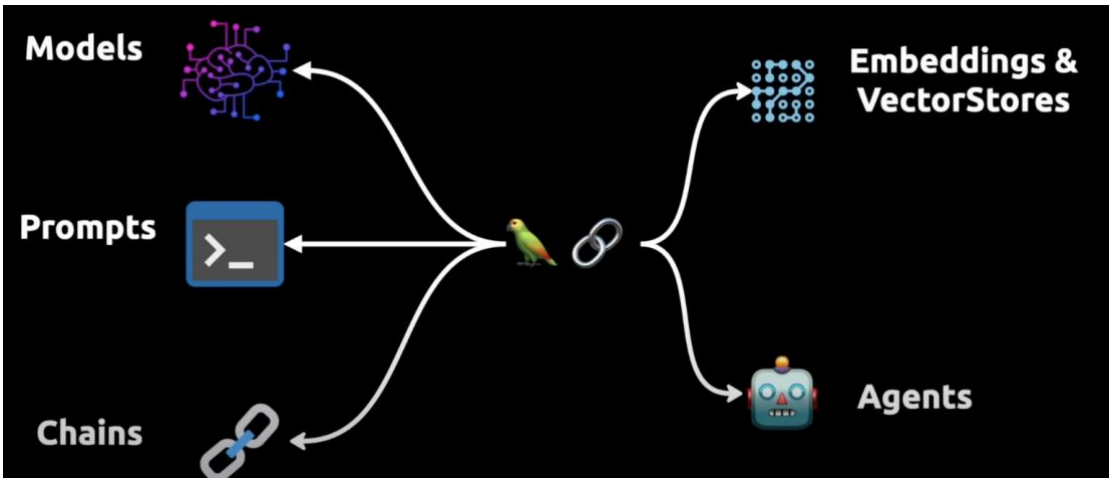
用 LLaMA 2 and LangChain 实现自有知识问答

背景知识

LangChain

LangChain 是一个开源框架，用于构建基于大型语言模型（LLM）的应用程序。LLM 是基于大量数据预先训练的大型深度学习模型，可以生成对用户查询的响应，例如回答问题或根据基于文本的提示创建图像。LangChain 提供各种工具和抽象，以提高模型生成的信息的定制性、准确性和相关性。例如，开发人员可以使用 LangChain 组件来构建新的提示链或自定义现有模板。LangChain 还包括一些组件，可让 LLM 无需重新训练即可访问新的数据集。



检索增强生成（RAG）

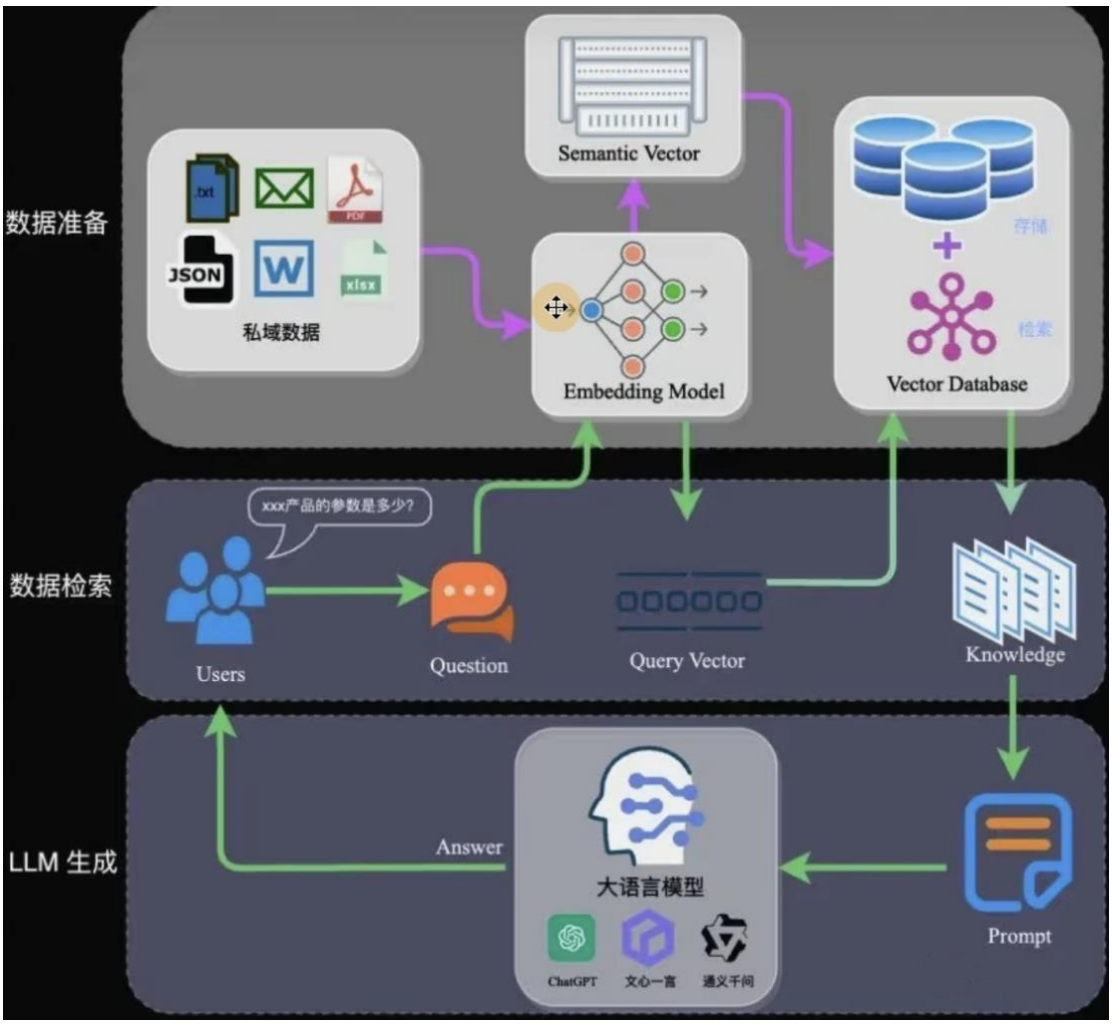
RAG 可以作为 LangChain（如果它是一个语言处理工具）的一部分，用于提供更加丰富和准确的语言生成能力。

检索增强生成（RAG）是指对大型语言模型输出进行优化，使其能够在生成响应之前引用训练数据来源之外的权威知识库。大型语言模型（LLM）用海量数据进行训练，使用数十亿个参数为回答问题、翻译语言和完成句子等任务生成原始输出。在 LLM 本就强大的功能基础上，RAG 将其扩展为能访问特定领域或组织的内部知识库，所有这些都无需重新训练模型。这是一种经济高效地改进 LLM 输出的方法，让它在各种情境下都能保持相关性、准确性和实用性。

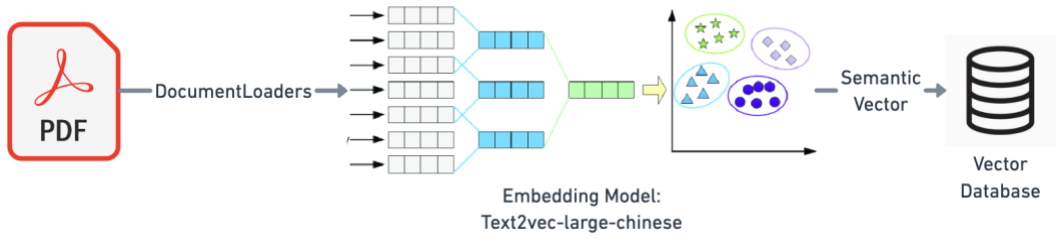
FAISS 向量库

FAISS（Facebook AI Similarity Search）是一个用于高效相似度搜索和密集向量聚类的库。它可以在标准数据库引擎（SQL）无法或效率低下地搜索多媒体文档（如图像）的情况下进行搜索。它包含了能够在可能不适用于 RAM 的任意大小的向量集合中进行搜索的算法。它还包含评估和支持代码参数调整。

处理流程



外挂知识库流程



- 加载：

首先，我们需要加载数据。我们使用 DocumentLoaders 来实现这一点。

- 分割：

文本分割器将大的 Documents 分割为较小的块。这对于索引数据和传递给模型都很有用，因为大块的数据较难搜索，而且无法在模型的有限上下文窗口中使用。

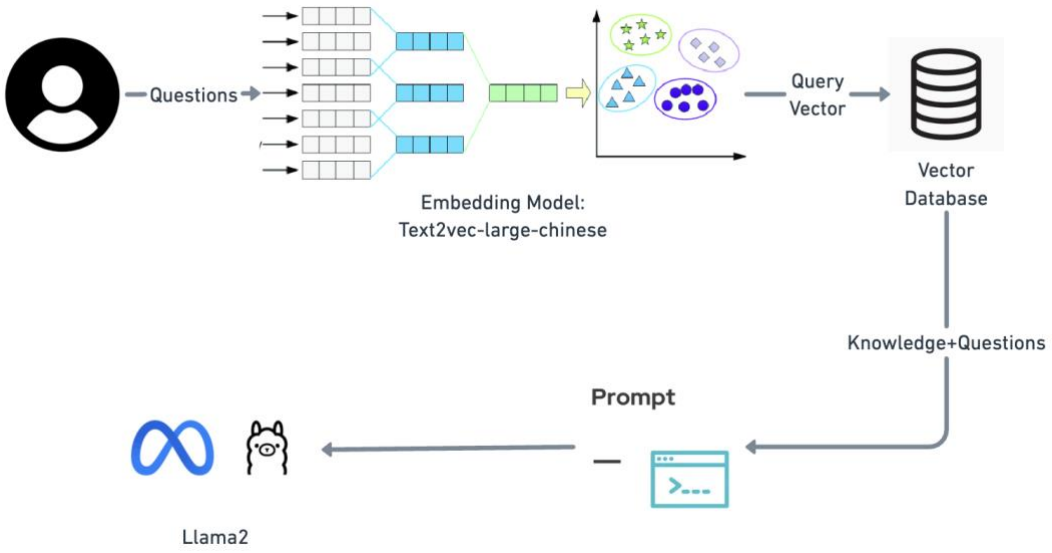
- 嵌入：

将文本块转换为数值（向量）表示，也称为嵌入。这些嵌入用于在大型数据库中快速搜索和检索类似或相关的文档，因为它们代表了文本的语义含义。这里我们使用 Text2vec-large-chinese 模型来实现。

- 存储:

将嵌入加载到向量存储（向量数据库）中：将嵌入加载到向量存储（在这种情况下是“FAISS”）中。与传统数据库相比，向量存储在基于文本嵌入的相似性搜索方面表现出色。

检索增强生成流程



- 检索：

首先，我们需要进行的是检索过程。在这个阶段，我们利用用户的查询内容，从外部知识源获取相关信息。具体来说，就是将用户的查询通过嵌入模型转化为向量，这样就可以与向量数据库中的其他上下文信息进行比对。通过这种相似性搜索，我们可以找到向量数据库中最匹配的前 k 个数据。

- 增强：

接下来，我们进入增强阶段。在这个阶段，我们将用户的查询和检索到的额外信息一起嵌入到一个预设的提示模板中。这个过程的目的是为了提供更丰富、更具上下文的信息以便于后续的生成过程。

- 生成：

最后，我们进行生成过程，在这个阶段，我们将经过检索增强的提示内容输入到大语言模型(Llama2)中，以生成所需的输出。这个过程是 RAG 的核心，它利用了 LLM 的强大生成能力，结合了前两个阶段的信息，生成了准确、丰富且与上下文相关的输出。

技术实现流程

1、设置知识库

```
Plain Text
from langchain.document_loaders import UnstructuredFileLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
from langchain.vectorstores import FAISS
```

知识库其实本质上就是一个 pdf。这里我们使用 unstructured 这个包，把带格式的文本，读取为无格式的纯文本

```
Plain Text
filepath="/root/autodl-tmp/knowledge.pdf"
loader=UnstructuredFileLoader(filepath)
docs=loader.load()
#打印一下看看，返回的是一个列表，列表中的元素是 Document 类型
print(docs)
```

对上面读取的文档进行 chunk。chunk 的意义在于，可以把长文本拆分成小段，以便搜索召回

chunk-size 是文本最大的字符数。chunk-overlap 是前后两个 chunk 的重叠部分最大字数

```
Plain Text
text_splitter=RecursiveCharacterTextSplitter(chunk_size=20,chunk_o
verlap=10)
docs=text_splitter.split_documents(docs)
```

下载并部署 embedding 模型

使用 text2vec-large-chinese 模型，对上面 chunk 后的 doc 进行 embedding。然后使用 FAISS 存储到向量数据库

```
Plain Text
import os
embeddings=HuggingFaceEmbeddings(model_name="/root/autodl-
tmp/text2vec-large-chinese", model_kwargs={'device': 'cuda'})
```

```
#如果之前没有本地的 faiss 仓库，就把 doc 读取到向量库后，再把向量库保存到本地
if os.path.exists("/root/autodl-tmp/my_faiss_store.faiss")==False:
    vector_store=FAISS.from_documents(docs,embeddings)
    vector_store.save_local("/root/autodl-tmp/my_faiss_store.faiss")
#如果本地已经有 faiss 仓库了，说明之前已经保存过了，就直接读取
else:
    vector_store=FAISS.load_local("/root/autodl-tmp/my_faiss_store.faiss",embeddings=embeddings)
```

2、加载模型

```
Plain Text
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
#先做 tokenizer
tokenizer = AutoTokenizer.from_pretrained('/root/autodl-tmp/Llama2-chat-13B-Chinese-50W',trust_remote_code=True)
#加载本地基础模型
#low_cpu_mem_usage=True,
#load_in_8bit="load_in_8bit",
base_model = AutoModelForCausalLM.from_pretrained(
        "/root/autodl-tmp/Llama2-chat-13B-Chinese-50W",
        torch_dtype=torch.float16,
        device_map='auto',
        trust_remote_code=True
    )
model=base_model.eval()
```

3、知识库的使用和构造 prompt_template

通过用户问句，到向量库中，匹配相似度高的【知识】

```
Plain Text
query="DataElit 是什么？"
docs=vector_store.similarity_search(query)#计算相似度，并把相似度高的 chunk 放在前面
context=[doc.page_content for doc in docs]#提取 chunk 的文本内容
print(context)
```

编写 prompt_template；把查找到的【知识】和【用户的问题】都输入到 prompt_template 中

```
Plain Text
my_input="\n".join(context)
prompt=f"已知：\n{my_input}\n 请回答：{query}"
print(prompt)
```

4、把 prompt 输入模型进行预测

```
Plain Text
inputs = tokenizer([f"Human:{prompt}\nAssistant:"],
return_tensors="pt")
input_ids = inputs["input_ids"].to('cuda')

generate_input = {
    "input_ids":input_ids,
    "max_new_tokens":1024,
    "do_sample":True,
    "top_k":50,
    "top_p":0.95,
    "temperature":0.3,
    "repetition_penalty":1.3
}
generate_ids  = model.generate(**generate_input)

new_tokens = tokenizer.decode(generate_ids[0],
skip_special_tokens=True)
print("new_tokens",new_tokens)
```

5、模型输出

```
Plain Text
DataElit 数据平台是集数据集成、数据开发、数据管理、数据治理于一体，运用成熟、先进的技术，自主研发的数据管理平台（以下简称 DataElit）。 DataElit 用一站式开发管理的界面，帮助企业专注于数据价值的挖掘和探索。
```