Project Title: Chat with Multiple PDFs

Team Members

Abhishek Basnet

Sushmitha Ugle Ashok

Yuvarekha Mahendran

Required libraries to be installed

```
pip install -q streamlit dotenv PyPDF2 langchain FAISS openai tiktoken
# streamlit: for building the web interface
# dotenv: for managing environment variables (e.g., API keys)
# PyPDF2: for extracting text from PDF files
# langchain: for text processing, embeddings, and conversational AI
# FAISS: for efficient similarity search in vector databases
# openai: for integrating OpenAI language models
# tiktoken: for tokenizing text for OpenAI models

# Importing necessary libraries
import streamlit as st  # For creating the web application
from dotenv import load_dotenv  # To load environment variables from a
.env file
from PyPDF2 import PdfReader  # For reading PDF files
from langchain.text_splitter import CharacterTextSplitter  # For
splitting text into manageable chunks
from langchain.embeddings import OpenAIEmbeddings  # To generate text
embeddings
from langchain.vectorstores import FAISS  # For storing and querying
vectors efficiently
from langchain.chat_models import ChatOpenAI  # Chat model interface
for OpenAI GPT
from langchain.memory import ConversationBufferMemory  # To store
conversation history in memory
from langchain.chains import ConversationalRetrievalChain  #
Conversational chain that retrieves relevant data
from htmlTemplates import css, bot_template, user_template  # Custom
HTML templates for styling
```

Function to extract text from PDF documents

```
def get_pdf_text(pdf_docs):
    text = ""  # Initialize an empty string to hold the extracted text
    for pdf in pdf_docs:  # Iterate through each uploaded PDF
        pdf_reader = PdfReader(pdf)  # Create a PDF reader object
        for page in pdf_reader.pages:  # Iterate through all pages in
the PDF
            text += page.extract_text()  # Extract and append text
```

```
from each page
    return text  # Return the combined text
  # Extracts text from a list of uploaded PDF files and Combined text
extracted from all the pages of all PDFs
```

Function to split text into smaller chunks

```
def get_text_chunks(text):
    text_splitter = CharacterTextSplitter(
        separator="\n",  # Split text by new lines
        chunk_size=1000,  # Maximum size of each text chunk
        chunk_overlap=200,  # Overlap size between consecutive chunks
for context
        length_function=len  # Function to determine the length of the
text
    )
    chunks = text_splitter.split_text(text)  # Perform the text
splitting
    return chunks  # Return the list of text chunks
 # Splits a large text into smaller chunks for easier processing
 # Looks for the newline separator (\n) to split the text. If no
newlines are found, it will fall back to splitting based
on the chunk_size.
```

Function to create a vectorstore using FAISS for storing embeddings of text chunks

```
def get_vectorstore(text_chunks):
    embeddings = OpenAIEmbeddings() # Initializing the embedding model
from OpenAI to generate text embeddings to transform chunks to
embeddings
    #numeric representations of words, phrases, or entire texts that
capture meaning(multi-dimension)

    vectorstore = FAISS.from_texts(texts=text_chunks,
embedding=embeddings)
    #vectors-represents data in numeric form(specific dimension)
    #FAISS (Facebook AI Similarity Search) is a library that allows
developers to quickly search for embeddings of documents that are
similar to each other.

    return vectorstore # Returns the created FAISS vectorstore
containing the text embeddings(storage mechanism for embeddings)
```

Function to create a conversational chain for handling chat interactions using LangChain

```
 def get_conversation_chain(vectorstore): #To set up a conversational
interface that uses the data.
    llm = ChatOpenAI(openai_api_key="sk-proj-
WfzHvAIUaDr155ETmMTbg1_xIq2w3KQbWnyE-
```

```
5iiPBS6JXAPD8of_Lnk26RRI9uEgyrm0RJ2gKT3BlbkFJAELqJe7DezOx907TS73hNI6UX
oBjw4Q2xOYVPpJI_-wI3hzKKaI-4TW6kWi6MCyt_-fvFjIJwA")
    #An LLM is a predictive engine that generates text with past data.
    #OpenAI API key is a unique identifier provided by OpenAI to
authenticate users to access their API.


    memory = ConversationBufferMemory(
        memory_key='chat_history', return_messages=True)
    # Setting up memory to track the chat history during the
conversation

    # 'return_messages=True' ensures that previous messages are
accessible for context
    conversation_chain = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=vectorstore.as_retriever(),
        memory=memory
    )
    #integrates natural language with external knowledge base like
vectorstore.
    #`retriever=vectorstore.as_retriever()`: Configures the
vectorstore to retrieve relevant data during the conversation

    return conversation_chain #Returns the configured conversation
chain for use in handling conversational queries
```

Function to process and handle the user's input question or query

```
def handle_userinput(user_question):
    # Using the user question to get a response from the conversation
chain.
    # 'conversation' is a session variable that stores the AI chat
model.
    # The function call retrieves a response based on the current user
question.
    response = st.session_state.conversation({'question':
user_question})

    # Store the chat history returned from the response in the session
state(feature to store variables across different runtimes).
    # 'chat_history' is updated with the complete conversation
history, including both user and AI messages.
    st.session_state.chat_history = response['chat_history']

    # Iterating through each message in the chat history. Each message
has two parts: user input and bot response.
    # The loop uses 'enumerate' to keep track of the index (i) of each
message in the chat history.
    for i, message in enumerate(st.session_state.chat_history):
```

```
        # Check if the index (i) is even. If it's even, it's a user
message.The chat history is assumed to store messages in an
alternating pattern: user message first, then bot response
        if i % 2 == 0:
            # Replacing the {{MSG}} placeholder in the user_template
with the user's message content.
            # Then display the formatted HTML using Streamlit's
st.write method.
            st.write(user_template.replace(
                "{{MSG}}", message.content), unsafe_allow_html=True)
        else:
            # If the index (i) is odd, it's a bot (AI) response.
            # Replacing the {{MSG}} placeholder in the bot_template
with the bot's message content.
            # Then displaying the formatted HTML using Streamlit's
st.write method.
            st.write(bot_template.replace(
                "{{MSG}}", message.content), unsafe_allow_html=True)

def main():
    load_dotenv()
    st.set_page_config(page_title="Chat with multiple PDFs",
                       page_icon="")
    st.write(css, unsafe_allow_html=True) #using css template to write

    if "conversation" not in st.session_state:
        st.session_state.conversation = None
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = None

    st.header("Chat with multiple PDFs")
    user_question = st.text_input("Ask a question about your
documents:")
    if user_question:
        handle_userinput(user_question)

    with st.sidebar:
        st.subheader("Your documents")
        pdf_docs = st.file_uploader(
            "Upload your PDFs here and click on 'Process'",
accept_multiple_files=True)
        if st.button("Process"):
            with st.spinner("Processing"):
                # get pdf text
                raw_text = get_pdf_text(pdf_docs)

                # get the text chunks
                text_chunks = get_text_chunks(raw_text)

                # create vector store
```

```
                vectorstore = get_vectorstore(text_chunks)

                # create conversation chain
                st.session_state.conversation =
get_conversation_chain(
                    vectorstore)

if __name__ == '__main__':
    main()
```

Chatbot With One PDFChat%20with%20multiple%20PDFs.png

Chatbot With Mulitple PDF's Chat%20with%20multiple%20PDFs.png

image.png

image.png

requirements.txt File

langchain==0.0.184

PyPDF2==3.0.1

python-dotenv==1.0.0

streamlit==1.18.1

openai==0.27.6

faiss-cpu==1.7.4

altair==4

tiktoken==0.4.0