使用Qwen-chat-7b微调自己的数据集详细教程

1、使用Xtuner大语言模型微调工具箱

教程参考https://github.com/InternLM/tutorial/blob/main/helloworld/hello_world.md

B站教程https://www.bilibili.com/video/BV1Ci4y1z72H/?spm_id_from=333.788&vd_source=f7f2cbaddde8e0387df87ac4c5ceaf52

2、我的平台和环境

Ubuntu20.04 + CUDA11.8 + RTX 4090(24G) + Python3.8 + Pytorch2.0

3、安装xtuner v0.1.9

```
# 创建版本文件夹并进入，以跟随本教程
mkdir xtuner019 && cd xtuner019

# 拉取 0.1.9 的版本源码
git clone -b v0.1.9  https://github.com/InternLM/xtuner
# 无法访问github的用户请从 gitee 拉取:# git clone -b v0.1.9 https://gitee.com/Internlm/xtuner# 进入源码目录cd xtuner

# 从源码安装 XTuner
pip install -e '.[all]'

# 创建一个微调自定义数据集的工作路径，进入
mkdir ~/finetune && cd ~/finetune
```

4、微调工作准备

1、准备配置文件

```
# 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
xtuner list-cfg
```

```
==========CONFIGS==========
PATTERN: internlm_20b
------------------------------
internlm_20b_chat_qlora_alpaca_e3
internlm_20b_chat_qlora_alpaca_enzh_e3
internlm_20b_chat_qlora_alpaca_enzh_oasst1_e3
internlm_20b_chat_qlora_alpaca_zh_e3
internlm_20b_chat_qlora_code_alpaca_e3
internlm_20b_chat_qlora_lawyer_e3
internlm_20b_chat_qlora_oasst1_512_e3
internlm_20b_chat_qlora_oasst1_e3
internlm_20b_chat_qlora_open_platypus_e3
internlm_20b_qlora_alpaca_e3
internlm_20b_qlora_alpaca_enzh_e3
internlm_20b_qlora_alpaca_enzh_oasst1_e3
internlm_20b_qlora_alpaca_zh_e3
internlm_20b_qlora_arxiv_gentitle_e3
internlm_20b_qlora_code_alpaca_e3
internlm_20b_qlora_colorist_e5
internlm_20b_qlora_lawyer_e3
internlm_20b_qlora_oasst1_512_e3
internlm_20b_qlora_oasst1_e3
internlm_20b_qlora_open_platypus_e3
internlm_20b_qlora_sql_e3
===========================
```

```
cd ~/finetune
xtuner copy-cfg qwen_7b_chat_qlssora_oasst1_e3 . #会下载一个qwen_7b_chat_qlora_oasst1_e3.py配置文件
```

配置文件名的解释：

> xtuner copy-cfg internlm_chat_7b_qlora_oasst1_e3 .

模型名	internlm_chat_7b
使用算法	qlora
数据集	oasst1
把数据集跑几次	跑3次：e3 (epoch 3)

*无 chat比如 `internlm-7b` 代表是基座(base)模型

2、模型下载

```
# 创建一个目录，放模型文件，防止散落一地
mkdir ~/finetune/qwen-chat-7b

# 装一下拉取模型文件要用的库
pip install modelscope
# 从 modelscope 下载下载模型文件
cd ~/finetune/qwen-chat-7b
apt install git git-lfs -y     ###sudo apt-get update → sudo apt-get git git-lfs -y
git lfs install
```

```
git clone https://www.modelscope.cn/qwen/Qwen-7B-Chat.git
```

3、数据集导入

```
cd ~/finetune
mkdir data
#将要训练的自定义json数据通过本地上传导入
```

json数据集制作教程参考https://github.com/InternLM/tutorial/blob/main/xtuner/README.md中的**3 自定义微调**，使用其中的xlsx2jsonl.py将原始数据从xlsx转化成json，链接中的xlsx2jsonl.py运行有报错，未进行了修改，整个修改后的xlsx2jsonl.py如下：

```
import openpyxl
import json

def process_excel_to_json(input_file, output_file):
    # Load the workbook
    wb = openpyxl.load_workbook(input_file)

    # Select the "DrugQA" sheet
    sheet = wb["Sheet1"]

    # Initialize the output data structure
    output_data = []

    # Iterate through each row in column A and D
    for row in sheet.iter_rows(min_row=2, max_col=4):
        system_value = "你是一名XXX专家，你能回答关于XXX的知识。"
        # print(row[0].value)
        # Create the conversation dictionary
        conversation = {
            "system": system_value,
            "input": row[0].value,  #改成对应自己数据集的input
            "output": row[3].value #改成对应自己数据集的output
        }

        # Append the conversation to the output data
        output_data.append({"conversation": [conversation]})

    # Write the output data to a JSON file
    with open(output_file, 'w', encoding='utf-8') as json_file:
        json.dump(output_data, json_file, indent=4)

```

```
    print(f"Conversion complete. Output written to {output_file}")

# 修改路中路径  Replace 'yzimu.xlsx' and 'output.jsonl' with your actual input and output file names
process_excel_to_json('yzimu.xlsx', 'output.jsonl')
```

4、根据导入的模型、数据集修改配置文件

打开刚刚下载的qwen_7b_chat_qlora_oasst1_e3.py进行如下修改

```
# 修改import部分
- from xtuner.dataset.map_fns import oasst1_map_fn, template_map_fn_factory
+ from xtuner.dataset.map_fns import template_map_fn_factory

# 根据自己存的模型位置，修改模型为本地路径
- pretrained_model_name_or_path = 'internlm/internlm-chat-7b'
+ pretrained_model_name_or_path = './Qwen-7B-Chat'

# 修改训练数据为 自定义.jsonl数据 路径，就是xlsx2jsonl.py的输出
- data_path = 'timdettmers/openassistant-guanaco'
+ data_path = './data/output.jsonl'

# 修改 train_dataset 对象
train_dataset = dict(
    type=process_hf_dataset,
-    dataset=dict(type=load_dataset, path=data_path),
+    dataset=dict(type=load_dataset, path='json', data_files=dict(train=data_path)),
    tokenizer=tokenizer,
    max_length=max_length,
-    dataset_map_fn=alpaca_map_fn,
+    dataset_map_fn=None,
    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)
```

建议修改qwen_7b_chat_qlora_oasst1_e3.py中的以下高亮部分，原来interval=1，则每个epoch都会产生一个pth权重参数文件，占用很大内存，改成interval=max_epochs即选择效果最好的保存即可。

```
# configure default hooks
```

```
default_hooks = dict(
    # record the time of every iteration.
    timer=dict(type=IterTimerHook),
    # print log every 100 iterations.
    logger=dict(type=LoggerHook, interval=10),
    # enable the parameter scheduler.
    param_scheduler=dict(type=ParamSchedulerHook),
    # save checkpoint per epoch.
    checkpoint=dict(type=CheckpointHook, interval=1),
    checkpoint=dict(type=CheckpointHook, interval=max_epochs),
    # set sampler seed in distributed evrionment.
    sampler_seed=dict(type=DistSamplerSeedHook),
)
```

文件里面的其他超参说明：

常用超参

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
evaluation_inputs	训练过程中，会根据给定的问题进行推理，便于观测训练状态
evaluation_freq	Evaluation 的评测间隔 iter 数
……	……

> 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。

5、开始微调

使用xtuner train命令运行刚刚修改的微调配置文件qwen_7b_chat_qlora_oasst1_e3.py，即可开始训练

```
# 单卡## 用刚才改好的config文件训练
xtuner train ./qwen_7b_chat_qlora_oasst1_e3.py

# 多卡
NPROC_PER_NODE=${GPU_NUM} xtuner train ./qwen_7b_chat_qlora_oasst1_e3.py --
deepspeed deepspeed_zero2

# 若要开启 deepspeed 加速，增加 --deepspeed deepspeed_zero2 即可
```

运行后，开始训练，结束后得到的 PTH 模型文件和其他杂七杂八的文件都默认在当前的 `./work_dirs` 中。

6、微调后部署测试

1、将得到的 PTH 模型转换为 HuggingFace 模型，即：生成 Adapter 文件夹

```
mkdir hf #存放训练得到的pth模型所生成的HuggingFace模型，即Adapter
export MKL_SERVICE_FORCE_INTEL=1
export MKL_THREADING_LAYER=GNU
xtuner convert pth_to_hf ./qwen_7b_chat_qlora_oasst1_e3.py
./work_dirs/qwen_7b_chat_qlora_oasst1_e3/epoch_100.pth ./hf #其中epoch_100.pth根据生成的名字进行修改
```

此时，hf 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

2、将 HuggingFace adapter 合并到大语言模型

运行下列程序后，会生成一个merged文件夹，里面是微调后的模型和大语言模型的融合。

```
xtuner convert merge ./Qwen-7B-Chat ./hf ./merged --max-shard-size 2GB
```

3、与合并后的模型对话

```
# 加载 Adapter 模型对话（Float 16）
xtuner chat ./merged --prompt-template qwen_chat

# 4 bit 量化加载
# xtuner chat ./merged --bits 4 --prompt-template qwen_chat
```

`xtuner chat` 的启动参数

启动参数	干哈滴
--prompt-template	指定对话模板
--system	指定SYSTEM文本
--system-template	指定SYSTEM模板
--bits	LLM位数
--bot-name	bot名称
--with-plugins	指定要使用的插件
--no-streamer	是否启用流式传输
--lagent	是否使用lagent
--command-stop-word	命令停止词
--answer-stop-word	回答停止词
--offload-folder	存放模型权重的文件夹（或者已经卸载模型权重的文件夹）
--max-new-tokens	生成文本中允许的最大 `token` 数量
--temperature	温度值
--top-k	保留用于顶k筛选的最高概率词汇标记数
--top-p	如果设置为小于1的浮点数，仅保留概率相加高于 `top_p` 的最小一组最有可能的标记
--seed	用于可重现文本生成的随机种子

7、LangChain 搭建知识库（https://github.com/InternLM/tutorial/tree/main/langchain）

1、安装依赖

```
pip install langchain==0.0.292
pip install gradio==4.4.0
pip install chromadb==0.4.15
pip install sentence-transformers==2.2.2
pip install unstructured==0.10.30
pip install markdown==3.3.7
```

同时，我们需要使用到开源词向量模型 Sentence Transformer:（我们也可以选用别的开源词向量模型来进行 Embedding，目前选用这个模型是相对轻量、支持中文且效果较好的，同学们可以自由尝试别的开源词向量模型）

首先需要使用 `huggingface` 官方提供的 `huggingface-cli` 命令行工具。安装依赖:

```
pip install -U huggingface_hub
```

然后在和 `/root/knowledge_data` 目录下新建python文件 `download_hf.py` ，填入以下代码：

- resume-download：断点续下
- local-dir：本地存储路径。（linux环境下需要填写绝对路径）

```
import os
# 设置环境变量,下载快一点
os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'
# 下载模型
os.system('huggingface-cli download --resume-download sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 --local-dir /root/knowledge_data/model/sentence-transformers')
```

```
#或者命令行下载，有点慢
git clone https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2
#git clone https://github.com/shinichiro-takahashi-sbr/paraphrase-multilingual-MiniLM-L12-v2.git
```

然后，在 `/root/data` 目录下执行该脚本即可自动开始下载：

```
python download_hf.py
```

2、下载 `nltk` 资源

我们在使用开源词向量模型构建开源词向量的时候，需要用到第三方库 `nltk` 的一些资源。我们用以下命令下载 nltk 资源并解压到服务器上：

```
cd /root
git clone https://gitee.com/yzy0612/nltk_data.git  --branch gh-pages
cd nltk_data
mv packages/*  ./
cd tokenizers
unzip punkt.zip
cd ../taggers
unzip averaged_perceptron_tagger.zip
```

3、下载相关代码

```
cd /root/knowledge_data
git clone https://github.com/InternLM/tutorial
```

4、知识库搭建

1）数据收集

把你准备好的自己的知识库，如txt、md等格式，存放在以下路径中

```
cd /root/knowledge_data
mkdir data
#txt、md等格式文档存放在这里面作为一个知识库
```

2）运行知识库脚本

```
cd /root/knowledge_data
mkdir demo
```

在demo中新建一个create_db.py文件，复制以下代码，修改数据路路径。

详细描述见https://github.com/InternLM/tutorial/tree/main/langchain的**2 知识库搭建**。可以在 `/root/knowledge_data` 下新建一个 `demo` 目录，将该脚本和后续脚本均放在该目录下运行。运行上述脚本，即可在本地构建已持久化的向量数据库，后续直接导入该数据库即可，无需重复构建。以下代码高亮部分要修改

```
# 首先导入所需第三方库
from langchain.document_loaders import UnstructuredFileLoader
from langchain.document_loaders import UnstructuredMarkdownLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Chroma
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
from tqdm import tqdm
import os

# 获取文件路径函数
def get_files(dir_path):
    # args: dir_path，目标文件夹路径
    file_list = []
    for filepath, dirnames, filenames in os.walk(dir_path):
```

```
        # os.walk 函数将递归遍历指定文件夹
        for filename in filenames:
            # 通过后缀名判断文件类型是否满足要求
            if filename.endswith(".md"):
                # 如果满足要求，将其绝对路径加入到结果列表
                file_list.append(os.path.join(filepath, filename))
            elif filename.endswith(".txt"):
                file_list.append(os.path.join(filepath, filename))
    return file_list

# 加载文件函数
def get_text(dir_path):
    # args: dir_path，目标文件夹路径
    # 首先调用上文定义的函数得到目标文件路径列表
    file_lst = get_files(dir_path)
    # docs 存放加载之后的纯文本对象
    docs = []
    # 遍历所有目标文件
    for one_file in tqdm(file_lst):
        file_type = one_file.split('.')[-1]
        if file_type == 'md':
            loader = UnstructuredMarkdownLoader(one_file)
        elif file_type == 'txt':
            loader = UnstructuredFileLoader(one_file)
        else:
            # 如果是不符合条件的文件，直接跳过
            continue
        docs.extend(loader.load())
    return docs

# 目标文件夹改成自己的数据
tar_dir = [
    "/root/knowledge_data/InternLM",
    "/root/knowledge_data/InternLM-XComposer"
]

# 加载目标文件
docs = []
for dir_path in tar_dir:
    docs.extend(get_text(dir_path))

# 对文本进行分块
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500, chunk_overlap=150)
split_docs = text_splitter.split_documents(docs)

# 加载开源词向量模型
```

```
embeddings = HuggingFaceEmbeddings(model_name="/root/data/model/sentence-transformer")

# 构建向量数据库
# 定义持久化路径
persist_directory = 'data_base/vector_db/chroma'
# 加载数据库
vectordb = Chroma.from_documents(
    documents=split_docs,
    embedding=embeddings,
    persist_directory=persist_directory  # 允许我们将persist_directory目录保存到磁盘上
)
# 将加载的向量数据库持久化到磁盘上
vectordb.persist()
```

然后进入到demo文件夹

```
cd demo
#然后运行脚本
python create_db.py
```

完成后会在当前目录生成一个data_base文件夹，这个就是所构建的向量数据库的文件

5、InternLM 接入 LangChain

为便捷构建 LLM 应用，我们需要基于本地部署的 InternLM，继承 LangChain 的 LLM 类自定义一个 InternLM LLM 子类，从而实现将 InternLM 接入到 LangChain 框架中。完成 LangChain 的自定义 LLM 子类之后，可以以完全一致的方式调用 LangChain 的接口，而无需考虑底层模型调用的不一致。

基于本地部署的 InternLM 自定义 LLM 类并不复杂，我们只需从 LangChain.llms.base.LLM 类继承一个子类，并重写构造函数与 `_call` 函数即可：

将上述代码放在demo文件夹下，命名为LLM.py，用于调用本地部署的大模型

```
from langchain.llms.base import LLM
from typing import Any, List, Optional
from langchain.callbacks.manager import CallbackManagerForLLMRun
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

class InternLM_LLM(LLM):
    # 基于本地 InternLM 自定义 LLM 类
    tokenizer : AutoTokenizer = None
```

```
    model: AutoModelForCausalLM = None

    def __init__(self, model_path :str):
        # model_path: InternLM 模型路径
        # 从本地初始化模型
        super().__init__()
        print("正在从本地加载模型...")
        self.tokenizer = AutoTokenizer.from_pretrained(model_path, trust_remote_code=True)
        self.model = AutoModelForCausalLM.from_pretrained(model_path, trust_remote_code=True).to(torch.bfloat16).cuda()
        self.model = self.model.eval()
        print("完成本地模型的加载")

    def _call(self, prompt : str, stop: Optional[List[str]] = None,
              run_manager: Optional[CallbackManagerForLLMRun] = None,
              **kwargs: Any):
        # 重写调用函数
        system_prompt = """You are an AI assistant whose name is InternLM (书生·浦语).
        - InternLM (书生·浦语) is a conversational language model that is developed by Shanghai AI Laboratory (上海人工智能实验室). It is designed to be helpful, honest, and harmless.
        - InternLM (书生·浦语) can understand and communicate fluently in the language chosen by the user such as English and 中文.
        """

        messages = [(system_prompt, '')]
        response, history = self.model.chat(self.tokenizer, prompt , history=messages)
        return response

    @property
    def _llm_type(self) -> str:
        return "InternLM"
```

6、构建检索问答链

LangChain 通过提供检索问答链对象来实现对于 RAG 全流程的封装。所谓检索问答链，即通过一个对象完成检索增强问答（即RAG）的全流程，针对 RAG 的更多概念，我们会在视频内容中讲解，也欢迎读者查阅该教程来进一步了解：《LLM Universe》。我们可以调用一个 LangChain 提供的 `RetrievalQA` 对象，通过初始化时填入已构建的数据库和自定义 LLM 作为参数，来简便地完成检索增强问答的全流程，LangChain 会自动完成基于用户提问进行检索、获取相关文档、拼接为合适的 Prompt 并交给 LLM 问答的全部流程。

在demo文件夹下新建一个web_demo.py文件数据以下代码：高亮部分要修改成自己的路径

```
1
2  from langchain.vectorstores import Chroma
3  from langchain.embeddings.huggingface import HuggingFaceEmbeddings
4  import os
5  from LLM import InternLM_LLM
6  from langchain.prompts import PromptTemplate
7  from langchain.chains import RetrievalQA
8
9  def load_chain():
10     # 加载问答链
11     # 定义 Embeddings
12     embeddings =
   HuggingFaceEmbeddings(model_name="/root/knowledge_data/model/sentence-
   transformer")
13
14     # 向量数据库持久化路径
15     persist_directory = './data_base/vector_db/chroma'
16
17     # 加载数据库
18     vectordb = Chroma(
19         persist_directory=persist_directory,  # 允许我们将persist_directory目录保
   存到磁盘上
20         embedding_function=embeddings
21     )
22
23     # 加载自定义 LLM,将微调后的大模型传入其中
24     llm = InternLM_LLM(model_path = "/root/finetune/merged")
25
26     # 定义一个 Prompt Template
27     template = """使用以下上下文来回答最后的问题。如果你不知道答案，就说你不知道，不要试
   图编造答
28     案。尽量使答案简明扼要。总是在回答的最后说“谢谢你的提问！”。
29     {context}
30     问题: {question}
31     有用的回答:"""
32
33     QA_CHAIN_PROMPT = PromptTemplate(input_variables=
   ["context","question"],template=template)
34
35     # 运行 chain
36     qa_chain =
   RetrievalQA.from_chain_type(llm,retriever=vectordb.as_retriever(),return_source
   _documents=True,chain_type_kwargs={"prompt":QA_CHAIN_PROMPT})
37
38     return qa_chain
```

8、网页部署

在上面的web_demo.py添加以下代码，该类负责加载并存储检索问答链，并响应 Web 界面里调用检索问答链进行回答的动作：

```
class Model_center():
    """
    存储检索问答链的对象
    """
    def __init__(self):
        # 构造函数，加载检索问答链
        self.chain = load_chain()

    def qa_chain_self_answer(self, question: str, chat_history: list = []):
        """
        调用问答链进行回答
        """
        if question == None or len(question) < 1:
            return "", chat_history
        try:
            chat_history.append(
                (question, self.chain({"query": question})["result"]))
            # 将问答结果直接附加到问答历史中，Gradio 会将其展示出来
            return "", chat_history
        except Exception as e:
            return e, chat_history

```

继续在上面的web_demo.py添加以下代码，然后我们只需按照 Gradio 的框架使用方法，实例化一个 Web 界面并将点击动作绑定到上述类的回答方法即可：

```
import gradio as gr

# 实例化核心功能对象
model_center = Model_center()
# 创建一个 Web 界面
block = gr.Blocks()
with block as demo:
    with gr.Row(equal_height=True):
        with gr.Column(scale=15):
            # 展示的页面标题
            gr.Markdown("""<h1><center>InternLM</center></h1>
                <center>书生浦语</center>
                """)
```

```
14
15      with gr.Row():
16          with gr.Column(scale=4):
17              # 创建一个聊天机器人对象
18              chatbot = gr.Chatbot(height=450, show_copy_button=True)
19              # 创建一个文本框组件，用于输入 prompt。
20              msg = gr.Textbox(label="Prompt/问题")
21
22              with gr.Row():
23                  # 创建提交按钮。
24                  db_wo_his_btn = gr.Button("Chat")
25              with gr.Row():
26                  # 创建一个清除按钮，用于清除聊天机器人组件的内容。
27                  clear = gr.ClearButton(
28                      components=[chatbot], value="Clear console")
29
30          # 设置按钮的点击事件。当点击时，调用上面定义的 qa_chain_self_answer 函数，并传入用户的消息和聊天历史记录，然后更新文本框和聊天机器人组件。
31          db_wo_his_btn.click(model_center.qa_chain_self_answer, inputs=[
32                              msg, chatbot], outputs=[msg, chatbot])
33
34      gr.Markdown("""提醒：<br>
35      1. 初始化数据库时间可能较长，请耐心等待。
36      2. 使用中如果出现异常，将会在文本输入框进行展示，请不要惊慌。 <br>
37      """)
38  gr.close_all()
39  # 直接启动
40  demo.launch()
```

通过将上述代码封装为 run_gradio.py 脚本，直接通过 python 命令运行，即可在本地启动知识库助手的 Web Demo，默认会在 7860 端口运行，接下来将服务器端口映射到本地端口即可访问。（查看本教程5.2配置本地端口后，将端口映射到本地。在本地浏览器输入 `http://127.0.0.1:6006` 即可。）

9、其他备注

1、网页部署

安装相关依赖

```
1  pip install streamlit==1.24.0
2  pip install gradio mdtex2html #??
3  pip install transformers==4.33.1 timm==0.4.12 sentencepiece==0.1.99
   gradio==3.44.4 markdown2==2.4.10 xlsxwriter==3.1.2 einops accelerate ##??
```

```
mkdir ./personal_assistant/code && cd ./personal_assistant/code
git clone https://github.com/Qwen/Qwen.git
```

将 ‘./personal_assistant/`code/Qwen/web_demo.py`’ 中模型路径DEFAULT_CKPT_PATH 更换为Merge后存放参数的路径 '~/finetune/merged'

```
DEFAULT_CKPT_PATH = '~/finetune/merged'#改成自己的路径
```

```
"""A simple web interactive chat demo based on gradio."""
import os
from argparse import ArgumentParser

import gradio as gr
import mdtex2html

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers.generation import GenerationConfig


DEFAULT_CKPT_PATH = '/root/autodl-tmp/yzumi/merged'


def _get_args():
    parser = ArgumentParser()
    parser.add_argument("-c", "--checkpoint-path", type=str, default=DEFAULT_CKPT_PATH,
                        help="Checkpoint name or path, default to %(default)r")
    parser.add_argument("--cpu-only", action="store_true", help="Run demo with CPU only")

    parser.add_argument("--share", action="store_true", default=False,
                        help="Create a publicly shareable link for the interface.")
    parser.add_argument("--inbrowser", action="store_true", default=False,
                        help="Automatically launch the interface in a new tab on the default browser.")
    parser.add_argument("--server-port", type=int, default=8000,
                        help="Demo server port.")
    parser.add_argument("--server-name", type=str, default="127.0.0.1",
                        help="Demo server name.")

```

web demo 运行

然后运行 '`/root/personal_assistant/code/Q`wen' 目录下的 `web_demo.py` 文件，输入以下命令后，查看本教程5.2配置本地端口后，将端口映射到本地。在本地浏览器输入 `http://127.0.0.1:6006` 即可。

```
bash
cd /root/personal_assistant/code/Qwen
streamlit run web_demo.py --server.address 127.0.0.1 --server.port 6006
#或者以下命令也可以
#streamlit run /root/finetune/personal_assistant/code/Qwen/web_demo.py --
server.address 127.0.0.1 --server.port 6006
```