基于 InternLM 和 LangChain 搭建你的知识库

1 环境配置

1.1 InternLM 模型部署

```
bash
# 创建并激活环境
/root/share/install_conda_env_internlm_base.sh InternLM
conda activate InternLM

# 安装依赖
python -m pip install --upgrade pip

pip install modelscope==1.9.5
pip install transformers==4.35.2
pip install streamlit==1.24.0
pip install sentencepiece==0.1.99
pip install accelerate==0.24.1
```

1.2 模型下载

本地拷贝

```
mkdir -p /root/data/model/Shanghai_AI_Laboratory
cp -r /root/share/temp/model_repos/internlm-chat-7b
/root/data/model/Shanghai_AI_Laboratory/internlm-chat-7b
```

或者用modelscope手动下载

```
import torch
from modelscope import snapshot_download, AutoModel, AutoTokenizer
import os
model_dir = snapshot_download('Shanghai_AI_Laboratory/internlm-chat-7b',
cache_dir='/root/data/model', revision='v1.0.3')
```

1.3 LangChain 相关环境配置

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
```

开源词向量模型 Sentence Transformer，相对轻量、支持中文且效果较好的

首先使用 `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/data` 目录下新建python文件 `download_hf.py`，填入代码并运行（**记得使用hf镜像**）：

```
import os

# 设置环境变量
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/data/model/sentence-transformer')
```

1.4 下载 NLTK 相关资源

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
```

1.5 下载本项目代码

将仓库 clone 到本地，可以直接在本地运行相关代码：

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

2 知识库搭建

2.1 数据收集

选择由上海人工智能实验室开源的一系列大模型工具开源仓库作为语料库来源

```
# 进入到数据库盘
cd /root/data
# clone 上述开源仓库
git clone https://gitee.com/open-compass/opencompass.git
git clone https://gitee.com/InternLM/lmdeploy.git
git clone https://gitee.com/InternLM/xtuner.git
git clone https://gitee.com/InternLM/InternLM-XComposer.git
git clone https://gitee.com/InternLM/lagent.git
git clone https://gitee.com/InternLM/InternLM.git
```

2.2 数据收集 & 数据加载 & 构建向量数据库

1. 数据收集

   选用上述仓库中所有的 **markdown**、**txt 文件**作为示例语料库（**代码文件**对逻辑联系要求较高，且规范性较强，在分割时最好基于代码模块进行分割再加入向量数据库）

2. 数据加载

   使用 LangChain 提供的 FileLoader 对象来加载目标文件，得到由目标文件解析出的纯文本内容

3. 构建向量数据库

   由纯文本对象构建向量数据库，我们需要先对文本进行分块，接着对文本块进行向量化。使用字符串递归分割器，并选择分块大小为 500，块重叠长度为 150。想要深入学习 LangChain 文本分块可以参考教程 《LangChain - Chat With Your Data》

```
# 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
tar_dir = [
    "/root/data/InternLM",
    "/root/data/InternLM-XComposer",
    "/root/data/lagent",
    "/root/data/lmdeploy",
    "/root/data/opencompass",
    "/root/data/xtuner"
]

# 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()
```

3 InternLM 接入 LangChain

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。

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：

```
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"
```

上述代码封装为`/root/data/LLM.py`，后续将直接从该文件中引入自定义的 LLM 类。

4 部署 Web Demo

在完成上述核心功能后，我们可以**基于 Gradio 框架**将其部署到 Web 网页，从而搭建一个小型 Demo，便于测试与使用。

我们首先将上文的代码内容封装为一个返回构建的**检索问答链对象的函数**，并在启动 Gradio 的第一时间调用该函数得到检索问答链对象，后续直接使用该对象进行问答对话，从而避免重复加载模型：

```
from langchain.vectorstores import Chroma
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
import os
from LLM import InternLM_LLM
from langchain.prompts import PromptTemplate
from langchain.chains import RetrievalQA

def load_chain():
    # 加载问答链
    # 定义 Embeddings
```

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

    # 向量数据库持久化路径
    persist_directory = 'data_base/vector_db/chroma'

    # 加载数据库
    vectordb = Chroma(
        persist_directory=persist_directory,  # 允许我们将persist_directory目录保存
到磁盘上
        embedding_function=embeddings
    )

    # 加载自定义 LLM
    llm = InternLM_LLM(model_path =
"/root/data/model/Shanghai_AI_Laboratory/internlm-chat-7b")

    # 定义一个 Prompt Template
    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
    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！”。
    {context}
    问题: {question}
    有用的回答:"""

    QA_CHAIN_PROMPT = PromptTemplate(input_variables=
["context","question"],template=template)

    # 运行 chain
    qa_chain =
RetrievalQA.from_chain_type(llm,retriever=vectordb.as_retriever(),return_source_
documents=True,chain_type_kwargs={"prompt":QA_CHAIN_PROMPT})

    return qa_chain
```

接着我们定义一个类，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
```

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>
                """)

    with gr.Row():
        with gr.Column(scale=4):
            # 创建一个聊天机器人对象
            chatbot = gr.Chatbot(height=450, show_copy_button=True)
            # 创建一个文本框组件，用于输入 prompt。
            msg = gr.Textbox(label="Prompt/问题")

            with gr.Row():
                # 创建提交按钮。
                db_wo_his_btn = gr.Button("Chat")
            with gr.Row():
                # 创建一个清除按钮，用于清除聊天机器人组件的内容。
                clear = gr.ClearButton(
                    components=[chatbot], value="Clear console")

        # 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。
        db_wo_his_btn.click(model_center.qa_chain_self_answer, inputs=[
                            msg, chatbot], outputs=[msg, chatbot])

    gr.Markdown("""提醒：<br>
    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。 <br>
    """)
gr.close_all()
# 直接启动
demo.launch()
```

通过将上述代码封装为 `/root/data/web_demo.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

5 结果展示

下图可以看出，问同一个问题，有时候可以生成正确答案，有时候却不行 /(ㄒoㄒ)/~~

比如问 “什么是InternLM"，如果先问一下相关问题铺垫一下，后续回答的效果可能不错；如果直接就问，答案就乱码了