大模型（LLMs）langchain 面

来自： AiGC面试宝典

宁静致远 2023年09月16日 21:19

1. 什么是 LangChain?

LangChain是一个强大的框架，旨在帮助开发人员使用语言模型构建端到端的应用程序。它提供了一套工具、组件和接口，可简化创建由大型语言模型 (LLM) 和聊天模型提供支持的应用程序的过程。LangChain 可以轻松管理与语言模型的交互，将多个组件链接在一起，并集成额外的资源，例如 API 和数据库。

2. LangChain 包含哪些 核心概念?

2.1 LangChain 中 Components and Chains 是什么?

- **Component**： **模块化的构建块**，可以组合起来创建强大的应用程序；
- **Chain**： **组合在一起以完成特定任务的一系列 Components**（或其他 Chain）；

> 注：一个 Chain 可能包括一个 Prompt 模板、一个语言模型和一个输出解析器，它们一起工作以处理用户输入、生成响应并处理输出。

2.2 LangChain 中 Prompt Templates and Values 是什么?

- **Prompt Template 作用**：**负责创建 PromptValue**，这是最终传递给语言模型的内容
- **Prompt Template 特点**：有助于将用户输入和其他动态信息转换为适合语言模型的格式。PromptValues 是具有方法的类，这些方法可以转换为每个模型类型期望的确切输入类型（如文本或聊天消息）。

2.3 LangChain 中 Example Selectors 是什么?

- 作用：当您想要在 Prompts 中动态包含示例时，Example Selectors 很有用。他们**接受用户输入并返回一个示例列表以在提示中使用，使其更强大和特定于上下文**。

2.4 LangChain 中 Output Parsers 是什么?

- 作用： **负责将语言模型响应构建为更有用的格式**
- 实现方法：
    - 一种**用于提供格式化指令**
    - 另一种**用于将语言模型的响应解析为结构化格式**
- 特点：使得在您的应用程序中处理输出数据变得更加容易。

2.5 LangChain 中 Indexes and Retrievers 是什么?

Index： 一种组织文档的方式，使语言模型更容易与它们交互；

Retrievers：用于获取相关文档并将它们与语言模型组合的接口；

注：LangChain 提供了用于处理不同类型的索引和检索器的工具和功能，例如矢量数据库和文本拆分器。

2.6 LangChain 中 Chat Message History 是什么?

- Chat Message History 作用：负责记住所有以前的聊天交互数据，然后可以将这些交互数据传递回模型、汇总或以其他方式组合；
- 优点：有助于维护上下文并提高模型对对话的理解

2.7 LangChain 中 Agents and Toolkits 是什么?

- Agent： 在 LangChain 中推动决策制定的实体。他们可以访问一套工具，并可以根据用户输入决定调用哪个工具；
- Tookits： 一组工具，当它们一起使用时，可以完成特定的任务。代理执行器负责使用适当的工具运行代理。

通过理解和利用这些核心概念，您可以利用 LangChain 的强大功能来构建适应性强、高效且能够处理复杂用例的高级语言模型应用程序。

3. 什么是 LangChain Agent?

- 介绍：LangChain Agent 是框架中**驱动决策制定的实体**。它可以**访问一组工具，并可以根据用户的输入决定调用哪个工具**；

- 优点：LangChain Agent 帮助构建复杂的应用程序，这些应用程序需要自适应和特定于上下文的响应。当存在取决于用户输入和其他因素的未知交互链时，它们特别有用。

4. 如何使用 LangChain ?

要使用 LangChain，开发人员首先要导入必要的组件和工具，例如 LLMs, chat models, agents, chains, 内存功能。这些组件组合起来创建一个可以理解、处理和响应用户输入的应用程序。

5. LangChain 支持哪些功能?

- **针对特定文档的问答**：根据给定的文档回答问题，使用这些文档中的信息来创建答案。
- **聊天机器人**：构建可以利用 LLM 的功能生成文本的聊天机器人。
- **Agents**：开发可以决定行动、采取这些行动、观察结果并继续执行直到完成的代理。

6. 什么是 LangChain model?

LangChain model 是一种抽象，表示框架中使用的不同类型的模型。LangChain 中的模型主要分为三类：

1. **LLM（大型语言模型）**：这些模型将文本字符串作为输入并返回文本字符串作为输出。它们是许多语言模型应用程序的支柱。
2. **聊天模型(Chat Model)**：聊天模型由语言模型支持，但具有更结构化的 API。他们将聊天消息列表作为输入并返回聊天消息。这使得管理对话历史记录和维护上下文变得容易。
3. **文本嵌入模型(Text Embedding Models)**：这些模型将文本作为输入并返回表示文本嵌入的浮点列表。这些嵌入可用于文档检索、聚类和相似性比较等任务。

开发人员可以为他们的用例选择合适的 LangChain 模型，并利用提供的组件来构建他们的应用程序。

7. LangChain 包含哪些特点?

LangChain 旨在为六个主要领域的开发人员提供支持：

1. **LLM 和提示**：LangChain 使管理提示、优化它们以及为所有 LLM 创建通用界面变得容易。此外，它还包括一些用于处理 LLM 的便捷实用程序。
2. **链(Chain)**：这些是对 LLM 或其他实用程序的调用序列。LangChain 为链提供标准接口，与各种工具集成，为流行应用提供端到端的链。
3. **数据增强生成**：LangChain 使链能够与外部数据源交互以收集生成步骤的数据。例如，它可以帮助总结长文本或使用特定数据源回答问题。
4. **Agent**s：Agents 让 LLM 做出有关行动的决定，采取这些行动，检查结果，并继续前进直到工作完成。LangChain 提供了代理的标准接口，多种代理可供选择，以及端到端的代理示例。
5. **内存**：LangChain 有一个标准的内存接口，有助于维护链或代理调用之间的状态。它还提供了一系列内存实现和使用内存的链或代理的示例。
6. **评估**：很难用传统指标评估生成模型。这就是为什么 LangChain 提供提示和链来帮助开发者自己使用 LLM 评估他们的模型。

8. LangChain 如何使用?

8.1 LangChain 如何调用 LLMs 生成回复?

Models: 指各类训练好大语言模型（eg: chatgpt(未开源)，chatglm，vicuna等）

```
# 官方llm使用OPENAI 接口
from langchain.llms import OpenAI
llm = OpenAI(model_name="text-davinci-003")
prompt = "你好"
response = llm(prompt)
# 你好，我是chatGPT,很高兴能够和你聊天。有什么我可以帮助你的吗？

-我们用chatglm来演示该过程，封装一下即可
from transformers import AutoTokenizer, AutoModel
class chatGLM():
    def __init__(self, model_name) -> None:
        self.tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
        self.model = AutoModel.from_pretrained(model_name,
trust_remote_code=True).half().cuda().eval()

    def __call__(self, prompt) -> Any:
```

```
        response, _ = self.model.chat(self.tokenizer , prompt) # 这里演示未使用流式接口.
stream_chat()
        return response

llm =  chatGLM(model_name="THUDM/chatglm-6b")
prompt = "你好"
response = llm(prompt)
print("response: %s"%response)
“” “
response: 你好 ！我是人工智能助手 ChatGLM-6B，很高兴见到你，欢迎问我任何问题。
” “”
```

8.2 LangChain 如何修改 提示模板?

langchain.PromptTemplate: langchain中的提示模板类

根据不同的下游任务设计不同的prompt模板，然后填入内容，生成新的prompt。目的其实就是为了通过设计更准确的提示词，来引导大模型输出更合理的内容。

```
from langchain import PromptTemplate
template = """
Explain the concept of {concept} in couple of lines
"""
prompt = PromptTemplate(input_variables=["concept"], template=template)
prompt = prompt.format(concept="regularization")
print(“prompt=%s” %prompt)
#'\nExplain the concept of regularization in couple of lines\n'

-------------------------
template = "请给我解释一下{concept}的意思"
prompt = PromptTemplate(input_variables=["concept"], template=template)
prompt = prompt.format(concept="人工智能")
print(“prompt=%s” %prompt)
#'\n请给我解释一下人工智能的意思\n'
```

8.3 LangChain 如何链接多个组件处理一个特定的下游任务?

```
#chains ---------
from langchain.chains import LLMChain
chain = LLMChain(llm=openAI(), prompt=promptTem)
print(chain.run("你好"))

#chains ---------Chatglm对象不符合LLMChain类llm对象要求，模仿一下
class DemoChain():
    def __init__(self, llm, prompt) -> None:
        self.llm = llm
        self.prompt = prompt

    def run(self, query) -> Any:
        prompt = self.prompt.format(concept=query)
        print("query=%s  ->prompt=%s"%(query, prompt))
        response = self.llm(prompt)
        return response

chain = DemoChain(llm=llm, prompt=promptTem)
print(chain.run(query="天道酬勤"))

“” “
```

```
query=天道酬勤  ->prompt=请给我解释一下天道酬勤的意思
天道酬勤是指自然界的规律认为只要一个人勤奋努力，就有可能会获得成功。这个成语的意思是说，尽管一个人可能需要付出很多努力才能取得成功，但只要他/她坚持不懈地努力，就有可能会得到回报。
” “”
```

8.4 LangChain 如何Embedding & vector store?

Emebdding这个过程想必大家很熟悉，简单理解就是把现实中的信息通过各类算法编码成一个高维向量，便于计算机快速计算。

1. DL的语言模型建模一般开头都是word embedding，看情况会加position embedding。比如咱们的LLM的建模
2. 常规检索一般是把refernce数据都先Embedding入库，服务阶段query进来Embedding后再快速在库中查询相似topk。比如langchain-chatGLM的本地知识库QA系统的入库和检测过程。
3. 多模态的方案：同时把语音，文字，图片用不同的模型做Embedding后，再做多模态的模型建模和多模态交互。比如这两天的Visual-chatGLM.

```
#官方示例代码，用的OpenAI的ada的文本Embedding模型
#1） Embeding model
from langchain.embeddings import OpenAIEmbeddings
embeddings = OpenAIEmbeddings(model_name="ada")
query_result = embeddings.embed_query("你好")

#2) 文本切割
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100, chunk_overlap=0
)
texts = """天道酬勤”并不是鼓励人们不劳而获，而是提醒人们要遵循自然规律，通过不断的努力和付出来追求自己的目标。\n这种努力不仅仅是指身体上的劳动，
也包括精神上的努力和思考，以及学习和适应变化的能力。\n只要一个人具备这些能力，他就有可能会获得成功。"""
texts = text_splitter.create_documents([texts])
print(texts[0].page_content)

# 3)入库检索，官方使用的Pinecone,他提供一个后台管理界面 | 用户需求太大，不好用了已经，一直加载中....
import pinecone
from langchain.vectorstores import Pinecone
pinecone.init(api_key=os.getenv(""), enviroment=os.getenv(""))

index_name = "demo"
search = Pinecone.from_documents(texts=texts, embeddings, index_name=index_name)
query = "What is magical about an autoencoder?"
result = search.similarity_search(query)

#----------------------------------------------这里参考langchain-chatglm代码
# 1） Embedding model:  text2vec-large-chinese
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
embeddings = HuggingFaceEmbeddings(model_name="GanymedeNil/text2vec-large-chinese",
                                   model_kwargs={'device': "cuda"})
query_result = embeddings.embed_query("你好")

#2)文本分割， 这里仅为了方便快速看流程，实际应用的会复杂一些
texts = """天道酬勤”并不是鼓励人们不劳而获，而是提醒人们要遵循自然规律，通过不断的努力和付出来追求自己的目标。\n这种努力不仅仅是指身体上的劳动，
也包括精神上的努力和思考，以及学习和适应变化的能力。\n只要一个人具备这些能力，他就有可能会获得成功。"""
from langchain.text_splitter import CharacterTextSplitter
from langchain.docstore.document import Document
class TextSpliter(CharacterTextSplitter):
```

```
    def __init__(self, separator: str = "\n\n", **kwargs: Any):
        super().__init__(separator, **kwargs)
    def split_text(self, text: str) -> List[str]:

        texts = text.split("\n")
        texts = [Document(page_content=text, metadata={" from ": "知识库.txt"}) for text in texts]
        return texts

text_splitter = TextSpliter()
texts = text_splitter.split_text(texts)

#3) 直接本地存储
vs_path = "./demo-vs"
from langchain.vectorstores import FAISS
docs = embeddings.embed_documents(sentences)
vector_store = FAISS.from_documents(texts, embeddings)
vector_store.save_local(vs_path)

vector_store = FAISS.load_local(vs_path, embeddings)
related_docs_with_score = vector_store.similarity_search_with_score(query, k=2)
```

LangChain 存在哪些问题及方法方案?

1. LangChain 低效的令牌使用问题

- 问题：Langchain的一个重要问题是它的令牌计数功能，对于小数据集来说，它的效率很低。虽然一些开发人员选择创建自己的令牌计数函数，但也有其他解决方案可以解决这个问题。
- 解决方案：Tiktoken是OpenAI开发的Python库，用于更有效地解决令牌计数问题。它提供了一种简单的方法来计算文本字符串中的令牌，而不需要使用像Langchain这样的框架来完成这项特定任务。

2. LangChain 文档的问题

- 问题：文档是任何框架可用性的基石，而Langchain因其不充分且经常不准确的文档而受到指责。误导性的文档可能导致开发项目中代价高昂的错误，并且还经常有404错误页面。这可能与Langchain还在快速发展有关，作为快速的版本迭代，文档的延后性问题

3. LangChain 太多概念容易混淆，过多的“辅助”函数问题

- 问题：Langchain的代码库因很多概念让人混淆而备受批评，这使得开发人员很难理解和使用它。这种问题的一个方面是存在大量的“helper”函数，仔细检查就会发现它们本质上是标准Python函数的包装器。开发人员可能更喜欢提供更清晰和直接访问核心功能的框架，而不需要复杂的中间功能。

简单的分割函数：

```
class CharacterTextSplitter(TextSplitter):
    """Implementation of splitting text that looks at characters."""

def __init__(self, separator: str = "\n\n", **kwargs: Any):
    """Create a new TextSplitter."""
    super().__init__(**kwargs)
    self._separator = separator

def split_text(self, text: str) -> List[str]:
    """Split incoming text and return chunks."""
    # First we naively split the large input into a bunch of smaller ones.
    if self._separator:
        splits = text.split(self._separator)
    else:
        splits = list(text)
```


4. LangChain 行为不一致并且隐藏细节问题

- 问题：LangChain因隐藏重要细节和行为不一致而受到批评，这可能导致生产系统出现意想不到的问题。

eg: Langchain ConversationRetrievalChain的一个有趣的方面，它涉及到输入问题的重新措辞。这种重复措辞有时会非常广泛，甚至破坏了对话的自然流畅性，使对话脱离了上下文。

5. LangChain 缺乏标准的可互操作数据类型问题

- 问题：缺乏表示数据的标准方法。这种一致性的缺乏可能会阻碍与其他框架和工具的集成，使其在更广泛的机器学习工具生态系统中工作具有挑战性。

LangChain 替代方案?

是否有更好的替代方案可以提供更容易使用、可伸缩性、活动性和特性。

- LlamaIndex是一个数据框架，它可以很容易地将大型语言模型连接到自定义数据源。它可用于存储、查询和索引数据，还提供了各种数据可视化和分析工具。
- Deepset Haystack是另外一个开源框架，用于使用大型语言模型构建搜索和问答应用程序。它基于Hugging Face Transformers，提供了多种查询和理解文本数据的工具。