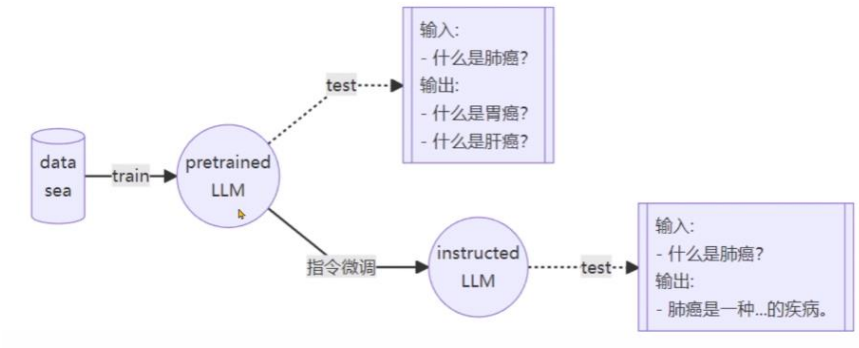
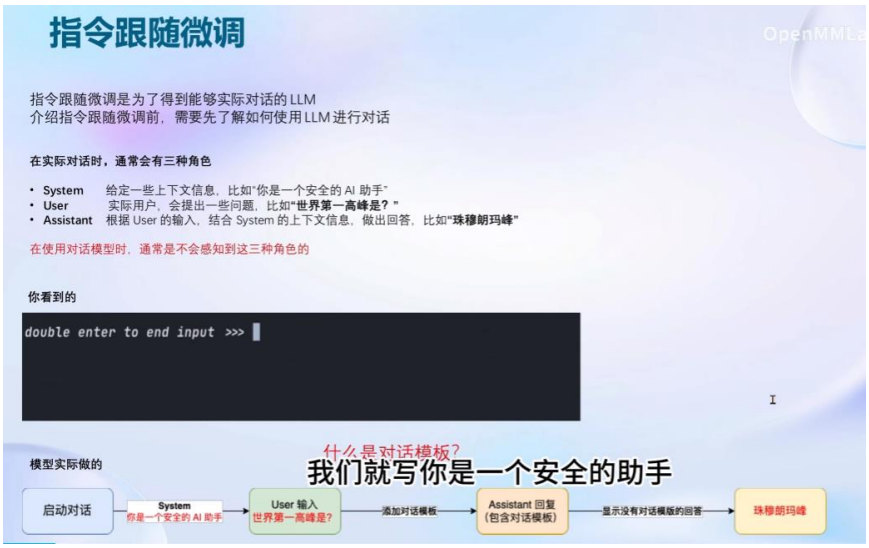
Lesson 4 笔记

XTuner 微调个人小助手认知

这节课分为 4 个部分，Finetune 简介、XTuner 介绍、8GB 显卡玩转 LLM 和动手实战环节。





指令跟随微调

对话模板

对话模板是为了能够让 LLM 区分出，System、User 和 Assistant 不同的模型会有不同的模板

LlaMa 2

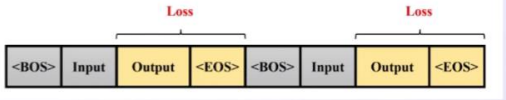
- <<SYS>> System 上下文开始
- <</SYS>> System 上下文结束
- [INST] User 指令开始
- [/INST] User 指令结束

InternLM

- <|System|>: System 上下文开始
- <|User|>: User 指令开始
- <eoh>: End of Human，User 指令结束
- <|Bot|>: Assistant 开始回答
- <eoa>: End of Assistant，Assistant 回答结束

Input ：世界第一高峰是?
Output: 珠穆朗玛峰

不同于增量预训练微调，数据中会有 Input 和 Output
希望模型学会的是答案(Output)，而不是问题(Input)
训练时只会对答案(Output)部分计算 Loss



训练时，会和推理时保持一致，对数据添加相应的对话模板，以下为 InternLM 的训练数据和标签

data	<s>	<|User|>:	世	界	第	一	高	峰	是	?	<eoh>	\n	<|Bot|>:	珠	穆	朗	玛	峰	<eoa>	</s>
label													珠	穆	朗	玛	峰	<eoa>	</s>	

增量预训练微调

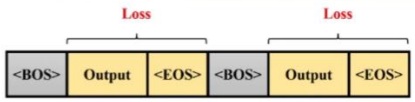
Output: 世界第一高峰是珠穆朗玛峰

```
"system": "",
"input": "",
"output": "I am an artificial intellig
```

为了让 LLM 知道什么时候开始一段话，什么时候结束一段话，
实际训练时需要对数据添加起始符（BOS）和结束符(EOS)；
大多数的模型都是使用 <s> 作为起始符，</s> 作为结束符

<s>世界第一高峰是珠穆朗玛峰</s>

训练 LLM 时，为了让模型学会“世界第一高峰是珠穆朗玛峰”，
并知道何时停止，对应的训练数据以及标签如下所示



data	<s>	世	界	第	一	高	峰	是	珠	穆	朗	玛	峰	</s>
label	世	界	第	一	高	峰	是	珠	穆	朗	玛	峰	</s>	

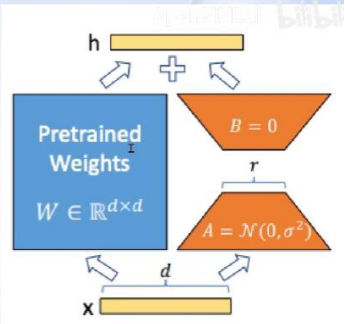
LoRA & QLoRA

LoRA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

LLM 的参数量主要集中在模型中的 Linear，训练这些参数会耗费大量的显存

LoRA 通过在原本的 Linear 旁，新增一个支路，包含两个连续的小 Linear，
新增的这个支路通常叫做 Adapter

Adapter 参数量远小于原本的 Linear，能大幅降低训练的显存消耗



微调原理

想象一下，你有一个超大的玩具，现在你想改造这个超大的玩具。但是，**对整个玩具进行全面的改动会非常昂贵**。

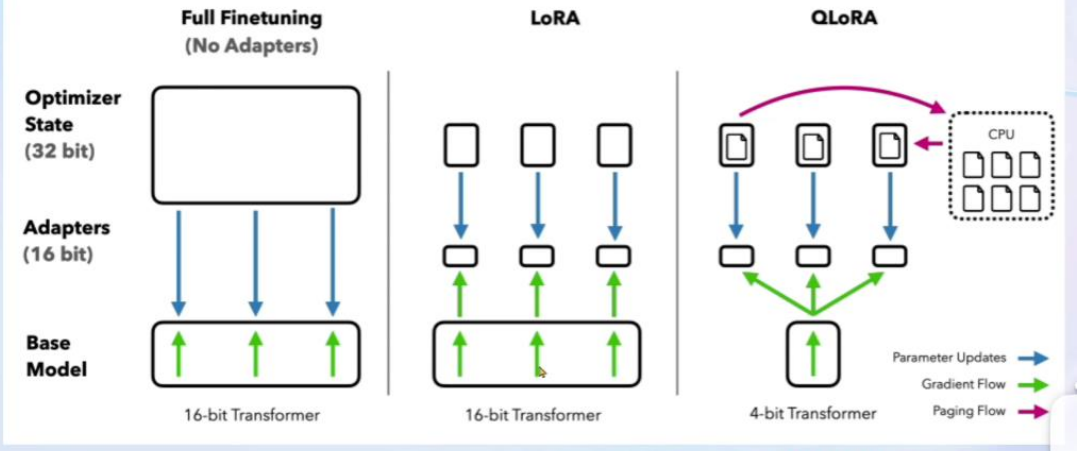
※ 因此，你找到了一种叫 **LoRA** 的方法：**只对玩具中的某些零件进行改动，而不是对整个玩具进行全面改动。**

※ 而 **QLoRA** 是 LoRA 的一种改进：如果你手里只有一把生锈的螺丝刀，也能改造你的玩具。

- Full: 😊 → 🚲
- LoRA: 😊 → 🚲
- QLoRA: 😊 → 🚲

如果我们要对整个模型的所有参数

LoRA & QLoRA



- Base Model 参与训练并更新参数
- 需要保存 Base Model 中参数的优化器状态

- Base Model 只参与 Forward
- 只有 Adapter 部分 Backward 更新参数
- 只需保存 Adapter 中参数的优化器状态

- Base Model 量化为 4-bit
- 优化器状态在 CPU 与 GPU 间 Offload
- Base Model 只参与 Forward
- 只有 Adapter 部分 Backward 更新参数
- 只需保存 Adapter 中参数的优化器状态

XTuner 简介

功能亮点

适配多种生态

- 多种微调算法

  多种微调策略与算法，覆盖各类 SFT 场景

- 适配多种开源生态

  支持加载 HuggingFace、ModelScope 模型或数据集

- 自动优化加速

  开发者无需关注复杂的显存优化与计算加速细节

适配多种硬件

- 训练方案覆盖 NVIDIA 20 系以上所有显卡
- 最低只需 8GB 显存即可微调 7B 模型

XTuner 技术架构图

XTuner 快速上手

1. 安装

```
pip install xtuner
```

2. 挑选配置模板

```
xtuner list-cfg -p internlm_20b
```

3. 一键训练

```
xtuner train internlm_20b_qlora_oasst1_512_e3
```

Config 命名规则

模型名	internlm_20b　无 chat 代表是基座模型
使用算法	qlora
数据集	oasst1
数据长度	512
Epoch	e3，epoch 3

对话

为了便于开发者查看训练效果，Xtuner 提供了一键对话接口

Float 16 模型对话

```
xtuner chat internlm/internlm-chat-20b
```

4bit 模型对话

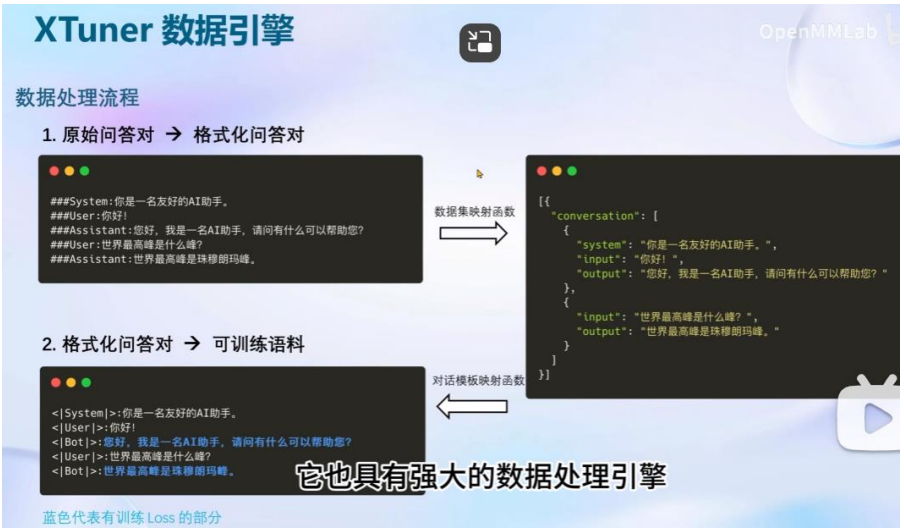
```
xtuner chat internlm/internlm-chat-20b --bits 4
```

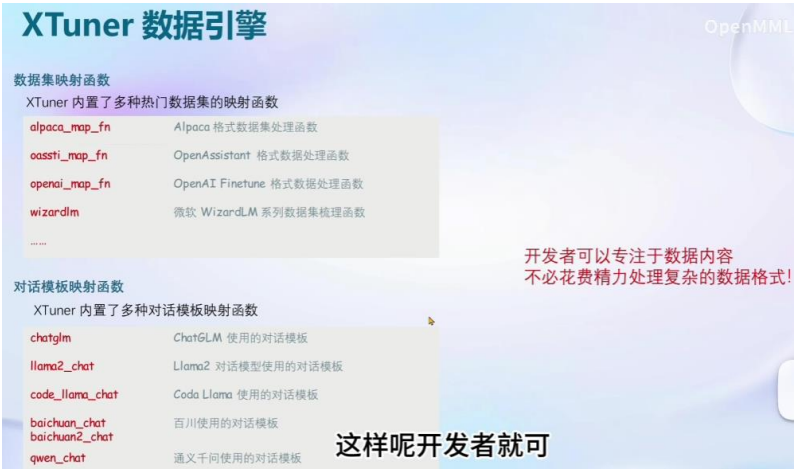
加载 Adapter 模型对话

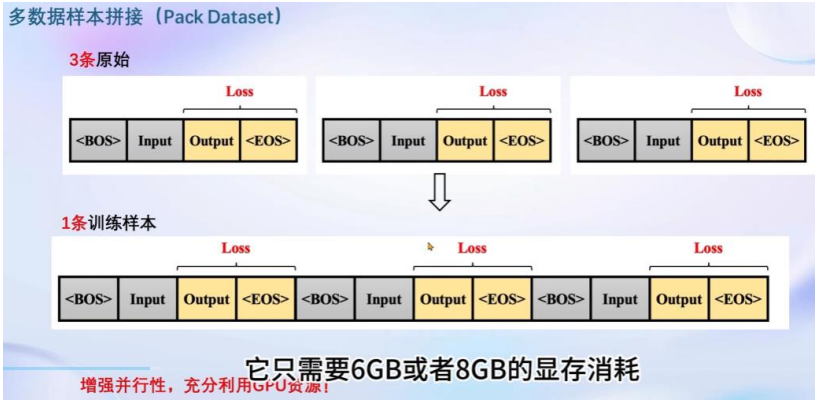
```
xtuner chat internlm/internlm-chat-20b --adapater $ADAPTER_DIR
```

```
double enter to end input >>>
```

XTuner 还支持工具类模型的对话，更多详见 HuggingFace Hub（xtuner/Llama-2-7b-qlora-moss-003-sft）

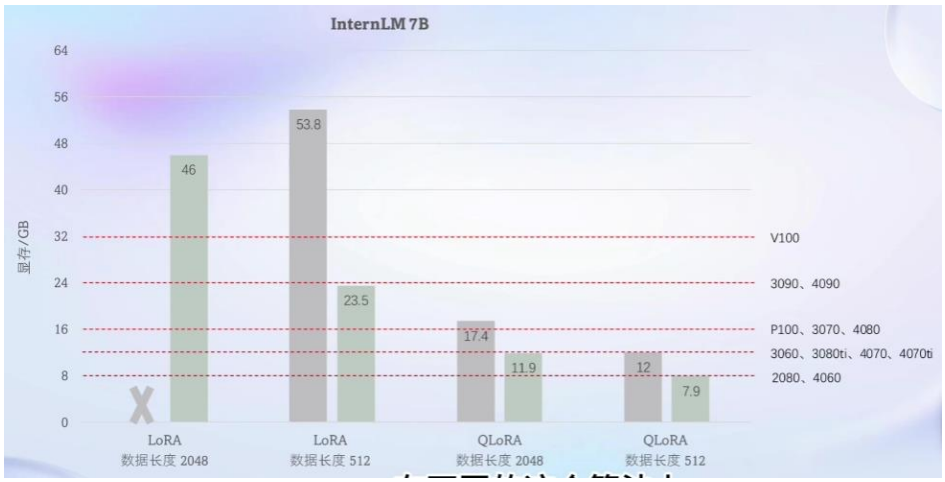






DeepSpeed 和 Flash Attention 虽然能够大幅降低训练成本，但使用门槛相对较高，需要复杂的配置，甚至修改代码。

为了让开发者专注于数据，XTuner 会自动 dispatch Flash Attention,并一键启动 DeepSpeed ZeRO。



利用 XTuner 完成个人小助手的微调步骤：

1　开发机准备，打开 Intern Studio 界面，点击 创建开发机 配置开发机系统。

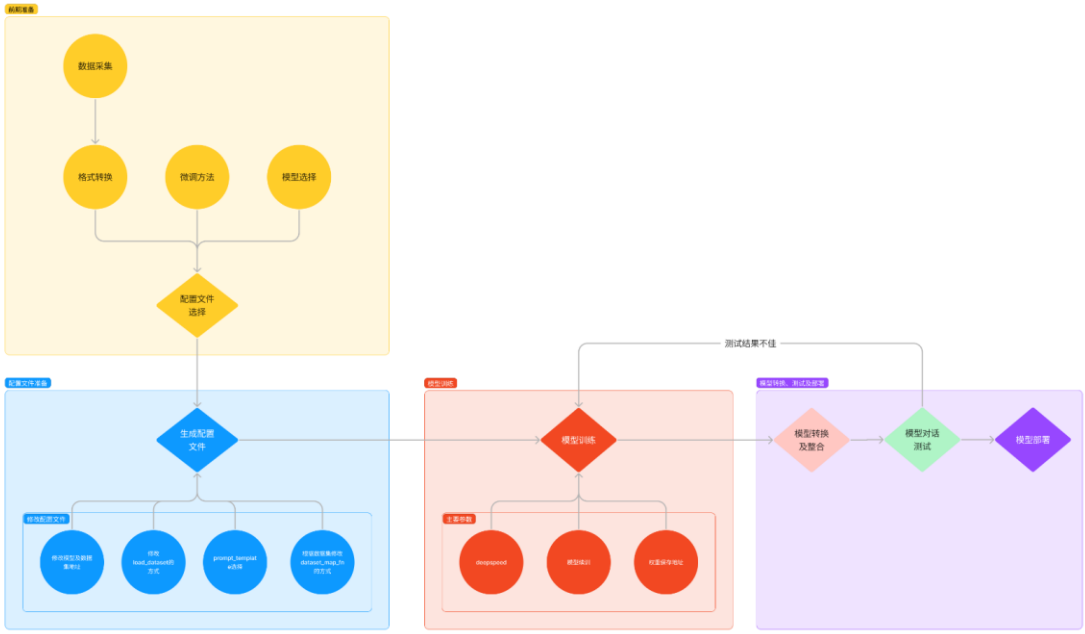
填写 开发机名称 后，点击 选择镜像 使用 Cuda11.7-conda 镜像，然后在资源配置中，使用 10% A100 * 1 的选项，然后立即创建开发机器。

进入开发机，点击 Terminal 进入终端界面即可开始操作！

```
(base) root@intern-studio-085887:/opt/jupyterlab# studio-conda xtuner0.1.17
```

2　快速上手

XTuner 的运行原理，如图：



首先，需要在本地安装 XTuner 微调工具包，这是使用该工具的基础前提。安装完毕后，应明确微调目标，考虑所需的数据资源和硬件资源。对于资源有限的开发者，还需要思考数据采集和提升模型效果的方法。随后，在 XTuner 的配置库中选择并修改合适的配置文件，以启动模型的训练。训练完成的模型可以通过简单的终端指令进行转换和部署。

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]'
```

假如速度太慢可以 Ctrl + C 退出后换成 pip install -e '.[all]' -i https://mirrors.aliyun.com/pypi/simple/

如果整个安装过程中没有遇到任何错误，这意味着 XTuner 运行所需的环境已经成功安装。这对于许多初学者来说，已经是成功的一大步。因此，下一步就是准备所需的数据集、模型和配置文件，为微调工作做好准备。

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

创建并运行一个名为 generate_data.py 的脚本于 data 目录下，以生成所需数据集。如需定制数据集以更好反映你的身份，可增大脚本中的 n 值。

创建 `generate_data.py` 文件

touch /root/ft/data/generate_data.py

打开该 python 文件后将下面的内容复制进去。

```
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)
```

并将文件 name 后面的内容修改为你的名称。比如说我是剑锋大佬的话就是:

```
# 将对应的 name 进行修改（在第 4 行的位置）
- name = '不要姜葱蒜大佬'
+ name = "剑锋大佬"
```

修改完成后运行 generate_data.py 文件即可。

```
# 确保先进入该文件夹
cd /root/ft/data
# 运行代码
python /root/ft/data/generate_data.py
```

在 data 目录下成功生成了一个名为 personal_assistant.json 的文件，里面包含了 5000 条 input-output 数据对，为微调准备了数据集。如果觉得 5000 条数据不足以满足需求，可以通过调整文件中第 6 行的 n 值来增加数据量。

```
|-- data/
    |-- personal_assistant.json
    |-- generate_data.py
```

文件结构树代码

文件结构树代码如下所示，使用方法为在终端调用该代码的同时在后方输入文件夹路径。

比如说我要打印 data 的文件结构树，假设我的代码文件保存在 /root/tree.py ，那我就要在终端输入 python /root/tree.py /root/ft/data

```
import os
import argparse
def print_dir_tree(startpath, prefix=''):
    """递归地打印目录树结构。"""
    contents = [os.path.join(startpath, d) for d in os.listdir(startpath)]
    directories = [d for d in contents if os.path.isdir(d)]
    files = [f for f in contents if os.path.isfile(f)]
    if files:
        for f in files:
            print(prefix + '|-- ' + os.path.basename(f))
    if directories:
        for d in directories:
            print(prefix + '|-- ' + os.path.basename(d) + '/')
            print_dir_tree(d, prefix=prefix + '    ')
def main():
    parser = argparse.ArgumentParser(description='打印目录树结构')
    parser.add_argument('folder', type=str, help='要打印的文件夹路径')
    args = parser.parse_args()
    print('|-- ' + os.path.basename(args.folder) + '/')
    print_dir_tree(args.folder, '    ')
if __name__ == "__main__":
    main()
```

除了我们自己通过脚本的数据集，其实网上也有大量的开源数据集可以供我们进行使用。有些时候我们可以在开源数据集的基础上添加一些我们自己独有的数据集，也可能会有很好的效果。

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 InterLM-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

存储空间不足的话，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

使用 XTuner 的 list-cfg 工具可以列出所有可用的配置文件。不加任何参数运行时，它会显示全部配置文件。另外，使用 -p 或 --pattern 参数可以对配置文件进行模糊匹配搜索，这对于寻找特定模型如 internlm2_1_8b 或特定微调方法如 qlora 的配置文件非常有用。例如，在搜索 internlm2-1.8B 时，指令会显示当前只有两个适用的模型配置文件。

==========CONFIGS==========

PATTERN: internlm2_1_8b

----

internlm2_1_8b_full_alpaca_e3

配置文件名的解释

以 internlm2_1_8b_qlora_alpaca_e3 举例:

模型名	说明
internlm2_1_8b	模型名称
qlora	使用的算法
alpaca	数据集名称
e3	把数据集跑3次

虽然用的数据集并不是 alpaca 而是自己通过脚本制作的小助手数据集，但是由于是通过 QLoRA 的方式对 internlm-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

在 XTuner 工具箱中，copy-cfg 是用来复制配置文件的第二个工具。它需要两个必填参数：{CONFIG_NAME} 和 {SAVE_PATH}。在你的例子中，{CONFIG_NAME} 是之前搜索到的 internlm2_1_8b_qlora_alpaca_e3，而 {SAVE_PATH} 是你新建的路径 /root/ft/config。如果想复制其他配置文件，只需更改这两个参数即可。执行这个命令后，你会在 /root/ft/config 目录下找到一个名为 internlm2_1_8b_qlora_alpaca_e3_copy.py 的文件。

|-- config/

|-- internlm2_1_8b_qlora_alpaca_e3_copy.py

2.2.4 小结

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。

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

最后在 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，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
```

确实，微调过程可能看起来不那么复杂，但关键在于准备一份高质量的数据集，这是决定微调成功与否的核心。微调有时被幽默地比喻为“炼丹”，强调在微调（炼丹）过程中考虑的各个因素：使用的数据（材料）、调整的参数（火候）、训练时长（烤时间）以及所用的微调工具（丹炉，比如 XTuner）。如果数据集（材料）本身质量不佳，那

么无论如何调整，最终的结果也不会满意。高质量的数据是成功的关键。

如果你想深入了解如何制作数据集，可以考虑加入一些专业社群或小组，比如书生.浦语的 RolePlay SIG，那里有经验丰富的人士可以提供指导，教你如何为喜欢的角色制作个性化的数据集。这样的社群还可以让你听到更多关于数据集制作的想法和经验分享。

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，下面要做的事情就是根据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

打开配置文件后，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。

在微调时，通常需要调整的配置文件内容主要集中在前三部分，这涉及到模型选择和数据集配置的修改。而后两部分，通常由 XTuner 官方优化，一般不需要用户进行改动，除非进行一些特殊的定制化调整。根据你的项目需求，我们可以逐步进行必要的修改和调整来达到最佳的微调效果。

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）。

参数修改细节

首先在 PART 1 的部分，由于不再需要在 Huggingface 上自动下载模型，因此先要更换模型的路径以及数据集的路径为本地的路径。

修改模型地址（在第 27 行的位置）

- pretrained_model_name_or_path = 'internlm/internlm2-1_8b'

+ pretrained_model_name_or_path = '/root/ft/model'

修改数据集地址为本地的 json 文件地址（在第 31 行的位置）

- alpaca_en_path = 'tatsu-lab/alpaca'

+ alpaca_en_path = '/root/ft/data/personal_assistant.json'

除此之外，还可以对一些重要的参数进行调整，包括学习率（lr）、训练的轮数（max_epochs）等等。由于这次只是一个简单的让模型知道自己的身份弟位，因此我们的训练轮数以及单条数据最大的 Token 数（max_length）都可以不用那么大。

修改 max_length 来降低显存的消耗（在第 33 行的位置）

- max_length = 2048

+ max_length = 102

减少训练的轮数（在第 44 行的位置）

- max_epochs = 3

+ max_epochs = 2

增加保存权重文件的总数（在第 54 行的位置）

- save_total_limit = 2

+ save_total_limit = 3

修改 max_length 来降低显存的消耗（在第 33 行的位置）

- max_length = 2048

+ max_length = 1024

减少训练的轮数（在第 44 行的位置）

- max_epochs = 3

+ max_epochs = 2

增加保存权重文件的总数（在第 54 行的位置）

- save_total_limit = 2

+ save_total_limit = 3

XTuner 提供了一个名为 evaluation_inputs 的参数，允许你设置多个检测问题，以便在训练过程中实时监控模型表现的变化，确保模型正朝着你期望的方向发展。例如，

如果你希望模型在被问到“请你介绍一下你自己”或“你是谁”时能够以“我是 XXX 的小助手...”的形式回应，就可以根据这一需求来设置 evaluation_inputs 参数，进而调整模型以满足特定的回答风格或内容。这样的设置有助于你更精准地微调模型，以达到预期的效果。

修改每多少轮进行一次评估（在第 57 行的位置）

- evaluation_freq = 500

+ evaluation_freq = 300

修改具体评估的问题（在第 59 到 61 行的位置）

可以自由拓展其他问题

- evaluation_inputs = ['请给我介绍五个上海的景点', 'Please tell me five scenic spots in Shanghai']

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

这样修改完后在评估过程中就会显示在当前的权重文件下模型对这几个问题的回复了。

由于我们的数据集不再是原本的 aplaca 数据集，因此我们也要进入 PART 3 的部分对相关的内容进行修改。包括说我们数据集输入的不是一个文件夹而是一个单纯的 json 文件以及我们的数据集格式要求改为我们最通用的 OpenAI 数据集格式。

把 OpenAI 格式的 map_fn 载入进来（在第 15 行的位置）

- from xtuner.dataset.map_fns import alpaca_map_fn, template_map_fn_factory

+ from xtuner.dataset.map_fns import openai_map_fn, template_map_fn_factory

将原本是 alpaca 的地址改为是 json 文件的地址（在第 102 行的位置）

- dataset=dict(type=load_dataset, path=alpaca_en_path),

+ dataset=dict(type=load_dataset, path='json', data_files=dict(train=alpaca_en_path)),

将 dataset_map_fn 改为通用的 OpenAI 数据集格式（在第 105 行的位置）

- dataset_map_fn=alpaca_map_fn,

+ dataset_map_fn=openai_map_fn,

把 OpenAI 格式的 map_fn 载入进来（在第 15 行的位置）

- from xtuner.dataset.map_fns import alpaca_map_fn, template_map_fn_factory

+ from xtuner.dataset.map_fns import openai_map_fn, template_map_fn_factory

将原本是 alpaca 的地址改为是 json 文件的地址（在第 102 行的位置）

- dataset=dict(type=load_dataset, path=alpaca_en_path),

+ dataset=dict(type=load_dataset, path='json', data_files=dict(train=alpaca_en_path)),

将 dataset_map_fn 改为通用的 OpenAI 数据集格式（在第 105 行的位置）

- dataset_map_fn=alpaca_map_fn,

+ dataset_map_fn=openai_map_fn,

常用参数介绍

参数名	解释
data_path	数据路径或 HuggingFace 仓库名
max_length	单条数据最大 Token 数，超过则截断
pack_to_max_length	是否将多条短数据拼接到 max_length，提高 GPU 利用率
accumulative_counts	梯度累积，每多少次 backward 更新一次参数
sequence_parallel_size	并行序列处理的大小，用于模型训练时的序列并行
batch_size	每个设备上的批量大小
dataloader_num_workers	数据加载器中工作进程的数量
max_epochs	训练的最大轮数
optim_type	优化器类型，例如 AdamW
lr	学习率
betas	优化器中的 beta 参数，控制动量和平方梯度的移动平均
weight_decay	权重衰减系数，用于正则化和避免过拟合
max_norm	梯度裁剪的最大范数，用于防止梯度爆炸
warmup_ratio	预热的比例，学习率在这个比例的训练过程中线性增加到初始学习率
save_steps	保存模型的步数间隔
save_total_limit	保存的模型总数限制，超过限制时删除旧的模型文件
prompt_template	模板提示，用于定义生成文本的格式或结构
.....	.....

如果想把显卡的现存吃满，充分利用显卡资源，可以将 max_length 和 batch_size 这两个参数调大。

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)
```

这一节我们讲述了微调过程中一些常见的需要调整的内容，包括各种的路径、超参数、评估问题等等。完成了这部分的修改后，就可以正式的开始我们下一阶段的旅程： XTuner 启动~!

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

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。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。

300 轮

```
# 300 轮

<|User|>:请你介绍一下你自己
<|Bot|>:我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦</s>

<|User|>:你是谁
<|Bot|>:我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦</s>

<|User|>:你是我的小助手吗
<|Bot|>:是的</s>

# 600 轮

<|User|>:请你介绍一下你自己
<|Bot|>:我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦</s>

<|User|>:你是谁
<|Bot|>:我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦</s>

<|User|>:你是我的小助手吗
<|Bot|>:我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦</s>
```

通过两者的对比我们其实就可以很清楚的看到，在 300 轮的时候模型已经学会了在我问“你是谁”或者说“请你介绍一下我自己”的时候回答“我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦”。

但是两者的不同是在询问“你是我的小助手”的这个问题上，300 轮的时候是回答正确的，回答了“是”，但是在 600 轮的时候回答的还是“我是剑锋大佬的小助手，内在是上海 AI 实验室书生·浦语的 1.8B 大模型哦”这一段话。这表明模型在第一批次第 600 轮的时候已经出现严重的过拟合（即模型丢失了基础的能力，只会成为某一句话的复读机）现象了，到后面的话无论我们再问什么，得到的结果也就只能是回答这一句话了，模型已经不会再说别的话了。因此假如以通用能力的角度选择最合适的权重文件的话我们可能会选择前面的权重文件进行后续的模型转化及整合工作。

假如我们想要解决这个问题，其实可以通过以下两个方式解决：

- 减少保存权重文件的间隔并增加权重文件保存的上限：这个方法实际上就是通过降低间隔结合评估问题的结果，从而找到最优的权重文。我们可以每隔 100 个批次来看什么时候模型已经学到了这部分知识但是还保留着基本的常识，什么时候已经过拟合严重只会说一句话了。但是由于再配置文件有设置权重文件保存数量

的上限，因此同时将这个上限加大也是非常必要的。

- 增加常规的对话数据集从而稀释原本数据的占比：这个方法其实就是希望我们正常用对话数据集做指令微调的同时还加上一部分的数据集来让模型既能够学到正常对话，但是在遇到特定问题时进行特殊化处理。比如说我在一万条正常的对话数据里混入两千条和小助手相关的数据集，这样模型同样可以在不丢失对话能力的前提下学到剑锋大佬的小助手这句话。这种其实是比较常见的处理方式，大家可以自己动手尝试实践一下。

另外假如我们模型中途中断了，也可以参考以下方法实现模型续训工作。

模型续训指南

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

2.4.4 小结

在本节重点是讲解模型训练过程中的种种细节内容，包括了模型训练中的各个参数以、权重文件的选择方式以及模型续训的方法。可以看到是否使用 --work-dir 和 是否使用 --deepspeed 会对文件的保存位置以及权重文件的保存方式有所不同，在训练完成后，就可以把训练得到的 .pth 文件进行下一步的转换和整合工作了!

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

转换完成后，可以看到模型被转换为 Huggingface 中常用的 .bin 格式文件，这就代表着文件成功被转化为 Huggingface 格式了。

|-- huggingface/

|-- adapter_config.json

|-- xtuner_config.py

|-- adapter_model.bin

|-- README.md

此时，huggingface 文件夹即为我们平时所理解的所谓“LoRA 模型文件”

可以简单理解：LoRA 模型文件 = Adapter

除此之外，我们其实还可以在转换的指令中添加几个额外的参数，包括以下两个:

参数名	解释
--fp32	代表以fp32的精度开启，假如不输入则默认为fp16
--max-shard-size {GB}	代表每个权重文件最大的大小（默认为2GB）

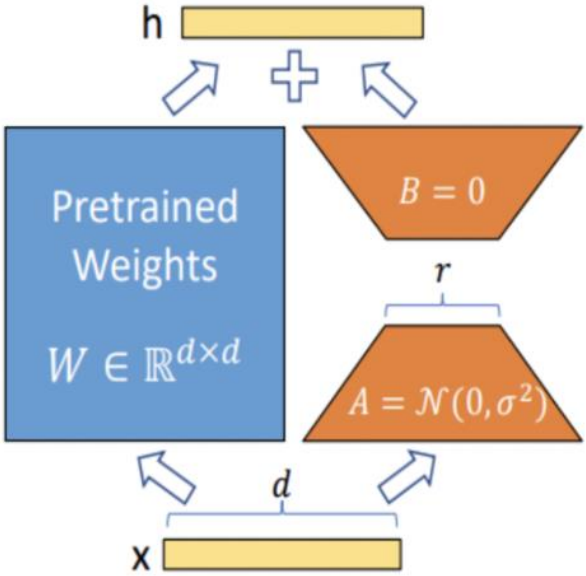
假如有特定的需要，我们可以在上面的转换指令后进行添加。由于本次测试的模型文件较小，并且已经验证过拟合，故没有添加。假如加上的话应该是这样的:

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 --fp32 --max-shard-size 2GB

2.5.2 模型整合

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
```

那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

可以通过一些简单的测试来看看微调后的模型的能力。

假如想要输入内容需要在输入文字后敲击两下回车，假如我们想清楚历史记录需要输入 RESET，假如我们想要退出则需要输入 EXIT。

同样的我们也可以和原模型进行对话进行对比

xtuner chat /root/ft/model --prompt-template internlm2_chat

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
```

将 /root/ft/web_demo/InternLM/chat/web_demo.py 中的内容替换为以下的代码（与源代码相比，此处修改了模型路径和分词器路径，并且也删除了 avatar 及 system_prompt 部分的内容，同时与 cli 中的超参数进行了对齐）。

```
"""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 = 32768
    top_p: float = 0.8
    temperature: float = 0.8
    do_sample: bool = True
    repetition_penalty: float = 1.005
@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()
```

在运行前，还需要做的就是将端口映射到本地。使用快捷键组合 Windows + R 打开指令界面，并输入命令，按下回车键。先查询端口，再根据端口键入命令。需要在 PowerShell 中输入以下内容（需要替换为自己的端口号）。

```
ssh -CNg -L 6006:127.0.0.1:6006 root@ssh.intern-ai.org.cn -p 38374
```

再复制下方的密码，输入到 password 中，直接回车。

之后我们需要输入以下命令运行 /root/personal_assistant/code/InternLM 目录下的 web_demo.py 文件。

```
streamlit run /root/ft/web_demo/InternLM/chat/web_demo.py --server.address 127.0.0.1 --server.port 6006
```

注意：要在浏览器打开 http://127.0.0.1:6006 页面后，模型才会加载。

打开 http://127.0.0.1:6006 后，等待加载完成即可进行对话。

假如我们还想和原来的 InternLM2-Chat-1.8B 模型对话（即在 /root/ft/model 这里的模型对话），我们其实只需要修改 183 行和 186 行的文件地址即可。

```
# 修改模型地址（第 183 行）
- model = (AutoModelForCausalLM.from_pretrained('/root/ft/final_model',
+ model = (AutoModelForCausalLM.from_pretrained('/root/ft/model',
# 修改分词器地址（第 186 行）
- tokenizer = AutoTokenizer.from_pretrained('/root/ft/final_model',
```

+ tokenizer = AutoTokenizer.from_pretrained('/root/ft/model',

然后使用上方同样的命令即可运行。

streamlit run /root/ft/web_demo/InternLM/chat/web_demo.py --server.address 127.0.0.1 --server.port 6006

加载完成后输入同样的问题。

2.5.5 小结

本节介绍了如何将微调后的模型（adapter）转换和集成，并通过 xtuner chat 命令测试了模型的实际对话表现。测试结果明显显示出模型回复的变化。确认模型满足需求后，接下来可进行模型的量化和部署。

2.6 总结

本节内容主要引导大家完成了一个完整的 XTuner 微调流程，包括了解数据集和模型的使用、配置文件的创建与调整、训练过程，以及最终的模型转换和整合。

用 MS-Agent 数据集 赋予 LLM 以 Agent 能力

MSAgent 数据集每条样本包含一个对话列表（conversations），其里面包含了 system、user、assistant 三种字段。其中：

- system: 表示给模型前置的人设输入，其中有告诉模型如何调用插件以及生成请求
- user: 表示用户的输入 prompt，分为两种，通用生成的 prompt 和调用插件需求的 prompt
- assistant: 为模型的回复。其中会包括插件调用代码和执行代码，调用代码是要 LLM 生成的，而执行代码是调用服务来生成结果的

xtuner 是从国内的 ModelScope 平台下载 MS-Agent 数据集，因此不用提前手动下载数据集文件。

准备工作

mkdir ~/ft-msagent && cd ~/ft-msagent

cp -r ~/ft-oasst1/internlm-chat-7b .

查看配置文件

xtuner list-cfg | grep msagent

复制配置文件到当前目录

xtuner copy-cfg internlm_7b_qlora_msagent_react_e3_gpu8 .

修改配置文件中的模型为本地路径

vim ./internlm_7b_qlora_msagent_react_e3_gpu8_copy.py

- pretrained_model_name_or_path = 'internlm/internlm-chat-7b'

+ pretrained_model_name_or_path = './internlm-chat-7b'

xtuner train ./internlm_7b_qlora_msagent_react_e3_gpu8_copy.py --deepspeed deepspeed_zero2

下载 Adapter

cd ~/ft-msagent

apt install git git-lfs

git lfs install

git lfs clone https://www.modelscope.cn/xtuner/internlm-7b-qlora-msagent-react.git

添加 serper 环境变量：export SERPER_API_KEY=abcdefg

xtuner + agent，启动!

xtuner chat ./internlm-chat-7b --adapter internlm-7b-qlora-msagent-react –lagent

如何在 OpenXLab 部署 streamlit 应用：
https://aicarrier.feishu.cn/docx/MQH6dygcKolG37x0ekcc4oZhnCe