Excited to share my latest article on leveraging the power of GPT4All and Langchain to enhance document-based conversations! In this post, I walk you through the steps to set up the environment and demonstrate how you can seamlessly chat with your own documents using advanced language models. Get ready to unlock new possibilities and streamline your document interactions. Let’s dive in!

**Packages need to install**

# Install langchain  
pip install langchain  
  
# Install vectorStore  
pip install faiss-cpu  
  
# Install gpt4all  
pip install gpt4all  
  
# Install huggingfaceHub  
pip install huggingface-hub  
  
# Install PyPdf for working with PDFs  
pip install pypdf

**After downloading the gpt4all model**

Note: to download llm follow these links  
[Alpaca-native-7b](https://huggingface.co/Pi3141/alpaca-native-7B-ggml/tree/397e872bf4c83f4c642317a5bf65ce84a105786e)

Import the necessary classes into your Python file.

from langchain.document\_loaders import PyPDFLoader  
from langchain import PromptTemplate, LLMChain  
from langchain.embeddings import HuggingFaceEmbeddings  
from langchain.llms import GPT4All  
from langchain.text\_splitter import RecursiveCharacterTextSplitter  
from langchain.vectorstores.faiss import FAISS  
from langchain.callbacks.base import BaseCallbackManager  
from langchain.callbacks.streaming\_stdout import StreamingStdOutCallbackHandler

Next, provide the path to your PDF files and split them into smaller chunks. Save these chunks for further processing.

documents = PyPDFLoader('path to your pdf').load\_and\_split()  
text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1024,chunk\_overlap=64)  
texts = text\_splitter.split\_documents(documents)  
embeddings = HuggingFaceEmbeddings(model\_name='sentence-transformers/all-MiniLM-L6-v2')  
faiss\_index = FAISS.from\_documents(texts, embeddings)  
faiss\_index.save\_local("path to folder where you want to store index")

After saving it you can comment above lines except this otherwise it’ll create indexes each time you run the script.

embeddings = HuggingFaceEmbeddings(model\_name='sentence-transformers/all-MiniLM-L6-v2')

define path for gpt4all model, and load indexes.

# load vector store  
print("loading indexes")  
faiss\_index = FAISS.load\_local("path to your index folder", embeddings)  
print("index loaded")  
gpt4all\_path = 'path to your llm bin file'

perform a similarity search for question in the indexes to get the similar contents. You can update the second parameter here in the similarity\_search method based on the pages from the index you want to do similarity search.

# # Set your query here manually  
question = "your query"  
matched\_docs = faiss\_index.similarity\_search(question, 4)  
context = ""  
for doc in matched\_docs:  
context = context + doc.page\_content + " \n\n "

After this create template and add the above context into that prompt.

template = """  
Please use the following context to answer questions.  
Context: {context}  
 - -  
Question: {question}  
Answer: Let's think step by step."""

define LLM, prompt and create a LLMChain

callback\_manager = BaseCallbackManager([StreamingStdOutCallbackHandler()])  
llm = GPT4All(model=gpt4all\_path,n\_ctx=1000, callback\_manager=callback\_manager, verbose=True,repeat\_last\_n=0)  
prompt = PromptTemplate(template=template, input\_variables=["context", "question"]).partial(context=context)  
llm\_chain = LLMChain(prompt=prompt, llm=llm)

The callback\_manager parameter is optional. If you wish to monitor and track the different stages of your LLM (Language Model) execution, you can provide an appropriate callback handler or a list of callback handlers to this parameter. In this article, a specific callback called StreamingStdOutCallbackHandler is used to stream the response. However, you can also utilize other callback handlers such as logging or monitoring handlers to log or monitor the LLM's processes. Think of callbacks in a similar way to hooks used in frameworks like Angular or React, where different hooks serve different purposes, such as OnInit or OnChanges. They allow you to perform specific actions at various stages of the LLM's execution. If you're interested in learning more about callbacks, you can find additional information in the provided resource [here](https://python.langchain.com/docs/modules/callbacks/).

To prevent multiple responses from being printed, you can utilize the repeat\_last\_n parameter. By setting it to 0, only a single response will be displayed in your console. This ensures a cleaner and more concise output.

The n\_ctx (Token context window) in GPT4All refers to the maximum number of tokens that the model considers as context when generating text. It determines the size of the context window that the model uses to understand and generate coherent responses.

Now run the chain using .run() method

print(llm\_chain.run(question))