1.数据集

1.数据集格式简介

LLM 的微调一般指指令微调过程。所谓指令微调，是说我们使用的微调数据形如：

JSON

```
{
    "instrution":"回答以下用户问题，仅输出答案。",
    "input":"1+1等于几?",
    "output":"2"
}
```

其中，`instruction` 是用户指令，告知模型其需要完成的任务；`input` 是用户输入，是完成用户指令所必须的输入内容；`output` 是模型应该给出的输出。

即核心训练目标是让模型具有理解并遵循用户指令的能力。因此，在指令集构建时，应针对我们的目标任务，针对性构建任务指令集。如下目标是构建一个能够模拟派蒙对话风格的个性化 LLM，因此构造的指令形如：

JSON

```
{
    "instruction": "现在你要扮演在一个开放世界中探险的向导--派蒙",
    "input":"你是谁？",
    "output":"我叫派蒙！很高兴认识你，我是一位向导。"
}
```

2.提取对话方式

使用 ChatGPT 和 Kor 工具提取小说中的对话。

`Kor` 是个支持 `LangChain` 的文本抽取库，可以把文本抽取成 `json` 格式。简单使用一下 `Kor`，首先用 `langchain` 的 `LLM` 模块重新封装一下，`langchian` 中的 `ChatOpenAI` 类。

openai 接口封装, 注意 `OPENAI_API_KEY`

Python

```python
import openai
import os
from langchain.llms.base import LLM
from typing import Dict, List, Optional, Tuple, Union
from dotenv import find_dotenv, load_dotenv

_ = load_dotenv(find_dotenv())

openai.api_key = os.environ["OPENAI_API_KEY"]

def get_completion(prompt, model="gpt-3.5-turbo"):
    '''
    prompt: 对应的提示词
    model: 调用的模型，默认为 gpt-3.5-turbo(ChatGPT)，有内测资格的用户可以选择
    '''
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # 模型输出的温度系数，控制输出的随机程度
    )
    # 调用 OpenAI 的 ChatCompletion 接口
    return response.choices[0].message["content"]

class OpenAI_LLM(LLM):
    model_type: str = "openai"


    def __init__(self):
        super().__init__()

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

    def _call(self, prompt: str, history: List = [], stop: Optional[Li
        res = get_completion(prompt)
        return res

    @property
```

```
    def _identifying_params(self):
        """Get the identifying parameters.
        """
        _param_dict = {
            "model": self.model_type
        }
        return _param_dict
```

`Kor` 的核心就要写一段对所提取信息的描述，以及几个示例，在下面的例子中（挑取了小说《诡秘之主》中的一个片段），所要提取的信息分别是 `role`，`dialogue`，也就是角色和其对应的台词。

纯文本

```
[
        Text(
            id="role",
            description="The character who is speaking",
        ),
        Text(
            id="dialogue",
            description="The dialogue spoken by the characters in the
        )
]
```

输出示例：

Python

```
(
            '''
            他下意识放轻了脚步，不制造出明显的噪音。
            刚登上二楼，他看见盥洗室的门突然打开，穿着旧布长裙的梅丽莎一副睡眼惺忪
            “你回来了……”梅丽莎还有些迷糊地揉了揉眼睛。
            克莱恩掩住嘴巴，打了个哈欠道：
            “是的，我需要一个美好的梦境，午餐之前都不要叫醒我。”
            梅丽莎“嗯”了一声，忽然想起什么似地说道：
            “我和班森上午要去圣赛琳娜教堂做祈祷，参与弥撒，午餐可能会迟一点。”
            ''',
            [
                {"role": "梅丽莎", "dialogue": "你回来了……"},
                {"role": "克莱恩", "dialogue": "是的，我需要一个美好的梦境，
```

```
                {"role": "梅丽莎", "dialogue":"我和班森上午要去圣赛琳娜教堂
            ],
)
```

3.构造数据集

对话可以由 ChatGPT 提取生成，也可以由自己收集，如下给出原神数据集构造部分代码：

Python

```python
import json
from tqdm import tqdm
import pandas as pd

def read_excel_column(excel_file, sheet_name="paimeng", column_index=1
    try:
        # 读取Excel文件指定sheet的内容
        df = pd.read_excel(excel_file, sheet_name=sheet_name)

        # 获取指定列的数据
        column_data = df.iloc[:, column_index].tolist()

        return column_data
    except Exception as e:
        print(f"Error: {e}")
        return []

def save_dataset(path, data):
    # filename = path.split('/')[-1].split('.')[0]
    with open(path, mode='w', encoding='utf-8') as f:
        f.write(json.dumps(data, ensure_ascii=False))

def gen_dataset(dialogue_list):
    res = []
    print('构造微调数据集')
    for i in tqdm(range(1, len(dialogue_list))):
        tmp = {
            "instruction": dialogue_list[i-1],
            "input": "",
            "output": dialogue_list[i]
```

```
            }
            res.append(tmp)
        return res

if __name__ == "__main__":
    dialogue_list = read_excel_column(excel_file="input\genshin_impact
    print(f"共有 {len(dialogue_list)} 条对话样本")
    dataset = gen_dataset(dialogue_list)
    print(f"获得 {len(dataset)} 条微调样本")
    save_dataset("output/paimeng.json", dataset)
```