

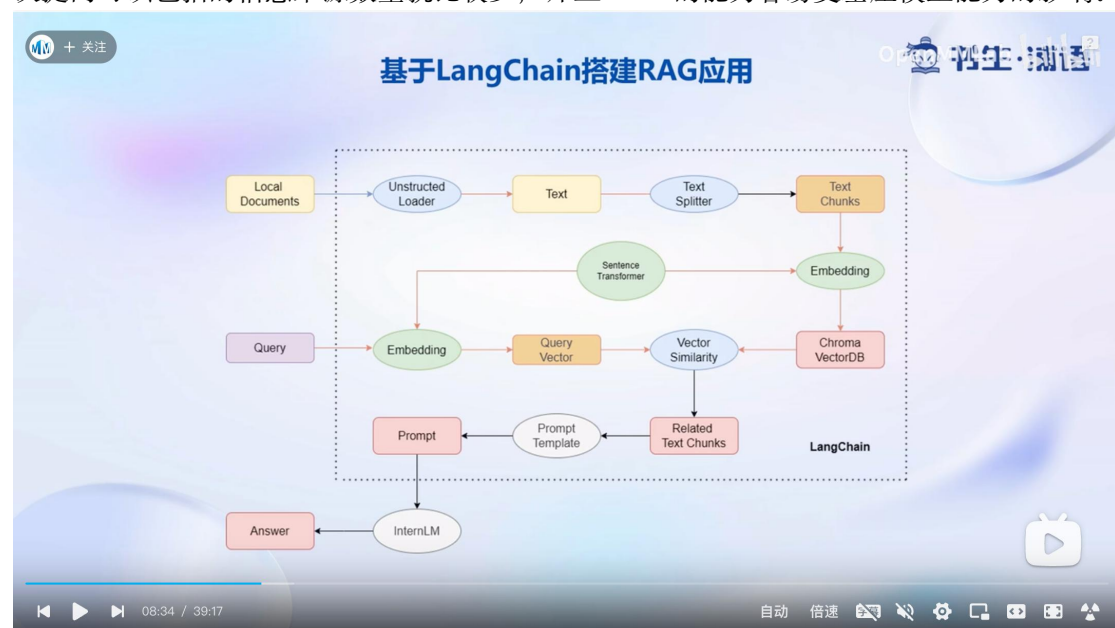
(3)基于 InternLM 和 LangChain 搭建你的知识库

基础作业:

复现课程知识库助手搭建过程 (截图)

首先我们回顾一下 RAG 的原理:

RAG 是将本地文档都转化为文本形式后, 使用 Embedding 将文本存储进向量数据库。在用户输入 prompt 以后, 将用户的 prompt 也转化成 embedding, 和向量数据库里的 embedding 进行匹配, 然后将用户的 prompt 和文本的 prompt 一起输入大模型, 得到模型的回答。RAG 可以低成本地实时更新大模型知识, 但是因为需要输入大量文本作为外部知识来源, 那么单次提问可以包括的信息来源数量就比较少, 并且 RAG 的能力容易受基座模型能力的影响。



1. 环境配置

其中 Internlm 模型部署和模型下载和上节课内容有重合, 在这里就不截图了。

这里贴一下 1.3 LangChain 相关环境配置的实现过程。

配置 langchain 用到了开源词向量模型 Sentence Transformer, 这里需要安装一下。

```
Help
< -> root
download_hf.py x doc2vec.py LLM.py run_gradio.py download.py
data > download_hf.py > ...
1 import os
2
3 # 设置环境变量
4 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com'
5
6 # 下载模型
7 os.system('huggingface-cli download --resume-download sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 --local-dir /root/data/model/sentence-transformer')
```

2. 知识库搭建

本小节分为 3 个子步骤:

- 将文件夹中格式为 txt 或 markdown 格式的文件路径找出来
- 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。
- 由纯文本对象构建向量数据库, 先对文本进行分块, 接着使用开源词向量模型 Sentence Transformer 对文本块进行向量化。选择 Chroma 作为向量数据库, 基于上文分块后的文档以及加载的开源向量化模型, 将语料加载到指定路径下的向量数据库

```
Help
< -> root
download_hf.py x doc2vec.py LLM.py run_gradio.py download.py
data > demo > doc2vec.py > ...
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 for filename in filenames: if filename.endswith(".md"): file_list.append(os.path.join(filepath, filename)) elif
18             if filename.endswith(".txt"): file_list.append(os.path.join(filepath, filename))
19             #
20             # Full name: data.demo.doc2vec.get_files.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_list = 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_list):
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
```

3. 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。实现的过程从 LangChain.llms.base.LLM 类继承一个子类, 并重写构造函数与 _call 函数即可

```
Help
< -> root
download_hf.py doc2vec.py LLM.py run_gradio.py download.py

data > demo > LLM.py > InternLM_LLM > _llm_type
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, trust_remote_code=True)
18        self.model = AutoModelForCausalLM.from_pretrained(model_path, trust_remote_code=True).to(torch.bfloat16).cuda()
19        self.model = self.model.eval()
20        print("完成本地模型的加载")
21
22    def _call(self, prompt: str, stop: Optional[List[str]] = None,
23             run_manager: Optional[CallbackManagerForLLMRun] = None,
24             **kwargs: Any):
25        # 重写调用函数
26        system_prompt = """You are an AI assistant whose name is InternLM (书生·浦语).
27        ~ 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
28        ~ InternLM (书生·浦语) can understand and communicate fluently in the language chosen by the user such as English and 中文.
29        """
30
31        messages = [(system_prompt, '')]
32        response, history = self.model.chat(self.tokenizer, prompt, history=messages)
33        return response
34
35    @property
36    def _llm_type(self) -> str:
37        return "InternLM"
```

4. 构建检索问答链

本小节分为以下几个步骤:

- 通过 Chroma 以及上文定义的词向量模型将上文构建的向量数据库导入进来
- 构建 Prompt Template, 该 Template 其实基于一个带变量的字符串, 在检索之后, LangChain 会将检索到的相关文档片段填入到 Template 的变量中, 从而实现带知识的 Prompt 构建。
- 调用 LangChain 提供的检索问答链构造函数 RetrievalQA.from_chain_type, 基于我们的自定义 LLM、Prompt Template 和向量知识库来构建一个基于 InternLM 的检索问答链

```
download_hf.py doc2vec.py LLM.py run_gradio.py X

data > demo > run_gradio.py > ...
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/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
24    llm = InternLM_LLM(model_path = "/root/data/model/Shanghai_AI_Laboratory/internlm-chat-7b")
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
39
40 class Model_center():
41    """
42    存储检索问答链的对象
43    """
44    def __init__(self):
45        # 构造函数, 加载检索问答链
```

5.demo 结果展示



可以看到，demo 已经部署成功，可以成功运行。

进阶作业：

选择一个垂直领域，收集该领域的专业资料构建专业知识库，并搭建专业问答助手，并在 OpenXLab 上成功部署（截图，并提供应用地址）

我选择了厨房菜谱领域搭建问答助手，代码在

<https://github.com/leonfrank/InternLM-HW/tree/main>

项目下面

应用地址在

https://openxlab.org.cn/apps/detail/chat-gpt/llm_cuisine

因为没有 GPU 资源，应用没有启动成功。待申请到硬件资源后更新。