Chatbot for PDF: Project Report

1. Overall Approach

The goal of this project was to create a chatbot capable of answering questions about a wine business based on a given corpus. The chatbot should be able to handle questions from the corpus and redirect users to contact the business for out-of-corpus questions. Additionally, the chatbot should maintain context across the conversation for more coherent interactions.

Steps:

1. **Text Extraction**: Extract text data from the provided PDF documents.
2. **Text Chunking**: Split the extracted text into smaller chunks to facilitate efficient processing and searching.
3. **Vector Store Creation**: Generate vector embeddings for the text chunks and store them using FAISS for efficient similarity search.
4. **Conversational Chain Setup**: Create a conversational chain using LangChain and Google Generative AI, ensuring the chatbot can handle context-aware responses.
5. **User Interaction**: Implement a Streamlit-based UI for users to interact with the chatbot, upload PDFs, and ask questions.

2. Frameworks/Libraries/Tools Used

2.1 Streamlit

- **Usage**: Created the web interface for user interaction.
- **Reason**: Provides an easy-to-use framework for building interactive web applications.

2.2 PyPDF2

- **Usage**: Extracted text from PDF documents.
- **Reason**: Efficiently handles PDF text extraction, supporting various PDF formats.

2.3 LangChain

- **Usage**: Managed the creation of conversational chains and maintained context throughout the conversation.
- **Reason**: Provides robust tools for building conversational AI applications.

2.4 FAISS

- **Usage**: Stored and performed similarity searches on text embeddings.
- **Reason**: Optimized for fast and efficient similarity search operations.

2.5 Google Generative AI

- **Usage**: Generated text embeddings and handled the QA model for the chatbot.
- **Reason**: Provides state-of-the-art generative models with robust performance.

2.6 Python-dotenv

- **Usage**: Managed environment variables for API keys.
- **Reason**: Simplifies the management of environment variables in Python projects.

3. Problems Faced and Solutions

3.1 Ensuring Accurate Context Handling

- **Problem**: Maintaining context across multiple user queries was challenging, especially for follow-up questions.
- **Solution**: Utilized LangChain's `ConversationBufferMemory` to keep track of the conversation history and provided relevant context to the model.

3.2 Reducing Latency and API Costs

- **Problem**: High latency and API costs due to the use of generative models.
- **Solution**: Optimized text chunking and embedding generation to reduce unnecessary API calls. Used FAISS for efficient similarity search to minimize the number of tokens processed by the model.

3.3 Handling Out-of-Corpus Questions

- **Problem**: Ensuring the chatbot correctly identifies and handles questions not covered by the corpus.
- **Solution**: Implemented a fallback mechanism where the chatbot directs users to contact the business if the similarity score of the retrieved documents is below a certain threshold.

4. Future Scope

4.1 Advanced Conversational Features

- **Description**: Implement more sophisticated conversational abilities, such as handling ambiguous queries and providing more nuanced responses.

4.2 Multi-Language Support

- **Description**: Extend the chatbot to support multiple languages to cater to a wider audience.

4.3 Improved UI/UX

- **Description**: Enhance the user interface for better interaction, including features like voice input and more interactive elements.

4.4 Error Handling and Fallback Mechanisms

- **Description**: Develop more robust error handling to gracefully manage API failures or unexpected user inputs.

4.5 Integration with CRM Systems

- **Description**: Connect the chatbot with Customer Relationship Management (CRM) systems to provide personalized responses based on user profiles and past interactions.