REPORT

The following steps were taken to accomplish the task at hand:

1. Install Required Libraries and import necessary modules

The first part of the code installs several libraries required for the chatbot and PDF processing. This includes LangChain, ChromaDB, **Google Generative AI, PyPDF2, and Unstructured**. The necessary modules from the installed packages are imported.

2. Load Documents from ZIP File

The function load_docs_from_zip extracts documents from a ZIP file and loads them using LangChain's **DirectoryLoader**.

First, the function defines a temporary directory where the ZIP file contents will be extracted. It then opens the ZIP file using Python's **zipfile library** and extracts all the contents into the temporary directory. Next, the function utilizes LangChain's **DirectoryLoader** to load the documents from the extracted directory. Finally, the loaded documents are returned as a list.

3. Split Documents

The documents are split into smaller chunks based on specified chunk size and overlap parameters using **RecursiveCharacterTextSplitter** for better processing.

4. Create Embeddings and Vector Store

I have used the **SentenceTransformer** library to initialize a pre-trained sentence embeddings model called "all-MiniLM-L6-v2." Additionally, I imported the Chroma class from **langchain.vectorstores** to handle vector storage efficiently. By creating a **Chroma database (db)** from a list of documents (docs) using the specified sentence embeddings, I enabled seamless storage and retrieval of vectorized representations of the textual data, empowering tasks like similarity search or semantic clustering.

5. Perform Similarity Search

Performed a similarity search using this query against the **db** object created earlier, storing the results in the **matching_docs** variable. This process retrieves documents from the database that are most similar in content or context to the provided query.

6. Markdown Display Helper Function

A helper function **to_markdown** to format text as Markdown for better readability in **Jupyter**.

7. Use LangChain for QA using LLMChains and load_qa_chain

A QA chain is created using **LangChain**, which takes the context as related date from document generated in step 5 and question, and generates an answer based on context.

8. Use LangChain for QA using LLMChains that keeps Conversation history

A QA chain is created using **LangChain prompt_template**, which takes the context as related date from document generated in step 5 , question and chat history as previous responses, and generates an answer based on context and chat history.

9. Generated Query_Refiner function

A refined query is generated using **Google Generative AI**. The refined query is based on the user's query and the previous context. It is based on semantics of the query with the previous conversation.

The function **query_refiner** uses a generative model from the **GenAI library (gemini-1.5-flash)** to refine a user **query based on a conversation log and the initial query**. It starts a chat session, sends the conversation log and query as input, and returns the model's generated refined query as the output.

10. QA chain using Query Refiner

Loads an **LLMChain** model and defines a function **fnc** that performs the following steps:

1. Uses **new_db.similarity_search(query)** to find matching documents based on the input query.
2. Combines the content of these matching documents with previous context (prev) to form a full context.
3. Refines the query using the **query_refiner** function.
4. Runs the refined query through the QA chain to generate an answer.

Overall, the code integrates language models for contextual understanding and question answering, with a focus on refining queries and generating accurate answers based on given contexts and questions.

Summary

- Installed required libraries and imported necessary modules.
- Loaded documents from a ZIP file and split them into smaller chunks.
- Created embeddings and a vector store using SentenceTransformer and Chroma.
- Performed a similarity search to retrieve relevant documents.
- Developed a markdown display helper function for better readability.
- Utilized LangChain for question-answering (QA) tasks using LLMChains and load_qa_chain.

- Implemented a query_refiner function using Google Generative AI for refining user queries.
- Integrated QA chain using query refiner for contextual understanding and accurate question answering.