

项目记录：使用Qwen-chat-7b微调+外挂RAG 知识库实现流程和效果展示

1、大语言模型微调工具箱

Xtuner

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

```
1 # 创建版本文件夹并进入，以跟随本教程
2 mkdir xtuner019 && cd xtuner019
3
4 # 拉取 0.1.9 的版本源码
5 git clone -b v0.1.9 https://github.com/InternLM/xtuner
6 # 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
7
8 # 从源码安装 XTuner
9 pip install -e '[all]'
10
11 # 创建一个微调自定义数据集的工作路径，进入
12 mkdir ~/finetune && cd ~/finetune
```

4、微调工作准备

1、准备配置文件

```
1 # 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
2 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
=====
```

- 1 cd ~/finetune
- 2 xtuner copy-cfg qwen_7b_chat_qlora_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

- 1 # 创建一个目录，放模型文件，防止散落一地
- 2 mkdir ~/finetune/qwen-chat-7b
- 3
- 4 # 装一下拉取模型文件要用的库
- 5 pip install modelscope
- 6 # 从 modelscope 下载模型文件
- 7 cd ~/finetune/qwen-chat-7b
- 8 apt install git git-lfs -y ###sudo apt-get update → sudo apt-get git git-
lfs -y
- 9 git lfs install

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

3、数据集导入

```
1 cd ~/finetune
2 mkdir data
3 #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：

```
1 import openpyxl
2 import json
3
4 def process_excel_to_json(input_file, output_file):
5     # Load the workbook
6     wb = openpyxl.load_workbook(input_file)
7
8     # Select the "DrugQA" sheet
9     sheet = wb["Sheet1"]
10
11     # Initialize the output data structure
12     output_data = []
13
14     # Iterate through each row in column A and D
15     for row in sheet.iter_rows(min_row=2, max_col=4):
16         system_value = "你是一名XXX专家，你能回答关于XXX的知识。"
17         # print(row[0].value)
18         # Create the conversation dictionary
19         conversation = {
20             "system": system_value,
21             "input": row[0].value, #改成对应自己数据集的input
22             "output": row[3].value #改成对应自己数据集的output
23         }
24
25         # Append the conversation to the output data
26         output_data.append({"conversation": [conversation]})
27
28     # Write the output data to a JSON file
29     with open(output_file, 'w', encoding='utf-8') as json_file:
30         json.dump(output_data, json_file, indent=4)
31
```

```

32     print(f"Conversion complete. Output written to {output_file}")
33
34 # 修改路中路径 Replace 'yzimu.xlsx' and 'output.jsonl' with your actual input
    and output file names
35 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

```

1 # 修改import部分
2 from xtuner.dataset.map_fns import oasst1_map_fn, template_map_fn_factory
3 + from xtuner.dataset.map_fns import template_map_fn_factory
4
5 # 根据自己存的模型位置，修改模型为本地路径
6 pretrained_model_name_or_path = 'internlm/internlm-chat-7b'
7 + pretrained_model_name_or_path = './Qwen-7B-Chat'
8
9 # 修改训练数据为 自定义.jsonl数据 路径，就是xlsx2jsonl.py的输出
10 data_path = 'timdettmers/openassistant-guanaco'
11 + data_path = './data/output.jsonl'
12
13 # 修改 train_dataset 对象
14 train_dataset = dict(
15     type=process_hf_dataset,
16 dataset=dict(type=load_dataset, path=data_path),
17 + dataset=dict(type=load_dataset, path='json',
    data_files=dict(train=data_path)),
18     tokenizer=tokenizer,
19     max_length=max_length,
20 dataset_map_fn=alpaca_map_fn,
21 + dataset_map_fn=None,
22     template_map_fn=dict(
23         type=template_map_fn_factory, template=prompt_template),
24     remove_unused_columns=True,
25     shuffle_before_pack=True,
26     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。

```

1 # configure default hooks

```

```

2 default_hooks = dict(
3     # record the time of every iteration.
4     timer=dict(type=IterTimerHook),
5     # print log every 100 iterations.
6     logger=dict(type=LoggerHook, interval=10),
7     # enable the parameter scheduler.
8     param_scheduler=dict(type=ParamSchedulerHook),
9     # save checkpoint per epoch.
10    checkpoint=dict(type=CheckpointHook, interval=1),
11    checkpoint=dict(type=CheckpointHook, interval=max_epochs),
12    # set sampler seed in distributed environment.
13    sampler_seed=dict(type=DistSamplerSeedHook),
14 )

```

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成 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
.....

如果想把显卡的内存吃满，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

```

1 # 单卡## 用刚才改好的config文件训练
2 xtuner train ./qwen_7b_chat_qlora_oasst1_e3.py
3
4 # 多卡
5 NPROC_PER_NODE=${GPU_NUM} xtuner train ./qwen_7b_chat_qlora_oasst1_e3.py --
  deepspeed deepspeed_zero2
6
7 # 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

```
1 mkdir hf #存放训练得到的pth模型所生成的HuggingFace模型，即Adapter
2 export MKL_SERVICE_FORCE_INTEL=1
3 export MKL_THREADING_LAYER=GNU
4 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。

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

3、与合并后的模型对话

```
1 # 加载 Adapter 模型对话 (Float 16)
2 xtuner chat ./merged --prompt-template qwen_chat
3
4 # 4 bit 量化加载
5 # xtuner chat ./merged --bits 4 --prompt-template qwen_chat
```

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

```

1 pip install langchain==0.0.292
2 pip install gradio==4.4.0
3 pip install chromadb==0.4.15
4 pip install sentence-transformers==2.2.2
5 pip install unstructured==0.10.30
6 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：

```
1 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）

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

```
1 #或者命令行下载, 有点慢
2 git clone https://huggingface.co/sentence-transformers/paraphrase-multilingual-
MiniLM-L12-v2
3 #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：

```
1 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：

```
1 cd /root
2 git clone https://gitee.com/yzy0612/nltk_data.git --branch gh-pages
3 cd nltk_data
4 mv packages/* ./
5 cd tokenizers
6 unzip punkt.zip
7 cd ../taggers
8 unzip averaged_perceptron_tagger.zip
```


3、下载相关代码

```
1 cd /root/knowledge_data
2 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

```
1 cd /root/knowledge_data
2 mkdir data
3 #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

```
1 cd /root/knowledge_data
2 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

```
1 # 首先导入所需第三方库
2 from langchain.document_loaders import UnstructuredFileLoader
3 from langchain.document_loaders import UnstructuredMarkdownLoader
4 from langchain.text_splitter import RecursiveCharacterTextSplitter
5 from langchain.vectorstores import Chroma
6 from langchain.embeddings.huggingface import HuggingFaceEmbeddings
7 from tqdm import tqdm
8 import os
9
10 # 获取文件路径函数
11 def get_files(dir_path):
12     # args: dir_path, 目标文件夹路径
13     file_list = []
14     for filepath, dirnames, filenames in os.walk(dir_path):
```

```

15     # os.walk 函数将递归遍历指定文件夹
16     for filename in filenames:
17         # 通过后缀名判断文件类型是否满足要求
18         if filename.endswith(".md"):
19             # 如果满足要求, 将其绝对路径加入到结果列表
20             file_list.append(os.path.join(filepath, filename))
21         elif filename.endswith(".txt"):
22             file_list.append(os.path.join(filepath, filename))
23     return file_list
24
25 # 加载文件函数
26 def get_text(dir_path):
27     # args: dir_path, 目标文件夹路径
28     # 首先调用上文定义的函数得到目标文件路径列表
29     file_lst = get_files(dir_path)
30     # docs 存放加载之后的纯文本对象
31     docs = []
32     # 遍历所有目标文件
33     for one_file in tqdm(file_lst):
34         file_type = one_file.split('.')[-1]
35         if file_type == 'md':
36             loader = UnstructuredMarkdownLoader(one_file)
37         elif file_type == 'txt':
38             loader = UnstructuredFileLoader(one_file)
39         else:
40             # 如果是不符合条件的文件, 直接跳过
41             continue
42         docs.extend(loader.load())
43     return docs
44
45 # 目标文件夹改成自己的数据
46 tar_dir = [
47     "/root/knowledge_data/InternLM",
48     "/root/knowledge_data/InternLM-XComposer"
49 ]
50
51 # 加载目标文件
52 docs = []
53 for dir_path in tar_dir:
54     docs.extend(get_text(dir_path))
55
56 # 对文本进行分块
57 text_splitter = RecursiveCharacterTextSplitter(
58     chunk_size=500, chunk_overlap=150)
59 split_docs = text_splitter.split_documents(docs)
60
61 # 加载开源词向量模型

```

```

62 embeddings = HuggingFaceEmbeddings(model_name="/root/data/model/sentence-
    transformer")
63
64 # 构建向量数据库
65 # 定义持久化路径
66 persist_directory = 'data_base/vector_db/chroma'
67 # 加载数据库
68 vectordb = Chroma.from_documents(
69     documents=split_docs,
70     embedding=embeddings,
71     persist_directory=persist_directory # 允许我们将persist_directory目录保存到磁
    盘上
72 )
73 # 将加载的向量数据库持久化到磁盘上
74 vectordb.persist()

```

然后进入到demo文件夹

```

1 cd demo
2 #然后运行脚本
3 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

```

1 from langchain.llms.base import LLM
2 from typing import Any, List, Optional
3 from langchain.callbacks.manager import CallbackManagerForLLMRun
4 from transformers import AutoTokenizer, AutoModelForCausalLM
5 import torch
6
7 class InternLM_LLM(LLM):
8     # 基于本地 InternLM 自定义 LLM 类
9     tokenizer : AutoTokenizer = None

```

```

10     model: AutoModelForCausalLM = None
11
12     def __init__(self, model_path :str):
13         # model_path: InternLM 模型路径
14         # 从本地初始化模型
15         super().__init__()
16         print("正在从本地加载模型...")
17         self.tokenizer = AutoTokenizer.from_pretrained(model_path,
18 trust_remote_code=True)
19         self.model = AutoModelForCausalLM.from_pretrained(model_path,
20 trust_remote_code=True).to(torch.bfloat16).cuda()
21         self.model = self.model.eval()
22         print("完成本地模型的加载")
23
24     def _call(self, prompt : str, stop: Optional[List[str]] = None,
25 run_manager: Optional[CallbackManagerForLLMRun] = None,
26 **kwargs: Any):
27         # 重写调用函数
28         system_prompt = """You are an AI assistant whose name is InternLM (书生·
29 浦语).
30 - InternLM (书生·浦语) is a conversational language model that is
31 developed by Shanghai AI Laboratory (上海人工智能实验室). It is designed to be
32 helpful, honest, and harmless.
33 - InternLM (书生·浦语) can understand and communicate fluently in the
34 language chosen by the user such as English and 中文.
35 """
36         messages = [(system_prompt, '')]
37         response, history = self.model.chat(self.tokenizer, prompt ,
38 history=messages)
39         return response
40
41     @property
42     def _llm_type(self) -> str:
43         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：

```
1 class Model_center():
2     """
3     存储检索问答链的对象
4     """
5     def __init__(self):
6         # 构造函数，加载检索问答链
7         self.chain = load_chain()
8
9     def qa_chain_self_answer(self, question: str, chat_history: list = []):
10        """
11        调用问答链进行回答
12        """
13        if question == None or len(question) < 1:
14            return "", chat_history
15        try:
16            chat_history.append(
17                (question, self.chain({"query": question})["result"]))
18            # 将问答结果直接附加到问答历史中，Gradio 会将其展示出来
19            return "", chat_history
20        except Exception as e:
21            return e, chat_history
22
```

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：

```
1 import gradio as gr
2
3 # 实例化核心功能对象
4 model_center = Model_center()
5 # 创建一个 Web 界面
6 block = gr.Blocks()
7 with block as demo:
8     with gr.Row(equal_height=True):
9         with gr.Column(scale=15):
10             # 展示的页面标题
11             gr.Markdown("""<h1><center>InternLM</center></h1>
12                 <center>书生浦语</center>
13                 """)
```

```

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、效果演示（初版）

微调训练过程：

损失0.5左右就有点过拟合了（经验值）

```
07 22:10:27 - mmengine - INFO - Epoch(train) [65][5/5] lr: 1.7292e-05 eta: 0:01:25 time: 1.1400 data_time: 0.0043 memory: 9868 loss: 1.5820
07 22:10:32 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:10:32 - mmengine - INFO - Epoch(train) [66][5/5] lr: 1.5149e-05 eta: 0:01:19 time: 1.1380 data_time: 0.0043 memory: 9868 loss: 1.5805
07 22:10:38 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:10:38 - mmengine - INFO - Epoch(train) [67][5/5] lr: 1.3137e-05 eta: 0:01:14 time: 1.1382 data_time: 0.0043 memory: 9868 loss: 1.5805
07 22:10:44 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:10:44 - mmengine - INFO - Epoch(train) [68][5/5] lr: 1.1259e-05 eta: 0:01:08 time: 1.1401 data_time: 0.0043 memory: 9868 loss: 1.5703
07 22:10:49 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:10:49 - mmengine - INFO - Epoch(train) [69][5/5] lr: 9.5173e-06 eta: 0:01:02 time: 1.1401 data_time: 0.0043 memory: 9868 loss: 1.5727
07 22:10:55 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:10:55 - mmengine - INFO - Epoch(train) [70][5/5] lr: 7.9155e-06 eta: 0:00:56 time: 1.1382 data_time: 0.0043 memory: 9868 loss: 1.5758
07 22:11:01 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:01 - mmengine - INFO - Epoch(train) [71][5/5] lr: 6.4556e-06 eta: 0:00:51 time: 1.1400 data_time: 0.0043 memory: 9868 loss: 1.5711
07 22:11:07 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:07 - mmengine - INFO - Epoch(train) [72][5/5] lr: 5.1400e-06 eta: 0:00:45 time: 1.1401 data_time: 0.0043 memory: 9868 loss: 1.5688
07 22:11:12 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:12 - mmengine - INFO - Epoch(train) [73][5/5] lr: 3.9706e-06 eta: 0:00:39 time: 1.1382 data_time: 0.0043 memory: 9868 loss: 1.5672
07 22:11:18 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:18 - mmengine - INFO - Epoch(train) [74][5/5] lr: 2.9494e-06 eta: 0:00:34 time: 1.1401 data_time: 0.0044 memory: 9868 loss: 1.5719
07 22:11:24 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:24 - mmengine - INFO - Epoch(train) [75][5/5] lr: 2.0777e-06 eta: 0:00:28 time: 1.1401 data_time: 0.0043 memory: 9868 loss: 1.5703
07 22:11:29 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:29 - mmengine - INFO - Epoch(train) [76][5/5] lr: 1.3571e-06 eta: 0:00:22 time: 1.1383 data_time: 0.0043 memory: 9868 loss: 1.5641
07 22:11:35 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:35 - mmengine - INFO - Epoch(train) [77][5/5] lr: 7.8853e-07 eta: 0:00:17 time: 1.1402 data_time: 0.0044 memory: 9868 loss: 1.5609
07 22:11:41 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:41 - mmengine - INFO - Epoch(train) [78][5/5] lr: 3.7296e-07 eta: 0:00:11 time: 1.1402 data_time: 0.0044 memory: 9868 loss: 1.5602
07 22:11:46 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:46 - mmengine - INFO - Epoch(train) [79][5/5] lr: 1.1101e-07 eta: 0:00:05 time: 1.1383 data_time: 0.0044 memory: 9868 loss: 1.5648
07 22:11:52 - mmengine - INFO - Exp name: internlm_chat_7b_qlora_oasst1_e3_copy_20240407_220354
07 22:11:52 - mmengine - INFO - Epoch(train) [80][5/5] lr: 3.0842e-09 eta: 0:00:00 time: 1.1403 data_time: 0.0044 memory: 9868 loss: 1.5641
07 22:11:52 - mmengine - INFO - Saving checkpoint at 80 epochs
24-04-07 22:11:52.792 [INFO] [logging.py:96:log_dist] [Rank 0] [Torch] Checkpoint epoch 80.pth is about to be saved!
```

微调后测试对话演示效果：

融合进了微调训练数据的知识。

```
double enter to end input (EXIT: exit chat, RESET: reset history) >>> 注塑机出现加料慢时间不稳定的现象，怎么办
```

1. 检查料筒温度是否正常，如果温度过低，会导致物料流动性差，加料慢。
2. 检查螺杆转速是否正常，如果转速过低，会导致物料流动性差，加料慢。
3. 检查螺杆是否需要清洗，如果螺杆上有残留物，会导致物料流动性差，加料慢。
4. 检查模具是否需要清洗，如果模具上有残留物，会导致物料流动性

网页端部署，加入知识库后的效果：

回复确实用到了知识库中的内容

Chatbot

你好你是谁

你好！我是上海人工智能实验室研发的一款AI助手，我叫InternLM，您可以随时向我提出您想要了解的问题，我会尽全力为您解答！

注塑机知识你有吗

注塑成型又称注射模塑成型，它是一种注射兼模塑的成型方法。

注塑成型方法的优点是生产速度快、效率高，操作可实现自动化，花色品种多，形状可以由简到繁，尺寸可以由大到小，而且制品尺寸精确，产品易更新换代，能成形状复杂的制件，注塑成型适用于大量生产与形状复杂产品等成型加工领域。

Prompt/问题

Chat

Clear console

提醒：

1. 初始化数据库时间可能较长，请耐心等待。
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。