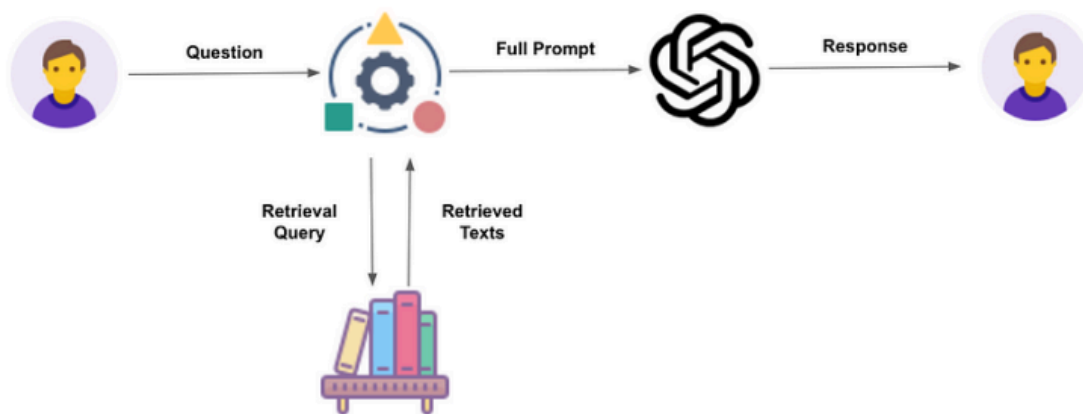


Relatório 11 - Prática: Introdução RAG (III)

Vitor Eduardo de Lima Kenor

Descrição da atividade

No primeiro vídeo do card temos a explicação sobre o que se trata o RAG (Retrieval-Augmented Generation). Através de exemplos é explicado que o RAG é o processo de otimizar a saída de um grande modelo de linguagem, de forma que ele faça referência a uma base de conhecimento confiável fora das suas fontes de dados de treinamento antes de gerar uma resposta. A RAG estende os já poderosos recursos dos LLMs para domínios específicos ou para a base de conhecimento interna de uma organização sem a necessidade de treinar novamente o modelo.



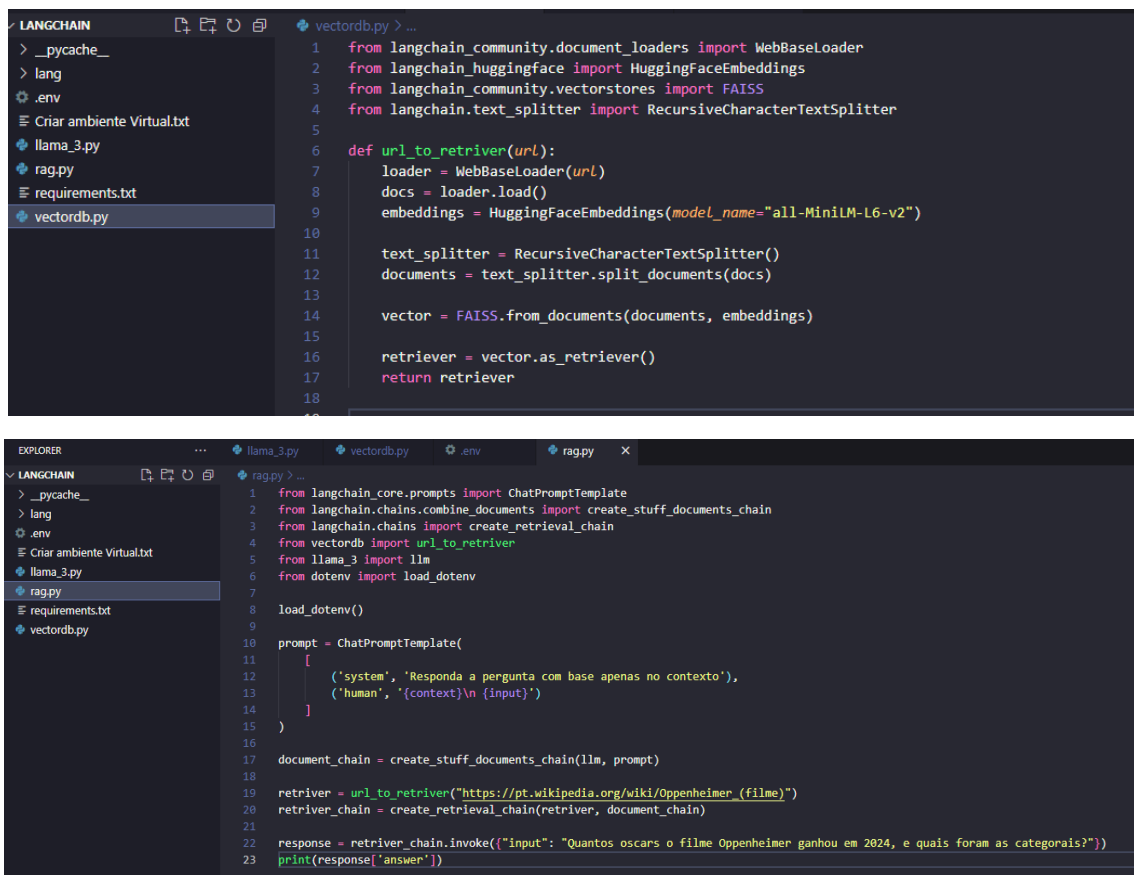
A tecnologia RAG também permite que a equipe de desenvolvimento teste e aprimore os recursos de conversação ou chat de forma mais eficiente. É possível gerenciar e modificar as fontes de informação do LLM para adequá-las a necessidades mutáveis ou para uso multifuncional.

No Segundo vídeo do card colocamos a mão na massa para aplicarmos o RAG em um caso onde queremos saber de uma informação atual, o modelo que usaremos não tem essa informação por se tratar de algo atual. No exemplo do vídeo ele usa o modelo da openAI gpt-3.5-turbo-instruct, como eu não possuo créditos na API da openAI, decidi usar o llama-3.1-8b-instant.

```
EXPLORER
└─ LANGCHAIN
  ├── __pycache__
  ├── lang
  ├── .env
  ├── Criar ambiente Virtual.txt
  ├── llama_3.py
  ├── rag.py
  ├── requirements.txt
  └── vectordb.py

llama_3.py
1 from dotenv import load_dotenv
2 from langchain_groq import ChatGroq
3 from langchain.prompts import PromptTemplate
4
5 load_dotenv()
6
7 def oscar(filme, ano, llm):
8     prompt = PromptTemplate(
9         input_variables=['filme', 'ano'],
10        template="Quanto oscar filme {filme} ganhou em {ano}"
11    )
12    oscar_chain = prompt | llm
13
14    response = oscar_chain.invoke({'filme': filme, 'ano': ano})
15
16    return response
17
18 llm = ChatGroq(
19     model="llama-3.1-8b-instant",
20     temperature=0,
21     max_tokens=8000,
22     timeout=1440,
23     max_retries=2
24 )
25
26 response = oscar('Oppenheimer', 2024, llm)
27 print(response.content)
```

Primeiramente importamos o modelo e verificamos a resposta, e como esperado ele não sabe nos responder por se tratar de uma pergunta atual que não esteve em seu treinamento. Então aplicamos o RAG para que ele consiga nos responder.



```
vectordb.py > ...
1 from langchain_community.document_loaders import WebBaseLoader
2 from langchain_huggingface import HuggingFaceEmbeddings
3 from langchain_community.vectorstores import FAISS
4 from langchain.text_splitter import RecursiveCharacterTextSplitter
5
6 def url_to_retriever(url):
7     loader = WebBaseLoader(url)
8     docs = loader.load()
9     embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
10
11     text_splitter = RecursiveCharacterTextSplitter()
12     documents = text_splitter.split_documents(docs)
13
14     vector = FAISS.from_documents(documents, embeddings)
15
16     retriever = vector.as_retriever()
17     return retriever
18
```

```
rag.py > ...
1 from langchain_core.prompts import ChatPromptTemplate
2 from langchain.chains.combine_documents import create_stuff_documents_chain
3 from langchain.chains import create_retrieval_chain
4 from vectordb import url_to_retriever
5 from llama_3 import llm
6 from dotenv import load_dotenv
7
8 load_dotenv()
9
10 prompt = ChatPromptTemplate(
11     [
12         ('system', 'Responda a pergunta com base apenas no contexto'),
13         ('human', '{context}\n {input}')
14     ]
15 )
16
17 document_chain = create_stuff_documents_chain(llm, prompt)
18
19 retriever = url_to_retriever("https://pt.wikipedia.org/wiki/Oppenheimer_(filme)")
20 retriever_chain = create_retrieval_chain(retriever, document_chain)
21
22 response = retriever_chain.invoke({"input": "Quantos oscar o filme Oppenheimer ganhou em 2024, e quais foram as categorias?"})
23 print(response['answer'])
```

Depois que aplicamos o RAG fornecendo o documento com as informações necessárias, o modelo consegue nos responder através dos dados a ele fornecidos.

O artigo discute um modelo híbrido de geração de linguagem natural, chamado RAG, que mescla memórias paramétricas e não paramétricas. Este modelo foi criado com o objetivo de aprimorar o desempenho em tarefas que requerem acesso a informações factuais e são intensivas em conhecimento. O modelo emprega uma memória paramétrica, inspirada no BART, e uma memória não paramétrica, que acessa dados de um índice vetorial denso da Wikipédia. O desempenho do modelo foi excelente em responder perguntas e gerar questões para o jogo "Jeopardy", sendo escolhido pelos avaliadores por oferecer respostas mais detalhadas. O RAG permite a adição de novos conhecimentos sem necessidade de novo treinamento, apenas substituindo a base de dados de recuperação. Ao usar fontes de informação comprovadas, o RAG se torna menos suscetível a ilusões, o que o torna benéfico para campos como a medicina e áreas que requerem exatidão. No entanto, o modelo também encontra obstáculos relacionados a desvios e precisão devido às restrições da fonte externa.

Conclusões

O RAG se sobressai como uma maneira inovadora de aumentar a eficácia dos LLMs, possibilitando o acesso a dados atualizados e precisos sem a necessidade de treinamento adicional. Esta estratégia não apenas melhora o desempenho dos modelos em várias situações, mas também apresenta uma maneira versátil de incluir novos dados, ajustando-se a contextos e necessidades variadas. Contudo, é importante considerar as restrições das fontes externas ao implementar o RAG para assegurar a exatidão e a confiabilidade das respostas produzidas.

Referências

- [Introdução ao RAG](#)
- [Langchain - RAG](#)
- <https://aws.amazon.com/pt/what-is/retrieval-augmented-generation/>