Overall Approach

1. I used Streamlit to create the chatbot interface and manage user interactions.
2. I implemented the open-source LLM model "Llama3-8b-8192," available on the Groq platform via API.
3. I utilized Groq's LPU inference engine to run the model for faster response generation.
4. I structured the chatbot's responses using a prompt template.
5. I loaded my PDF document using PDF loaders and split it into smaller chunks.
6. I generated embeddings for the document using GoogleGenerativeAIEmbeddings.
7. I employed FAISS to create a vector store from the document embeddings.
8. Upon user input, the system performs document retrieval and generates a response.

Frameworks/Libraries/Tools Used in the Chatbot

1. **Streamlit**

   Streamlit is an open-source app framework used for creating and sharing web apps for data science and machine learning.
   - Creating the user interface (st.title, st.write, st.chat_input).
   - Managing session states to store embeddings and vectors.

2. **LangChain**

   LangChain is a framework for developing applications powered by language models. It provides utilities for prompt engineering, chaining together components, and working with embeddings. .
   - Creating prompt templates (ChatPromptTemplate).
   - Combining document chains (create_stuff_documents_chain).
   - Text splitting (RecursiveCharacterTextSplitter).
   - Creating retrieval chains (create_retrieval_chain).

3. **LangChain Groq**

   Provides integration with Groq's language models, allowing the use of Groq-powered large language models in LangChain applications.
   - Configuring the language model (ChatGroq).

4. **LangChain Community Vector Stores (FAISS) and Document Loaders**

   FAISS (Facebook AI Similarity Search) is a library for efficient similarity search and clustering of dense vectors. The LangChain Community Vector Stores provide integration with FAISS. Document Loaders provides utilities for loading documents into the LangChain framework.
   - 'PyPDFLoader' is used to load PDF documents.
   - Creating vector embeddings and storing them (FAISS).

5. **Google Generative AI Embeddings**

   Google Generative AI provides tools for creating embeddings, which are numerical representations of text that can be used for various natural language processing task.
   - Generating embeddings for the documents (GoogleGenerativeAIEmbeddings).

6. **Dotenv**

   Python library used to read key-value pairs from a .env file and set them as environment variables.
   - Loading API keys from a .env file (load_dotenv).

7. **Python Standard Library**
   - Setting environment variables (os.environ).
   - Measuring response time (time.process_time).

Problem Faced during development

- One of the major problem I faced was the time taken by LLM model to generate the response.

To solve this problem I used Groq's powered large language models which generate faster response.

- Next Challenge for me was to built an interface where user can chat with the chatbot.

To overcome this challenge I used Streamlit which provides me a proper interface for my chatbot.

Future Scope

- I**ntegration with External Services**: Connect the chatbot to various external APIs and services, such as calendars, weather forecasts, and task management tools, to provide more utility.
- **Multilingual Support**: Implement language translation capabilities to support users from different linguistic backgrounds.
- **Personalization**: Develop user profiles to provide personalized responses and recommendations based on previous interactions.
- **Improved Document Handling**: Expand the types of documents the chatbot can handle (e.g., images, spreadsheets) and improve the efficiency of document processing and retrieval.
- **Feedback Mechanism**: Incorporate a feedback mechanism where users can rate responses and provide suggestions for improvement.