Lab-6 Report

Team Details

Team Name: DPS

Name	USC ID
Dolly Rijwani	9040345530
Prathmesh Lonkar	3661203370
Shreyan Yoge	9655139013

Table of Contents

Implementation

This project is designed to provide a conversational interface where users can interact with PDF documents using natural language queries. The web-based interface allows users to upload PDFs, process them, and then engage in a Q&A session with the content of the documents. The implementation uses a combination of modern AI and machine learning techniques, leveraging OpenAI's language models and LangChain's retrieval-augmented generation (RAG) framework.

The key objectives of the project include:

- Extracting textual content from multiple PDF files.
- Splitting this content into manageable chunks for efficient processing.
- Embedding these chunks into vector representations for semantic search.
- Allowing users to query the documents, where the system retrieves the most relevant sections based on the query.
- Creating a conversational experience that maintains context between queries and responses.

This conversational experience is built using Streamlit, which provides a simple and interactive web interface. The system retrieves relevant information from the uploaded PDFs using embeddings and vector search to find the most relevant parts of the documents. The results are then passed to a language model to generate natural responses.

Source Code

Libraries

- **`os`**: Used for accessing environment variables, specifically to retrieve the OpenAI API key stored in a `.env` file.
- **`streamlit`**: The framework used to build the web-based user interface. It handles input/output functions like uploading files, displaying text, and managing interaction between the user and the backend logic.
- **`PyPDF2`**: A library used to parse PDF files and extract the text from their pages. It enables reading and processing multiple pages in a PDF document.
- **`langchain.text_splitter.CharacterTextSplitter`**: A utility from LangChain that splits text into smaller chunks for efficient processing. This prevents overwhelming the model with large inputs and helps maintain context by adding overlap between chunks.
- **`langchain_community.embeddings.OpenAIEmbeddings`**: This component generates vector embeddings (numerical representations) of text using OpenAI’s

embedding models. These embeddings are used in vector search for similarity-based retrieval.

- **`langchain_community.vectorstores.FAISS`**: FAISS (Facebook AI Similarity Search) is a library that efficiently stores and searches through large vectors. This is used to store the text embeddings and retrieve the most relevant text chunks for a query.
- **`langchain_community.chat_models.ChatOpenAI`**: The conversational AI model, typically powered by GPT-3 or GPT-4, which generates human-like responses to the user's questions based on the retrieved document sections.
- **`langchain.chains.ConversationalRetrievalChain`**: This chain combines both retrieval and conversation, enabling the system to retrieve relevant sections of the PDF content based on the user query and use the language model to generate responses. It also integrates memory to store the conversation history, maintaining context between interactions.
- **`dotenv`**: Used to load environment variables (such as the OpenAI API key) from a `.env` file.
- **`htmlTemplates`**: A local module containing HTML templates for rendering the conversation between the user and the AI. It formats the conversation in a chatbot style within the web interface.

Functions

1. **`get_pdf_text(pdf_docs)`**:
   - **Purpose**: This function takes the uploaded PDF files and extracts their text content.
   - **Explanation**: It reads each PDF file using `PdfReader` from PyPDF2, then loops through all the pages to extract the text. The extracted text from all pages is concatenated into a single string and returned. This function serves as the first step in processing the PDF content.
2. **`get_text_chunks(text):`**
   - **Purpose**: To split the extracted text into smaller, manageable chunks.
   - **Explanation**: The LangChain `CharacterTextSplitter` is used to break the extracted text into chunks of 500 characters with a 100-character overlap. The overlap ensures that there is some continuity between the chunks, making it easier to retrieve meaningful answers when the text spans across chunks. This function prepares the text for embedding and vector storage.
3. **`get_vectorstore(text_chunks)`**:
   - **Purpose**: To create a vector store for the text chunks, enabling fast and efficient similarity search.
   - **Explanation**: This function uses the OpenAI embedding model to convert each text chunk into vector embeddings. These vectors are stored in FAISS, which allows for efficient retrieval of the most relevant chunks when a user query is

processed. The output of this function is a FAISS vector store that holds all the embeddings for the PDF content.

4. **`get_conversation_chain(vectorstore)`**:
   - **Purpose**: To set up the retrieval-based conversational chain.
   - **Explanation**: This function creates the core of the AI conversation. It initializes the OpenAI chat model (`ChatOpenAI`) and links it with the FAISS vector store, using the `ConversationalRetrievalChain` component. It also integrates a memory buffer (`ConversationBufferMemory`), which allows the conversation to retain context and maintain a natural flow over multiple interactions. This conversational chain is what processes the user queries and provides the relevant responses.
5. **`display_chat()`**:
   - **Purpose**: To display the conversation history between the user and the AI.
   - **Explanation**: This function is responsible for rendering the chat history using the HTML templates defined in `htmlTemplates`. It reads the conversation history stored in `st.session_state` and displays the alternating user and AI messages in a visually appealing format within the Streamlit app. The chat history is reversed to show the latest messages at the top.
6. **`handle_userinput(user_question)`**:
   - **Purpose**: To process user questions and update the conversation.
   - **Explanation**: When the user inputs a question, this function sends the question to the conversational chain created earlier. It receives a response, which contains both the answer and the updated conversation history. The history is then stored in `st.session_state` for future reference and display.
7. **`build_interface()`**:
   - **Purpose**: To create the entire user interface for the web app.
   - **Explanation**: This function sets up the web interface using Streamlit. It defines the page layout, the chat interface, and the sidebar for PDF uploads. Users can type questions, upload PDF files, and process the PDFs for interaction. If PDFs are uploaded and the 'Process' button is clicked, it extracts the text, splits it, generates vector embeddings, and creates the conversation chain. If the user asks a question, it checks if the PDF has been processed and interacts with the conversation chain accordingly.
8. **`__main__` section**:
   - **Purpose**: To launch the Streamlit app when the script is run.
   - **Explanation**: The `build_interface()` function is called here to start the application, which will then run in the Streamlit environment.

Conclusion

This project successfully combines several cutting-edge technologies to create an intuitive system for interacting with PDF documents using conversational AI. The use of LangChain's retrieval-augmented generation, combined with OpenAI embeddings and FAISS, ensures that the system can provide relevant and context-aware responses based on the content of the uploaded PDFs. The web-based interface built with Streamlit makes this system easily accessible to users, providing a seamless experience for document-based question answering.

Individual Contribution

Dolly Rijwani

- Worked on development of PDF extraction and embedding part.
- Assisted in creating the Readme document to list all the steps required to reproduce the working of this chatbot.

Shreyan Yoge & Prathmesh Lonkar

- Worked on the development of User parsing, handling, Implementation of OpenAI GPT model, and development of Streamlit GUI.
- Documented the team discussions, minutes of meeting and produced the main submission report by documenting all the chatbot features and source code details.

***** End of Document *****