Project Report: Conversational Q&A Chatbot

1. Overall Approach

This project involves creating a conversational Q&A chatbot that utilizes document-based knowledge to answer user queries. The core approach integrates document processing with natural language understanding to provide accurate and contextually relevant answers. The chatbot is designed to interact with users in a conversational manner, maintaining a history of interactions to provide context-aware responses.

2. Frameworks/Libraries/Tools Used

- **LangChain**: Used for document processing, creating prompt templates, and chaining different components to handle the question-answering task.
    - `PromptTemplate`: For creating prompts that guide the language model in generating responses.
    - `Document`: Represents the text content extracted from the PDF.
    - `StrOutputParser`: For parsing the output of the language model into a structured format.
    - `Chroma`: A vector store for managing and retrieving document embeddings.
- **Google Generative AI**: Provides the language model and embeddings used for understanding and generating responses.
    - `ChatGoogleGenerativeAI`: The model used for generating responses based on user queries and context.
    - `GoogleGenerativeAIEmbeddings`: Used to create embeddings for document vectors.
- **Streamlit**: For creating the web interface where users interact with the chatbot.
    - Used to display chat history and handle user input in a web-based interface.
- **PyPDF2**: For reading and extracting text from PDF documents.
    - `PdfReader`: Reads the content of the PDF and extracts text for processing.
- **dotenv**: Used for loading environment variables, such as the API key for Google Generative AI.

3. Problems Faced and Solutions

- **API Key Management**: Initial issues with API key management and authentication were resolved by using the `dotenv` library to securely manage environment variables.
- **Handling Text Extraction from PDF**: The text extraction from PDF documents sometimes resulted in formatting issues. This was addressed by carefully processing the extracted text and splitting it into manageable segments.
- **Streamlit Integration**: There were challenges in integrating Streamlit with LangChain components. The issue was resolved by ensuring that the input and output formats match the expectations of both Streamlit and LangChain components.

- **Error Handling**: Encountered errors related to context management and incorrect handling of Streamlit components. Solutions involved revising code to ensure proper context handling and use of Streamlit functions.

4. Future Scope

- **Enhanced Contextual Understanding**: Integrate more advanced context management to handle complex conversations and multi-turn interactions.
- **Integration with Other Data Sources**: Extend the chatbot's capabilities by integrating it with other data sources like databases, APIs, or real-time data feeds.
- **User Personalization**: Implement user-specific profiles and preferences to provide personalized responses based on user history and preferences.
- **Multi-Language Support**: Add support for multiple languages to make the chatbot accessible to a broader audience.
- **Improved User Interface**: Enhance the Streamlit interface to include features such as multimedia support, better formatting, and interactive elements.
- **Continuous Learning**: Implement mechanisms for continuous learning and updating the knowledge base from user interactions and feedback.

Arnav Lahane