

Relatório 9 - Prático: Introdução a LangChain (III)

Vitor Eduardo de Lima Kenor

Descrição da atividade

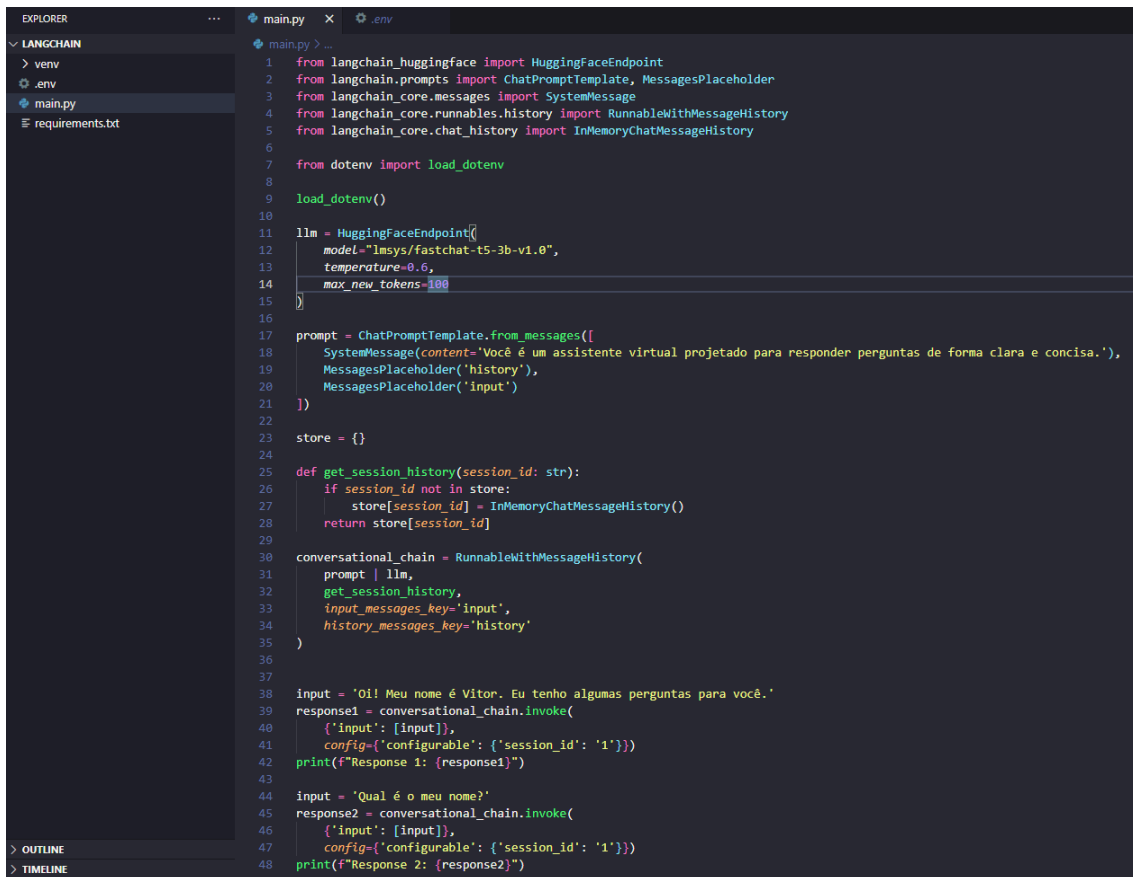
Começamos o card com um vídeo que introduz LangChain, que é uma biblioteca que facilita a criação de aplicações com modelos de linguagem de grande escala. No vídeo é explicado que a LangChain agiliza a programação de aplicativos LLM por meio de abstrações. As abstrações do LangChain representam componentes e conceitos comuns necessários para trabalhar com modelos de linguagem. O primeiro componente é a LLM wrappers, quase qualquer LLM pode ser usada no LangChain sendo só necessário uma chave de API. A classe LLM foi projetada para fornecer uma interface padrão para todos os modelos. O segundo componente são os Prompts, que se tratam das instruções fornecidas à LLM. A classe de modelo de Prompt em LangChain formaliza a composição dos prompts, sem a necessidade de codificar manualmente o contexto e as consultas. O terceiro componente são as Chains, se tratam do núcleo dos fluxos de trabalho do LangChain, combinam LLMs com outros componentes. O quarto componente se trata de Índices, que são estruturas que organizam e armazenam dados textuais para otimizar buscas e consultas em LLMs. Dentro desse componente possuem alguns tópicos sendo document loaders, Vector DB e Text Splitters. No final do vídeo vemos alguns casos de uso onde o LangChain onde colocamos tudo o que aprendemos para funcionar em aplicativos. Os exemplos dados foram: Chatbots, Summarization, Question Answering, Data Augmentation e Virtual Agents.

No segundo vídeo presente no card temos algumas explicações sobre o LangChain, o necessário para que possamos entender sua estrutura. Depois da explicação ele vai direto para a prática, primeiro ele pega a chave de API da OpenAi e depois ele configura o ambiente virtual do projeto. Depois de configurar o ambiente aprendemos a criar um prompt para que a LLM utilize.



```
1 from langchain_openai import ChatOpenAI
2 from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder
3 from dotenv import load_dotenv
4
5 load_dotenv()
6
7 llm = ChatOpenAI(model='gpt-4o-mini', temperature= 0.6)
8
9 prompt = ChatPromptTemplate.from_template(
10     'Você tem um {animal_type} filhote com a cor {color} novo e gostaria de dar um nome legal a ele. me de uma lista de 5 possíveis nomes.'
11 )
12
13 response = llm(prompt.format(animal_type="cachorro", color="marrom"))
14
15 print(response)
```

O terceiro vídeo do card é um tutorial que ensina como criar chatbots que podem responder perguntas baseando-se no histórico de conversas. Utilizando as bibliotecas Langchain e Hugging Face, ele mostra na prática a implementação de memória em chatbots, permitindo que eles aprendam e personalizem as conversas baseadas nas interações anteriores.



```
1 from langchain.huggingface import HuggingFaceEndpoint
2 from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder
3 from langchain_core.messages import SystemMessage
4 from langchain_core.runnables.history import RunnableWithMessageHistory
5 from langchain_core.chat_history import InMemoryChatMessageHistory
6
7 from dotenv import load_dotenv
8
9 load_dotenv()
10
11 llm = HuggingFaceEndpoint(
12     model="lmstudio/fastchat-t5-3b-v1.0",
13     temperature=0.6,
14     max_new_tokens=100
15 )
16
17 prompt = ChatPromptTemplate.from_messages([
18     SystemMessage(content='Você é um assistente virtual projetado para responder perguntas de forma clara e concisa.'),
19     MessagesPlaceholder('history'),
20     MessagesPlaceholder('input')
21 ])
22
23 store = {}
24
25 def get_session_history(session_id: str):
26     if session_id not in store:
27         store[session_id] = InMemoryChatMessageHistory()
28     return store[session_id]
29
30 conversational_chain = RunnableWithMessageHistory(
31     prompt | llm,
32     get_session_history,
33     input_messages_key='input',
34     history_messages_key='history'
35 )
36
37 input = 'Olá! Meu nome é Vitor. Eu tenho algumas perguntas para você.'
38 response1 = conversational_chain.invoke(
39     {'input': [input]},
40     config={'configurable': {'session_id': '1'}}
41 )
42 print(f'Response 1: {response1}')
43
44 input = 'Qual é o meu nome?'
45 response2 = conversational_chain.invoke(
46     {'input': [input]},
47     config={'configurable': {'session_id': '1'}}
48 )
49 print(f'Response 2: {response2}')
```

Conclusões

A Partir deste card conseguimos ter uma noção de como a biblioteca LangChain funciona e toda sua estrutura. Nos vídeos em que vamos para o código conseguimos ter uma noção de como podemos implementar as LLMs em nossos algoritmos utilizando LangChain. Vale observar que muitas das funções já mudaram pois a biblioteca está em constantes atualizações. Por isso é importante rodar os ambientes virtuais para conseguir rodar a versão que se deseja.

Referências

- [Vídeo: Introdução ao LangChain](#)
- [Tutorial básico de Langchain](#)
- [Memory using Langchain](#)