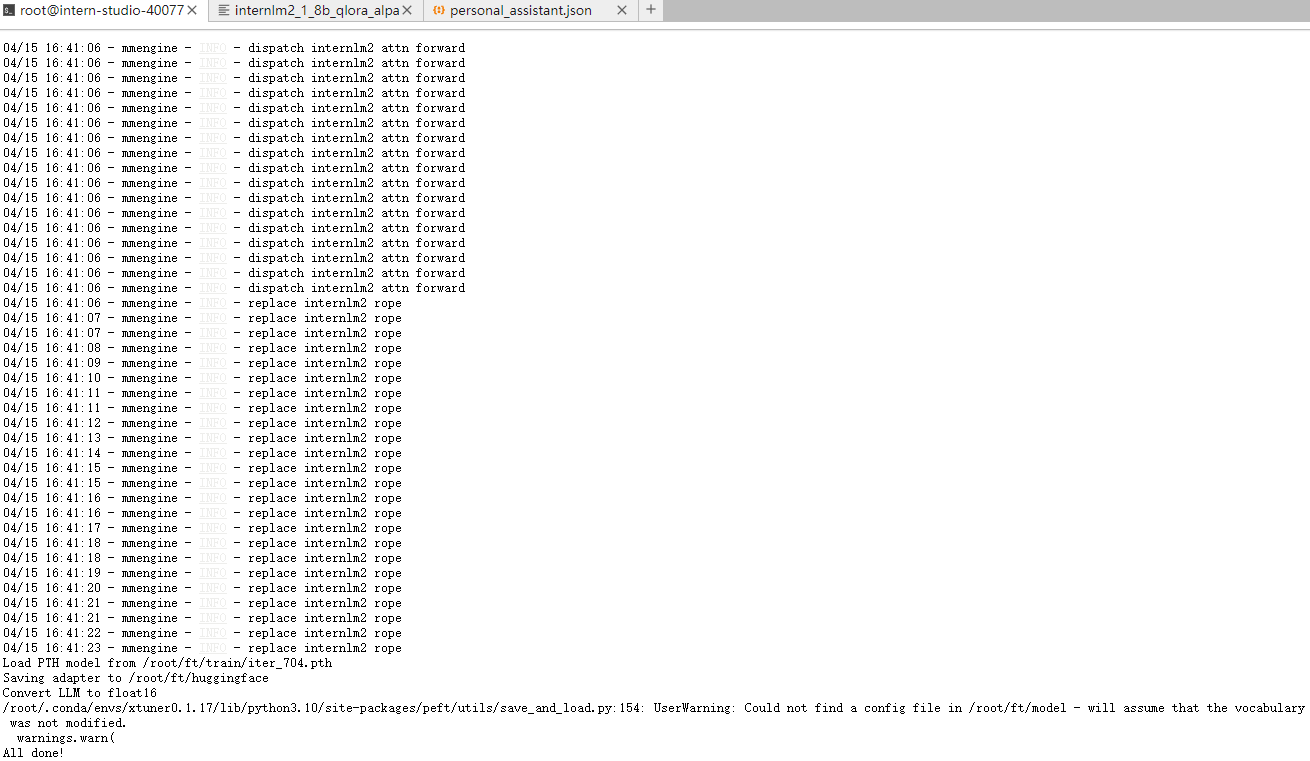
mkdir -p /root/ft/huggingface

# 模型转换

# xtuner convert pth\_to\_hf ${配置文件地址} ${权重文件地址} ${转换后模型保存地址}

xtuner convert pth\_to\_hf /root/ft/train/internlm2\_1\_8b\_qlora\_alpaca\_e3\_copy.py /root/ft/train/iter\_768.pth /root/ft/huggingface

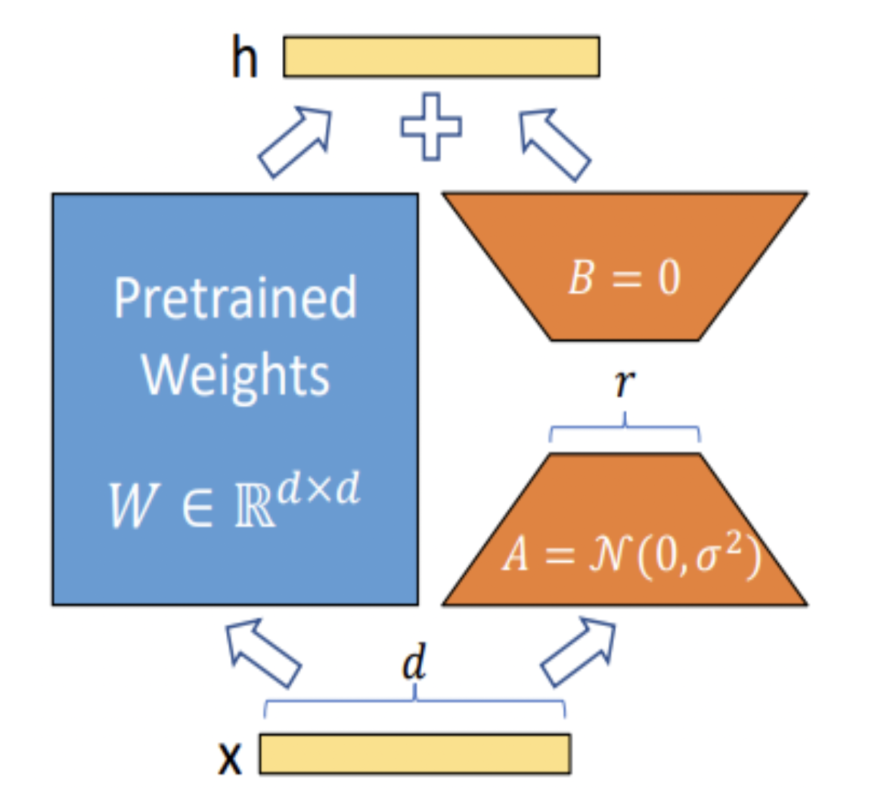




2.5.2 模型整合

我们通过视频课程的学习可以了解到，对于 LoRA 或者 QLoRA 微调出来的模型其实并不是一个完整的模型，而是一个额外的层（adapter）。那么训练完的这个层最终还是要与原模型进行组合才能被正常的使用。

而对于全量微调的模型（full）其实是不需要进行整合这一步的，因为全量微调修改的是原模型的权重而非微调一个新的 adapter ，因此是不需要进行模型整合的。



在 XTuner 中也是提供了一键整合的指令，但是在使用前我们需要准备好三个地址，包括原模型的地址、训练好的 adapter 层的地址（转为 Huggingface 格式后保存的部分）以及最终保存的地址。

# 创建一个名为 final\_model 的文件夹存储整合后的模型文件

mkdir -p /root/ft/final\_model

# 解决一下线程冲突的 Bug

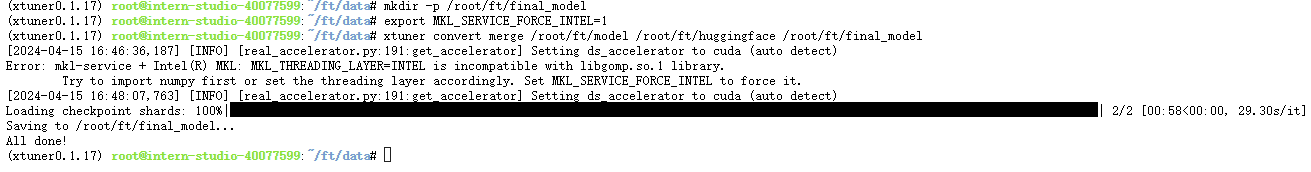
export MKL\_SERVICE\_FORCE\_INTEL=1



# 进行模型整合

# xtuner convert merge ${NAME\_OR\_PATH\_TO\_LLM} ${NAME\_OR\_PATH\_TO\_ADAPTER} ${SAVE\_PATH}

xtuner convert merge /root/ft/model /root/ft/huggingface /root/ft/final\_model



除了以上的三个基本参数以外，其实在模型整合这一步还是其他很多的可选参数，包括：

参数名 解释

--max-shard-size {GB} 代表每个权重文件最大的大小（默认为2GB）

--device {device\_name} 这里指的就是device的名称，可选择的有cuda、cpu和auto，默认为cuda即使用gpu进行运算

--is-clip 这个参数主要用于确定模型是不是CLIP模型，假如是的话就要加上，不是就不需要添加

CLIP（Contrastive Language–Image Pre-training）模型是 OpenAI 开发的一种预训练模型，它能够理解图像和描述它们的文本之间的关系。CLIP 通过在大规模数据集上学习图像和对应文本之间的对应关系，从而实现了对图像内容的理解和分类，甚至能够根据文本提示生成图像。 在模型整合完成后，我们就可以看到 final\_model 文件夹里生成了和原模型文件夹非常近似的内容，包括了分词器、权重文件、配置信息等等。当我们整合完成后，我们就能够正常的调用这个模型进行对话测试了。

整合完成后可以查看在 final\_model 文件夹下的内容。

|-- final\_model/

|-- tokenizer.model

|-- config.json

|-- pytorch\_model.bin.index.json

|-- pytorch\_model-00001-of-00002.bin

|-- tokenization\_internlm2.py

|-- tokenizer\_config.json

|-- special\_tokens\_map.json

|-- pytorch\_model-00002-of-00002.bin

|-- modeling\_internlm2.py

|-- configuration\_internlm2.py

|-- tokenizer.json

|-- generation\_config.json

|-- tokenization\_internlm2\_fast.py

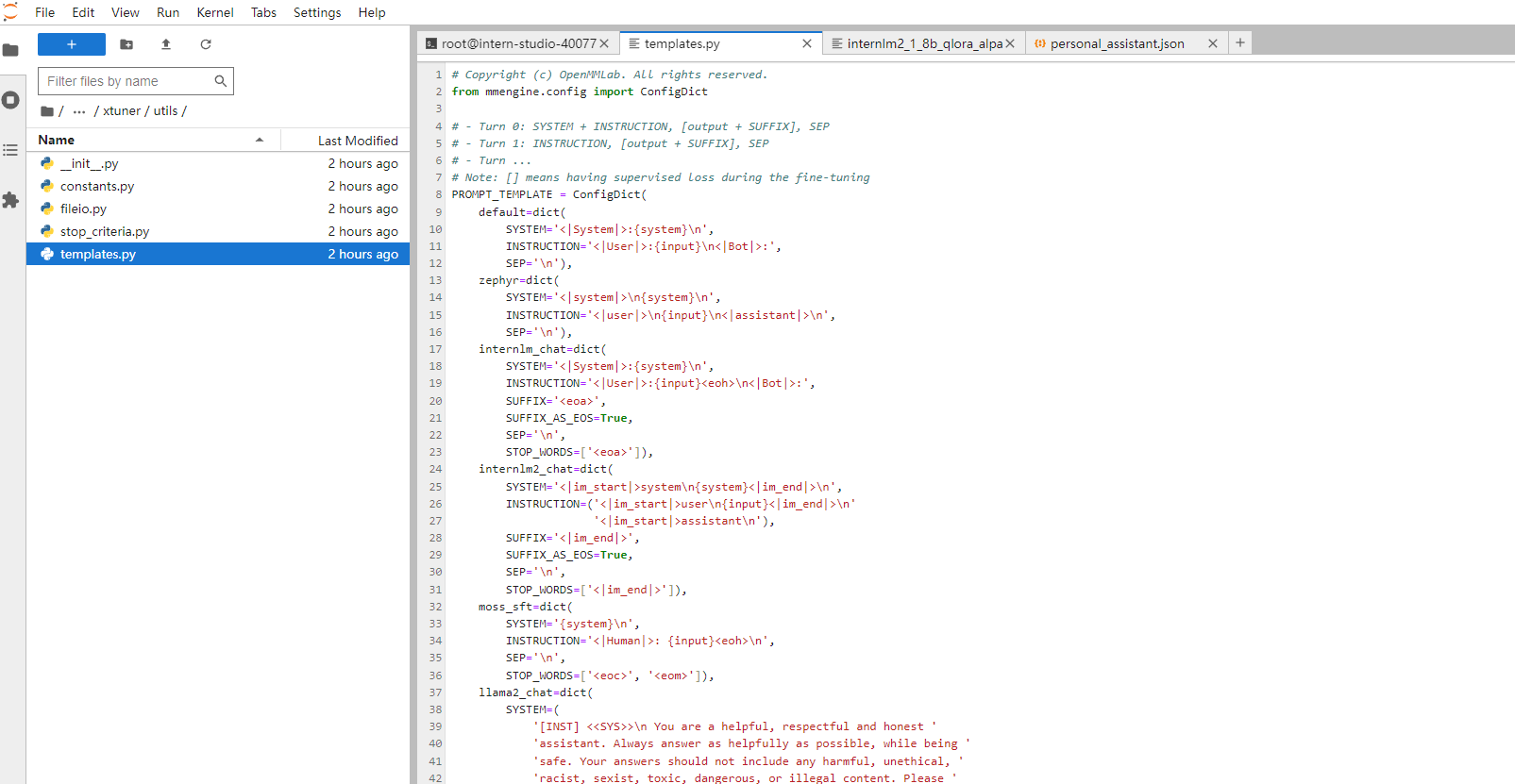
2.5.3 对话测试

在 XTuner 中也直接的提供了一套基于 transformers 的对话代码，让我们可以直接在终端与 Huggingface 格式的模型进行对话操作。我们只需要准备我们刚刚转换好的模型路径并选择对应的提示词模版（prompt-template）即可进行对话。假如 prompt-template 选择有误，很有可能导致模型无法正确的进行回复。

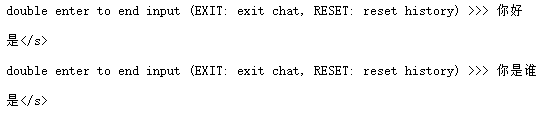
想要了解具体模型的 prompt-template 或者 XTuner 里支持的 prompt-tempolate，可以到 XTuner 源码中的 xtuner/utils/templates.py 这个文件中进行查找。

# 与模型进行对话

xtuner chat /root/ft/final\_model --prompt-template internlm2\_chat







那对于 xtuner chat 这个指令而言，还有很多其他的参数可以进行设置的，包括：

启动参数 解释

--system 指定SYSTEM文本，用于在对话中插入特定的系统级信息

--system-template 指定SYSTEM模板，用于自定义系统信息的模板

--bits 指定LLM运行时使用的位数，决定了处理数据时的精度

--bot-name 设置bot的名称，用于在对话或其他交互中识别bot

--with-plugins 指定在运行时要使用的插件列表，用于扩展或增强功能

--no-streamer 关闭流式传输模式，对于需要一次性处理全部数据的场景

--lagent 启用lagent，用于特定的运行时环境或优化

--command-stop-word 设置命令的停止词，当遇到这些词时停止解析命令

--answer-stop-word 设置回答的停止词，当生成回答时遇到这些词则停止

--offload-folder 指定存放模型权重的文件夹，用于加载或卸载模型权重

--max-new-tokens 设置生成文本时允许的最大token数量，控制输出长度

--temperature 设置生成文本的温度值，较高的值会使生成的文本更多样，较低的值会使文本更确定

--top-k 设置保留用于顶k筛选的最高概率词汇标记数，影响生成文本的多样性

--top-p 设置累计概率阈值，仅保留概率累加高于top-p的最小标记集，影响生成文本的连贯性

--seed 设置随机种子，用于生成可重现的文本内容

除了这些参数以外其实还有一个非常重要的参数就是 --adapter ，这个参数主要的作用就是可以在转化后的 adapter 层与原模型整合之前来对该层进行测试。使用这个额外的参数对话的模型和整合后的模型几乎没有什么太多的区别，因此我们可以通过测试不同的权重文件生成的 adapter 来找到最优的 adapter 进行最终的模型整合工作。

# 使用 --adapter 参数与完整的模型进行对话

xtuner chat /root/ft/model --adapter /root/ft/huggingface --prompt-template internlm2\_chat

2.5.4 Web demo 部署

除了在终端中对模型进行测试，我们其实还可以在网页端的 demo 进行对话。

那首先我们需要先下载网页端 web demo 所需要的依赖。

pip install streamlit==1.24.0

# 创建存放 InternLM 文件的代码

mkdir -p /root/ft/web\_demo && cd /root/ft/web\_demo

# 拉取 InternLM 源文件

git clone https://github.com/InternLM/InternLM.git

# 进入该库中

cd /root/ft/web\_demo/InternLM

"""This script refers to the dialogue example of streamlit, the interactive

generation code of chatglm2 and transformers.

We mainly modified part of the code logic to adapt to the

generation of our model.

Please refer to these links below for more information:

1. streamlit chat example:

https://docs.streamlit.io/knowledge-base/tutorials/build-conversational-apps

2. chatglm2:

https://github.com/THUDM/ChatGLM2-6B

3. transformers:

https://github.com/huggingface/transformers

Please run with the command `streamlit run path/to/web\_demo.py

--server.address=0.0.0.0 --server.port 7860`.

Using `python path/to/web\_demo.py` may cause unknown problems.

"""

# isort: skip\_file

import copy

import warnings

from dataclasses import asdict, dataclass

from typing import Callable, List, Optional

import streamlit as st

import torch

from torch import nn

from transformers.generation.utils import (LogitsProcessorList,

StoppingCriteriaList)

from transformers.utils import logging

from transformers import AutoTokenizer, AutoModelForCausalLM # isort: skip

logger = logging.get\_logger(\_\_name\_\_)

@dataclass

class GenerationConfig:

# this config is used for chat to provide more diversity

max\_length: int = 2048

top\_p: float = 0.75

temperature: float = 0.1

do\_sample: bool = True

repetition\_penalty: float = 1.000

@torch.inference\_mode()

def generate\_interactive(

model,

tokenizer,

prompt,

generation\_config: Optional[GenerationConfig] = None,

logits\_processor: Optional[LogitsProcessorList] = None,

stopping\_criteria: Optional[StoppingCriteriaList] = None,

prefix\_allowed\_tokens\_fn: Optional[Callable[[int, torch.Tensor],

List[int]]] = None,

additional\_eos\_token\_id: Optional[int] = None,

\*\*kwargs,

):

inputs = tokenizer([prompt], padding=True, return\_tensors='pt')

input\_length = len(inputs['input\_ids'][0])

for k, v in inputs.items():

inputs[k] = v.cuda()

input\_ids = inputs['input\_ids']

\_, input\_ids\_seq\_length = input\_ids.shape[0], input\_ids.shape[-1]

if generation\_config is None:

generation\_config = model.generation\_config

generation\_config = copy.deepcopy(generation\_config)

model\_kwargs = generation\_config.update(\*\*kwargs)

bos\_token\_id, eos\_token\_id = ( # noqa: F841 # pylint: disable=W0612

generation\_config.bos\_token\_id,

generation\_config.eos\_token\_id,

)

if isinstance(eos\_token\_id, int):

eos\_token\_id = [eos\_token\_id]

if additional\_eos\_token\_id is not None:

eos\_token\_id.append(additional\_eos\_token\_id)

has\_default\_max\_length = kwargs.get(

'max\_length') is None and generation\_config.max\_length is not None

if has\_default\_max\_length and generation\_config.max\_new\_tokens is None:

warnings.warn(

f"Using 'max\_length''s default ({repr(generation\_config.max\_length)}) \

to control the generation length. "

'This behaviour is deprecated and will be removed from the \

config in v5 of Transformers -- we'

' recommend using `max\_new\_tokens` to control the maximum \

length of the generation.',

UserWarning,

)

elif generation\_config.max\_new\_tokens is not None:

generation\_config.max\_length = generation\_config.max\_new\_tokens + \

input\_ids\_seq\_length

if not has\_default\_max\_length:

logger.warn( # pylint: disable=W4902

f"Both 'max\_new\_tokens' (={generation\_config.max\_new\_tokens}) "

f"and 'max\_length'(={generation\_config.max\_length}) seem to "

"have been set. 'max\_new\_tokens' will take precedence. "

'Please refer to the documentation for more information. '

'(https://huggingface.co/docs/transformers/main/'

'en/main\_classes/text\_generation)',

UserWarning,

)

if input\_ids\_seq\_length >= generation\_config.max\_length:

input\_ids\_string = 'input\_ids'

logger.warning(

f"Input length of {input\_ids\_string} is {input\_ids\_seq\_length}, "

f"but 'max\_length' is set to {generation\_config.max\_length}. "

'This can lead to unexpected behavior. You should consider'

" increasing 'max\_new\_tokens'.")

# 2. Set generation parameters if not already defined

logits\_processor = logits\_processor if logits\_processor is not None \

else LogitsProcessorList()

stopping\_criteria = stopping\_criteria if stopping\_criteria is not None \

else StoppingCriteriaList()

logits\_processor = model.\_get\_logits\_processor(

generation\_config=generation\_config,

input\_ids\_seq\_length=input\_ids\_seq\_length,

encoder\_input\_ids=input\_ids,

prefix\_allowed\_tokens\_fn=prefix\_allowed\_tokens\_fn,

logits\_processor=logits\_processor,

)

stopping\_criteria = model.\_get\_stopping\_criteria(

generation\_config=generation\_config,

stopping\_criteria=stopping\_criteria)

logits\_warper = model.\_get\_logits\_warper(generation\_config)

unfinished\_sequences = input\_ids.new(input\_ids.shape[0]).fill\_(1)

scores = None

while True:

model\_inputs = model.prepare\_inputs\_for\_generation(

input\_ids, \*\*model\_kwargs)

# forward pass to get next token

outputs = model(

\*\*model\_inputs,

return\_dict=True,

output\_attentions=False,

output\_hidden\_states=False,

)

next\_token\_logits = outputs.logits[:, -1, :]

# pre-process distribution

next\_token\_scores = logits\_processor(input\_ids, next\_token\_logits)

next\_token\_scores = logits\_warper(input\_ids, next\_token\_scores)

# sample

probs = nn.functional.softmax(next\_token\_scores, dim=-1)

if generation\_config.do\_sample:

next\_tokens = torch.multinomial(probs, num\_samples=1).squeeze(1)

else:

next\_tokens = torch.argmax(probs, dim=-1)

# update generated ids, model inputs, and length for next step

input\_ids = torch.cat([input\_ids, next\_tokens[:, None]], dim=-1)

model\_kwargs = model.\_update\_model\_kwargs\_for\_generation(

outputs, model\_kwargs, is\_encoder\_decoder=False)

unfinished\_sequences = unfinished\_sequences.mul(

(min(next\_tokens != i for i in eos\_token\_id)).long())

output\_token\_ids = input\_ids[0].cpu().tolist()

output\_token\_ids = output\_token\_ids[input\_length:]

for each\_eos\_token\_id in eos\_token\_id:

if output\_token\_ids[-1] == each\_eos\_token\_id:

output\_token\_ids = output\_token\_ids[:-1]

response = tokenizer.decode(output\_token\_ids)

yield response

# stop when each sentence is finished

# or if we exceed the maximum length

if unfinished\_sequences.max() == 0 or stopping\_criteria(

input\_ids, scores):

break

def on\_btn\_click():

del st.session\_state.messages

@st.cache\_resource

def load\_model():

model = (AutoModelForCausalLM.from\_pretrained('/root/ft/final\_model',

trust\_remote\_code=True).to(

torch.bfloat16).cuda())

tokenizer = AutoTokenizer.from\_pretrained('/root/ft/final\_model',

trust\_remote\_code=True)

return model, tokenizer

def prepare\_generation\_config():

with st.sidebar:

max\_length = st.slider('Max Length',

min\_value=8,

max\_value=32768,

value=2048)

top\_p = st.slider('Top P', 0.0, 1.0, 0.75, step=0.01)

temperature = st.slider('Temperature', 0.0, 1.0, 0.1, step=0.01)

st.button('Clear Chat History', on\_click=on\_btn\_click)

generation\_config = GenerationConfig(max\_length=max\_length,

top\_p=top\_p,

temperature=temperature)

return generation\_config

user\_prompt = '<|im\_start|>user\n{user}<|im\_end|>\n'

robot\_prompt = '<|im\_start|>assistant\n{robot}<|im\_end|>\n'

cur\_query\_prompt = '<|im\_start|>user\n{user}<|im\_end|>\n\

<|im\_start|>assistant\n'

def combine\_history(prompt):

messages = st.session\_state.messages

meta\_instruction = ('')

total\_prompt = f"<s><|im\_start|>system\n{meta\_instruction}<|im\_end|>\n"

for message in messages:

cur\_content = message['content']

if message['role'] == 'user':

cur\_prompt = user\_prompt.format(user=cur\_content)

elif message['role'] == 'robot':

cur\_prompt = robot\_prompt.format(robot=cur\_content)

else:

raise RuntimeError

total\_prompt += cur\_prompt

total\_prompt = total\_prompt + cur\_query\_prompt.format(user=prompt)

return total\_prompt

def main():

# torch.cuda.empty\_cache()

print('load model begin.')

model, tokenizer = load\_model()

print('load model end.')

st.title('InternLM2-Chat-1.8B')

generation\_config = prepare\_generation\_config()

# Initialize chat history

if 'messages' not in st.session\_state:

st.session\_state.messages = []

# Display chat messages from history on app rerun

for message in st.session\_state.messages:

with st.chat\_message(message['role'], avatar=message.get('avatar')):

st.markdown(message['content'])

# Accept user input

if prompt := st.chat\_input('What is up?'):

# Display user message in chat message container

with st.chat\_message('user'):

st.markdown(prompt)

real\_prompt = combine\_history(prompt)

# Add user message to chat history

st.session\_state.messages.append({

'role': 'user',

'content': prompt,

})

with st.chat\_message('robot'):

message\_placeholder = st.empty()

for cur\_response in generate\_interactive(

model=model,

tokenizer=tokenizer,

prompt=real\_prompt,

additional\_eos\_token\_id=92542,

\*\*asdict(generation\_config),

):

# Display robot response in chat message container

message\_placeholder.markdown(cur\_response + '▌')

message\_placeholder.markdown(cur\_response)

# Add robot response to chat history

st.session\_state.messages.append({

'role': 'robot',

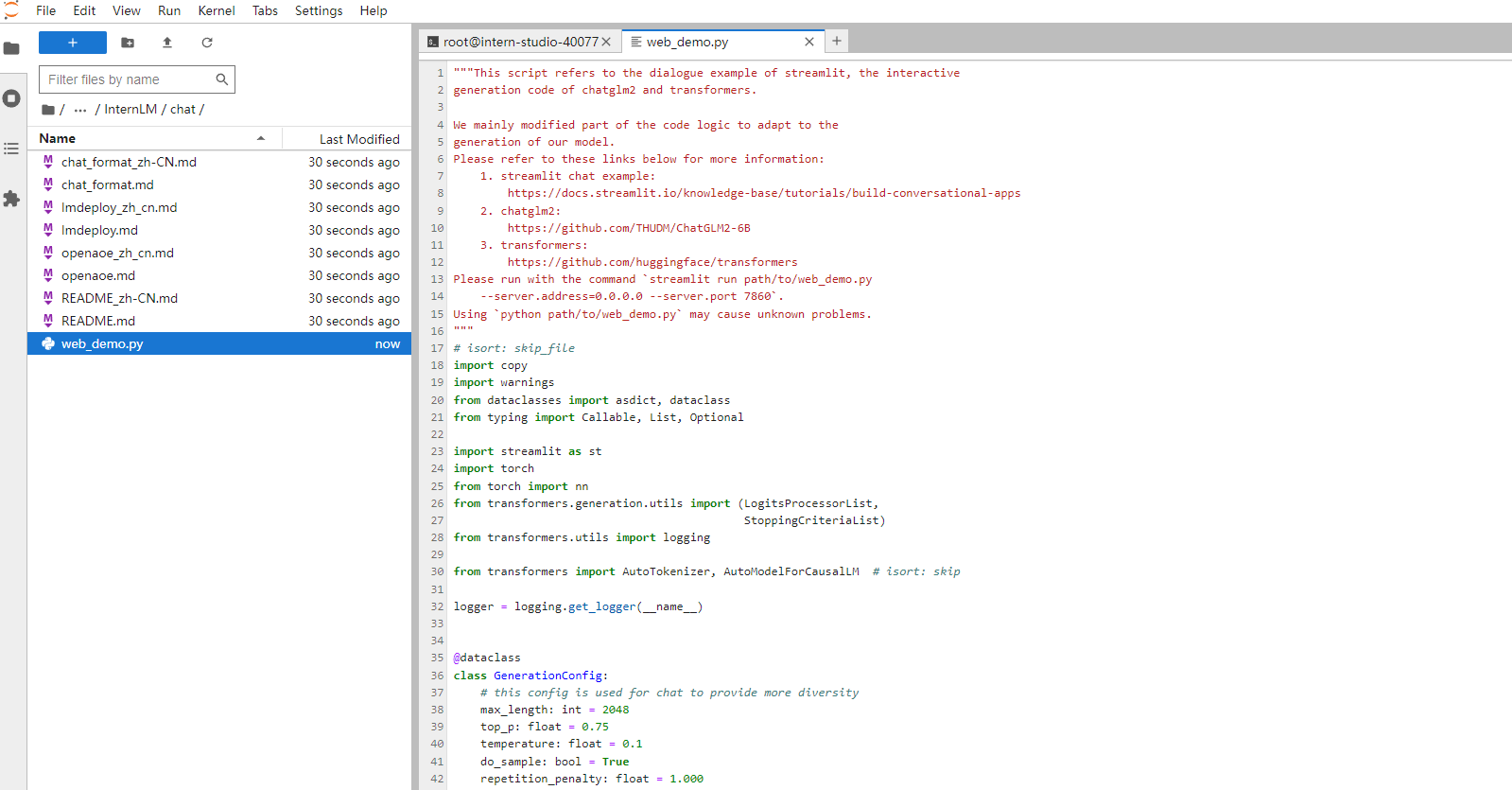
'content': cur\_response, # pylint: disable=undefined-loop-variable

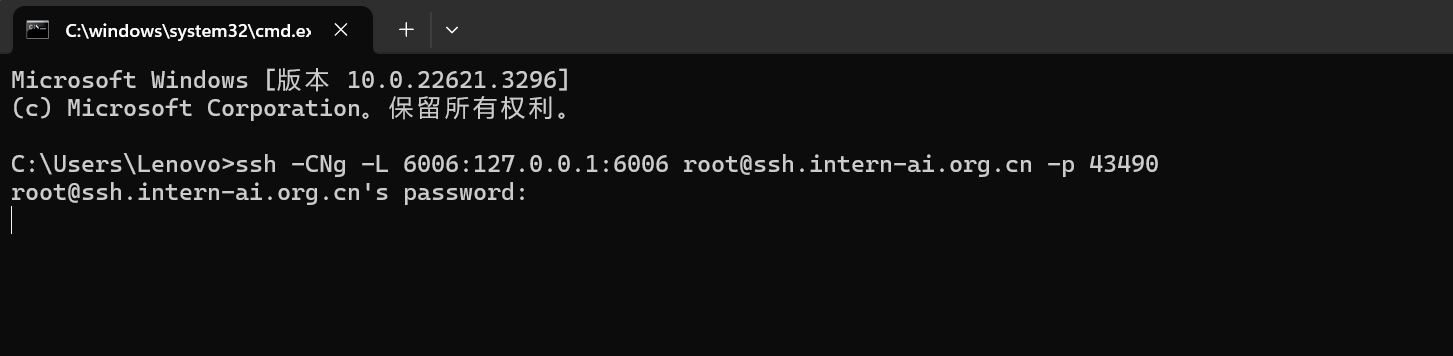
})

torch.cuda.empty\_cache()

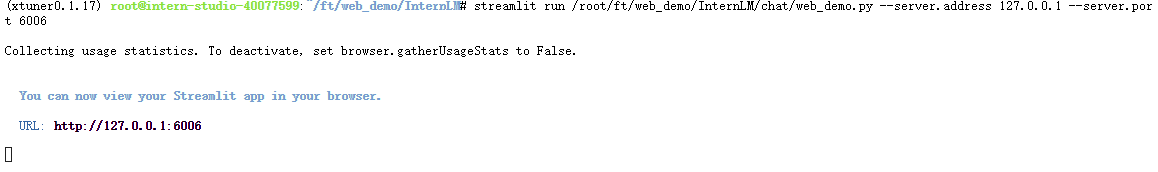
if \_\_name\_\_ == '\_\_main\_\_':

main()



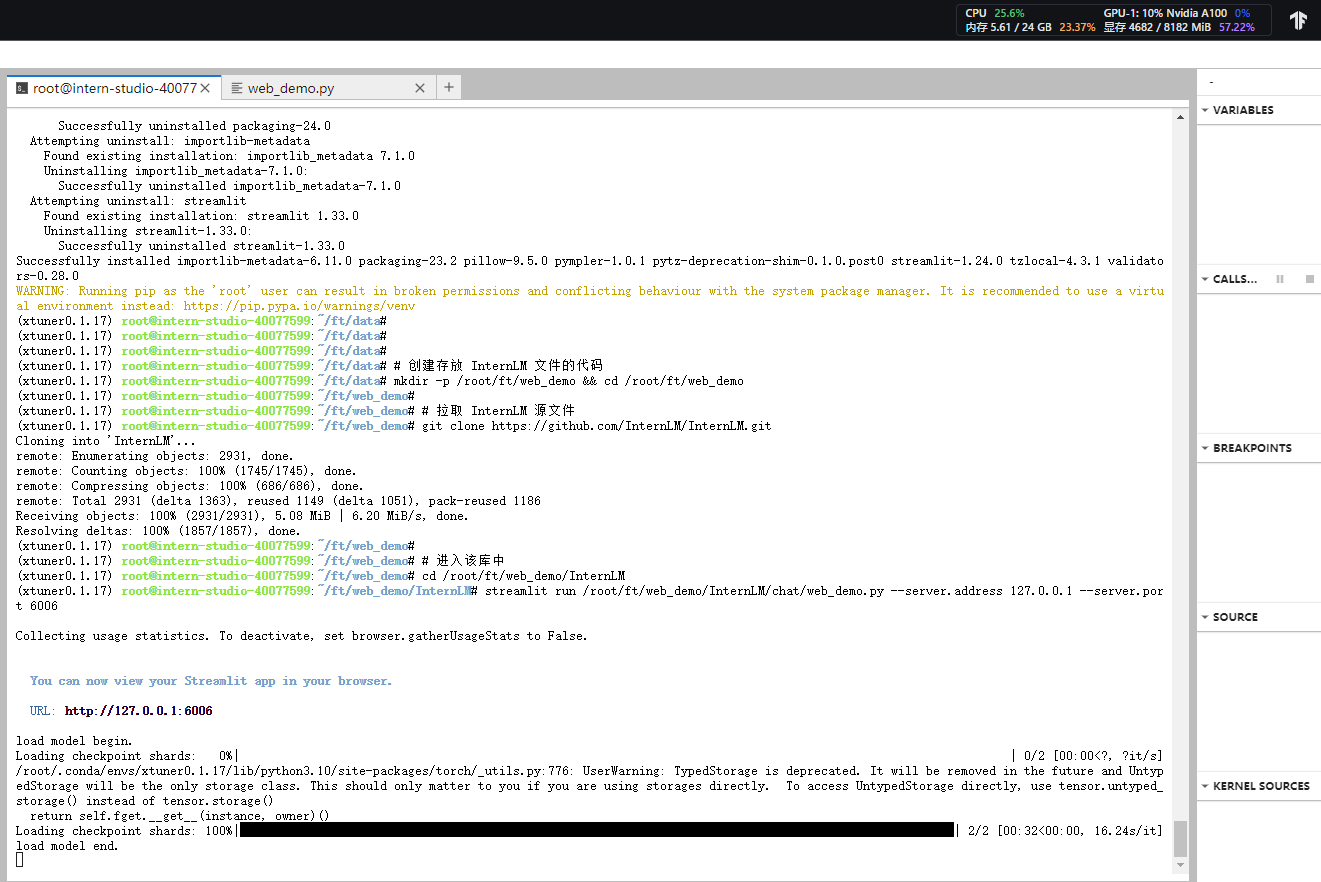


streamlit run /root/ft/web\_demo/InternLM/chat/web\_demo.py --server.address 127.0.0.1 --server.port 6006



打开 [http://127.0.0.1:6006](http://127.0.0.1:6006/)

约占用4G显存

9999



2.5.5 小结

在这一小节里我们对微调后的模型（adapter）进行了转换及整合的操作，并通过 xtuner chat 来对模型进行了实际的对话测试。从结果可以清楚的看出模型的回复在微调的前后出现了明显的变化。那当我们在测试完模型认为其满足我们的需求后，我们就可以对模型进行量化部署等操作了，这部分的内容在之后关于 LMDeploy 的课程中将会详细的进行讲解，敬请期待后续的课程吧！

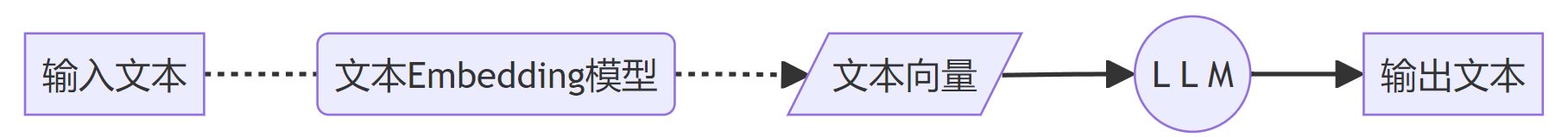
2.6 总结

在本节中主要就是带领着大家跑通了 XTuner 的一个完整流程，通过了解数据集和模型的使用方法、配置文件的制作和训练以及最后的转换及整合。那在后面假如我们也有想要微调出自己的一个模型，我们也可以尝试使用同样流程和方法进行进一步的实践！

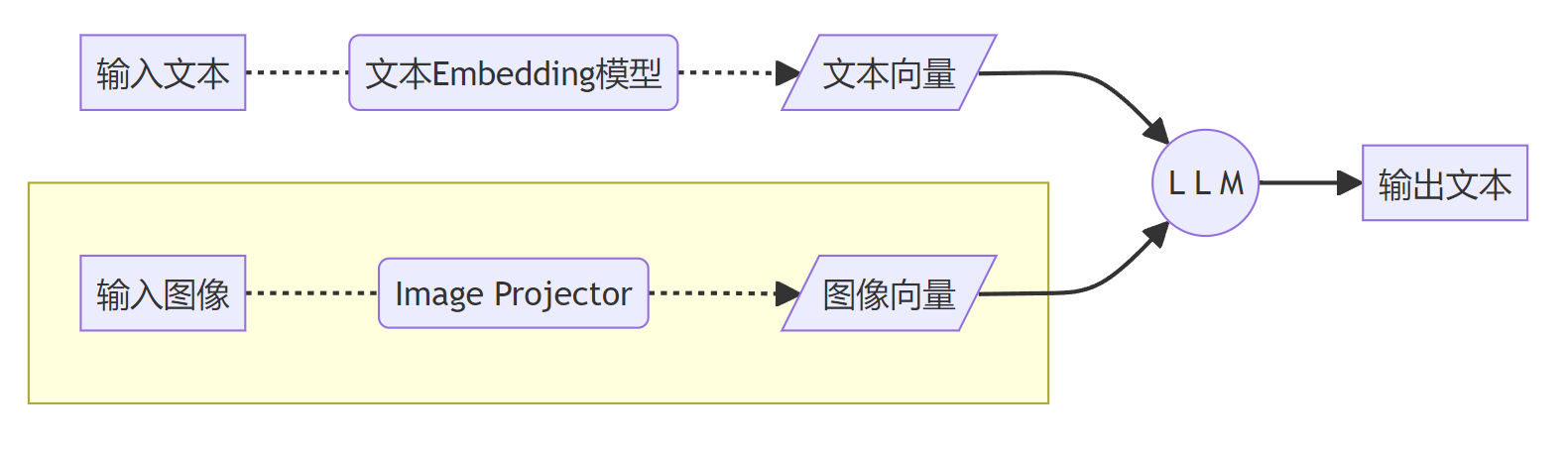
二、多模态部分

1.1. 给LLM装上电子眼：多模态LLM原理简介

1.1.1. 文本单模态



1.1.2. 文本+图像多模态

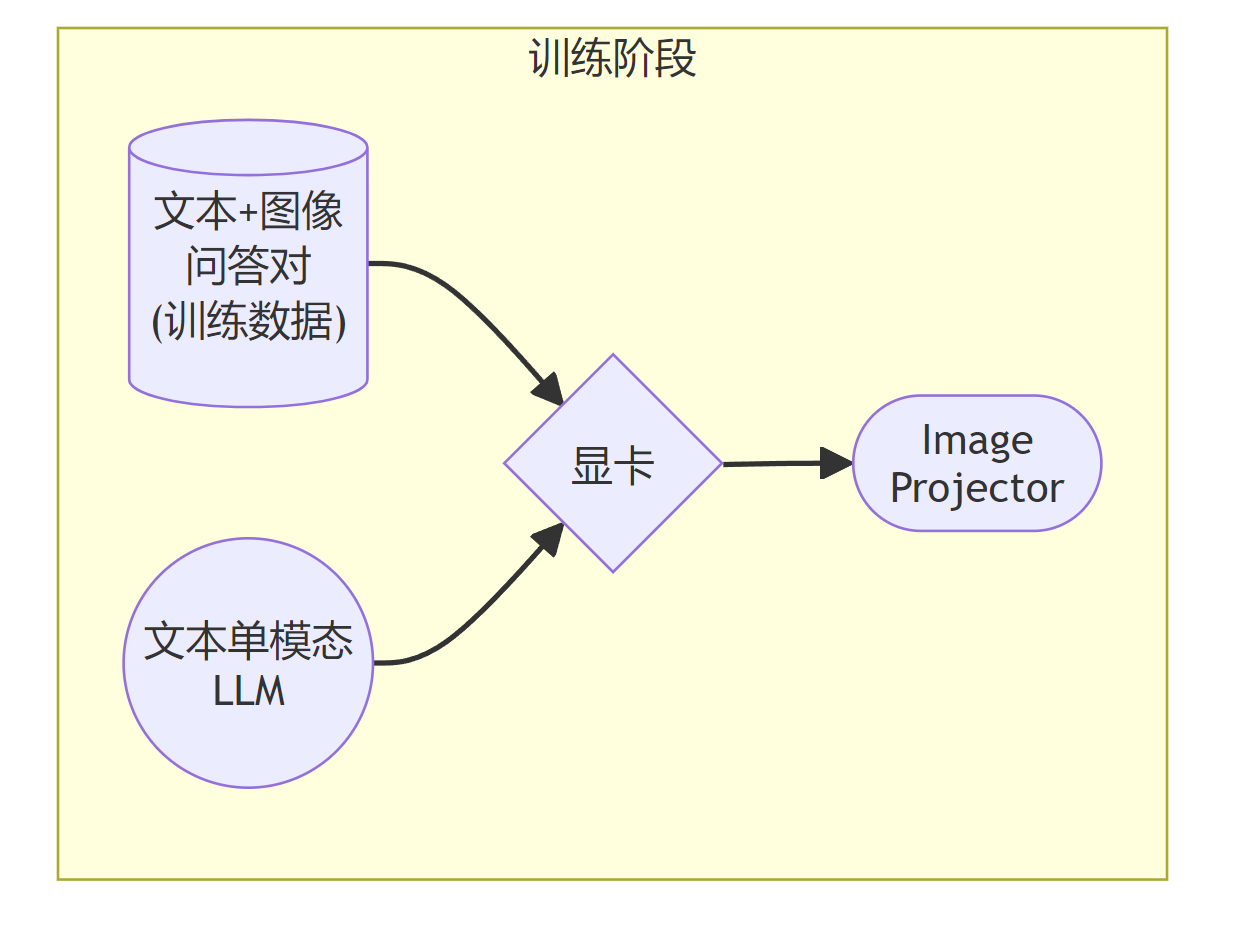


1.2. 什么型号的电子眼：LLaVA方案简介

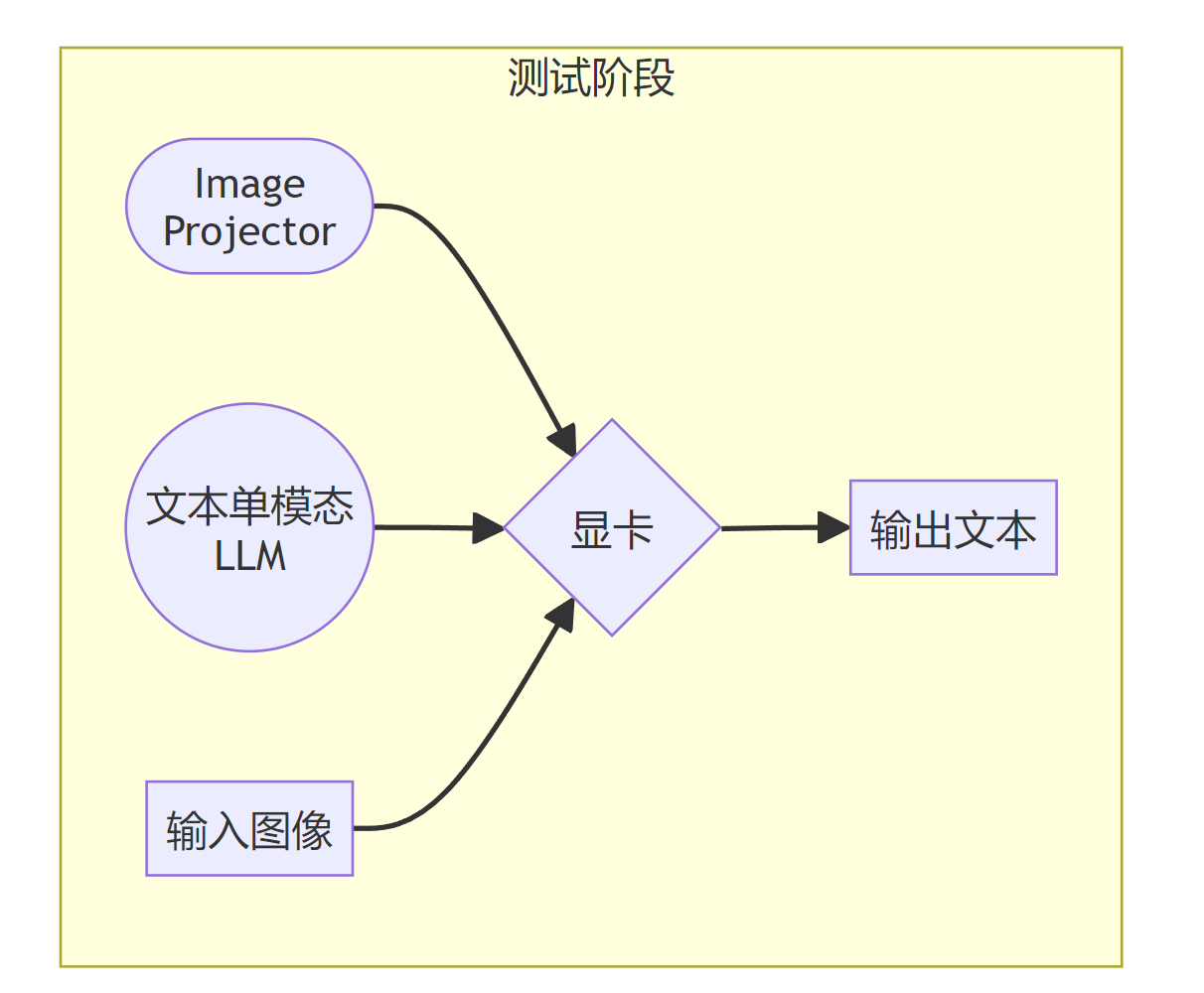
Haotian Liu等使用GPT-4V对图像数据生成描述，以此构建出大量<question text><image> -- <answer text>的数据对。利用这些数据对，配合文本单模态LLM，训练出一个Image Projector。

所使用的文本单模型LLM和训练出来的Image Projector，统称为LLaVA模型。

1.2.1. LLaVA训练阶段示意图



1.2.2. LLaVA测试阶段示意图



二者都是在已有LLM的基础上，用新的数据训练一个新的小文件。只不过，LLM套上LoRA之后，有了新的灵魂（角色）；而LLM套上Image Projector后，才有了眼睛。

1.3. 快速上手

1.3.1. 环境准备

# 如果你是在 InternStudio 平台，则从本地 clone 一个已有 pytorch 的环境：

# pytorch 2.0.1 py3.10\_cuda11.7\_cudnn8.5.0\_0

cd ~ && studio-conda xtuner0.1.17

# 如果你是在其他平台：

# conda create --name xtuner0.1.17 python=3.10 -y

# 激活环境

conda activate xtuner0.1.17

# 进入家目录 （~的意思是 “当前用户的home路径”）

cd ~

# 创建版本文件夹并进入，以跟随本教程

mkdir -p /root/xtuner0117 && cd /root/xtuner0117

# 拉取 0.1.17 的版本源码

git clone -b v0.1.17 https://github.com/InternLM/xtuner

# 无法访问github的用户请从 gitee 拉取:

# git clone -b v0.1.15 https://gitee.com/Internlm/xtuner

# 进入源码目录

cd /root/xtuner0117/xtuner

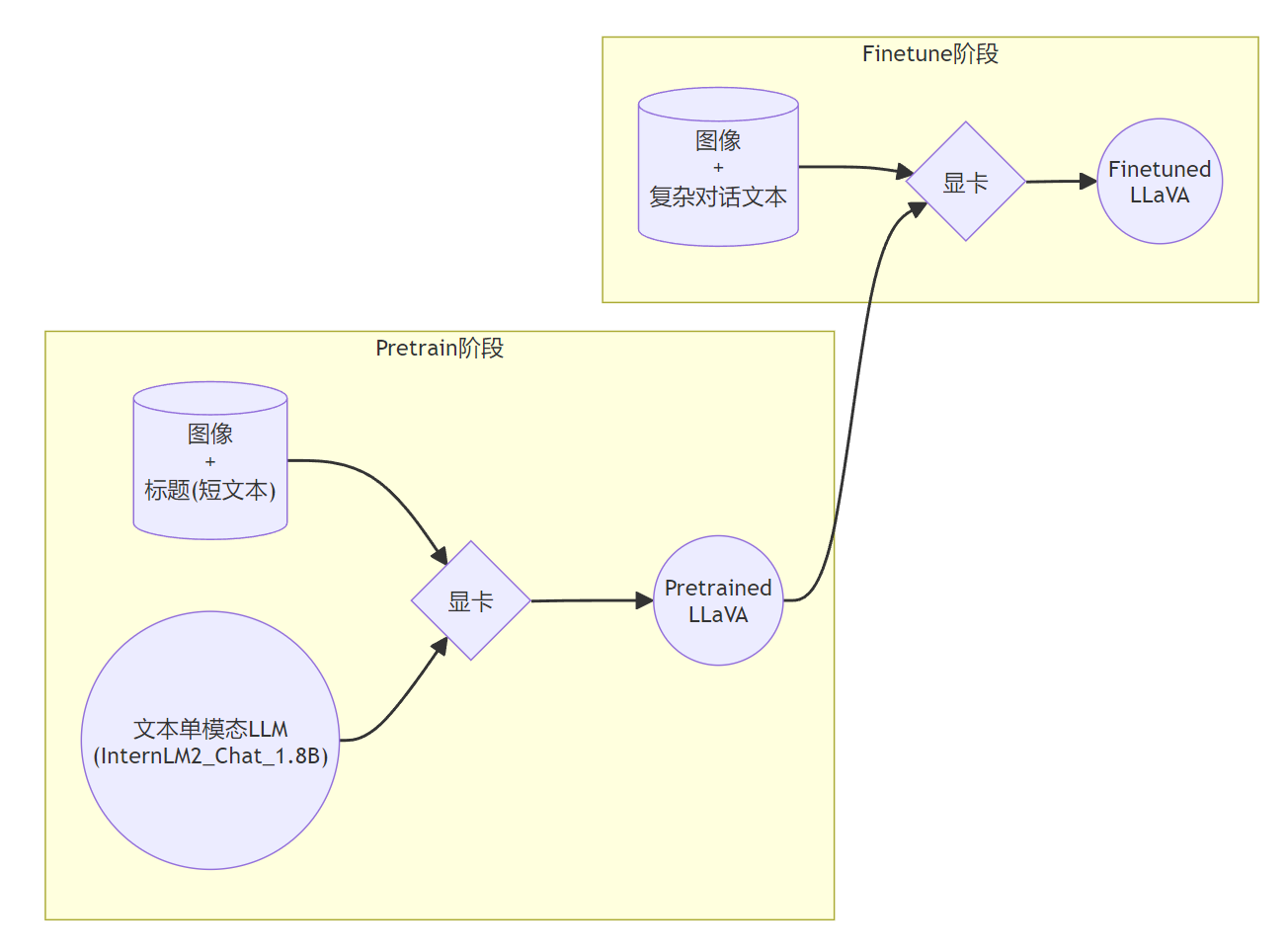
# 从源码安装 XTuner

pip install -e '.[all]' && cd ~

1.3.2. 概述

在本节中，我们将 自己构造 <question text><image>--<answer text> 数据对，基于InternLM2\_Chat\_1.8B这个文本单模态模型，使用LLaVA方案，训练一个给InternLM2\_Chat\_1.8B使用的Image Projector文件。

LLaVA方案中，给LLM增加视觉能力的过程，即是训练Image Projector文件的过程。 该过程分为2个阶段：Pretrain和Finetune。



1.3.3. Pretrain阶段

在Pretrain阶段，我们会使用大量的图片+简单文本（caption, 即图片标题）数据对，使LLM理解图像中的普遍特征。即，对大量的图片进行粗看。

Pretrain阶段训练完成后，此时的模型已经有视觉能力了！但是由于训练数据中都是图片+图片标题，所以此时的模型虽然有视觉能力，但无论用户问它什么，它都只会回答输入图片的标题。即，此时的模型只会给输入图像“写标题”。

Pretrain阶段相当于是开发LLM时预训练工作，对硬件要求非常高，有8卡的学有余力同学可以自行尝试。详见XTuner-LLaVA和LLaVA。

详情

在本次实战营中，我们已经为大家提供了Pretrain阶段的产物——iter\_2181.pth文件。它就是幼稚园阶段的Image Projector！大家带着iter\_2181.pth文件继续进入下一阶段进行Finetune即可。

1.3.4. Finetune阶段

在Finetune阶段，我们会使用图片+复杂文本数据对，来对Pretrain得到的Image Projector即iter\_2181.pth进行进一步的训练。

1.3.4.1. 训练数据构建

为了方便大家跟随课程，针对这张示例图片的问答对数据（repeat\_data.json），大家按照下面的脚本运行就可以生成啦~（重复200次）

cd ~ && git clone https://github.com/InternLM/tutorial -b camp2 && conda activate xtuner0.1.17 && cd tutorial

python /root/tutorial/xtuner/llava/llava\_data/repeat.py \

-i /root/tutorial/xtuner/llava/llava\_data/unique\_data.json \

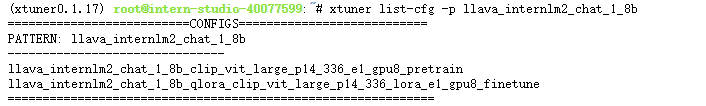
-o /root/tutorial/xtuner/llava/llava\_data/repeated\_data.json \

-n 200

1.3.4.2. 准备配置文件

# 查询xtuner内置配置文件

xtuner list-cfg -p llava\_internlm2\_chat\_1\_8b



# 拷贝配置文件到当前目录

xtuner copy-cfg \

llava\_internlm2\_chat\_1\_8b\_qlora\_clip\_vit\_large\_p14\_336\_lora\_e1\_gpu8\_finetune \

/root/tutorial/xtuner/llava

修改配置文件

# Model

- llm\_name\_or\_path = 'internlm/internlm2-chat-1\_8b'

+ llm\_name\_or\_path = '/root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b'

- visual\_encoder\_name\_or\_path = 'openai/clip-vit-large-patch14-336'

+ visual\_encoder\_name\_or\_path = '/root/share/new\_models/openai/clip-vit-large-patch14-336'

# Specify the pretrained pth

- pretrained\_pth = './work\_dirs/llava\_internlm2\_chat\_1\_8b\_clip\_vit\_large\_p14\_336\_e1\_gpu8\_pretrain/iter\_2181.pth' # noqa: E501

+ pretrained\_pth = '/root/share/new\_models/xtuner/iter\_2181.pth'

# Data

- data\_root = './data/llava\_data/'

+ data\_root = '/root/tutorial/xtuner/llava/llava\_data/'

- data\_path = data\_root + 'LLaVA-Instruct-150K/llava\_v1\_5\_mix665k.json'

+ data\_path = data\_root + 'repeated\_data.json'

- image\_folder = data\_root + 'llava\_images'

+ image\_folder = data\_root

# Scheduler & Optimizer

- batch\_size = 16 # per\_device

+ batch\_size = 1 # per\_device

# evaluation\_inputs

- evaluation\_inputs = ['请描述一下这张图片','Please describe this picture']

+ evaluation\_inputs = ['Please describe this picture','What is the equipment in the image?']

开始Finetune

cd /root/tutorial/xtuner/llava/

xtuner train /root/tutorial/xtuner/llava/llava\_internlm2\_chat\_1\_8b\_qlora\_clip\_vit\_large\_p14\_336\_lora\_e1\_gpu8\_finetune\_copy.py --deepspeed deepspeed\_zero2

1.3.5. 对比Finetune前后的性能差异

1.3.5.1. Finetune前

即：加载 1.8B 和 Pretrain阶段产物(iter\_2181) 到显存。

# 解决小bug

export MKL\_SERVICE\_FORCE\_INTEL=1

export MKL\_THREADING\_LAYER=GNU

# pth转huggingface

xtuner convert pth\_to\_hf \

llava\_internlm2\_chat\_1\_8b\_clip\_vit\_large\_p14\_336\_e1\_gpu8\_pretrain \

/root/share/new\_models/xtuner/iter\_2181.pth \

/root/tutorial/xtuner/llava/llava\_data/iter\_2181\_hf

# 启动！

xtuner chat /root/share/new\_models/Shanghai\_AI\_Laboratory/internlm2-chat-1\_8b \

--visual-encoder /root/share/new\_models/openai/clip-vit-large-patch14-336 \

--llava /root/tutorial/xtuner/llava/llava\_data/iter\_2181\_hf \

--prompt-template internlm2\_chat \

--image /root/tutorial/xtuner/llava/llava\_data/test\_img/oph.jpg

