Chatbot for PDF Interaction using Google Generative AI

Overall Approach

The project involves creating a Streamlit app that allows users to upload PDF files, process the text content, and ask questions based on the extracted text. The approach can be summarized in the following steps:

1. **File Upload and Text Extraction**: Users can upload multiple PDF and PPTX files, and the app extracts text from these files.
2. **Text Chunking**: The extracted text is split into manageable chunks to facilitate efficient processing.
3. **Vector Store Creation**: The text chunks are converted into embeddings using Google Generative AI and stored in a FAISS vector store.
4. **Question Answering**: Users can ask questions related to the uploaded files, and the app uses the vector store and Google Generative AI to find the most relevant chunks and generate answers.

Frameworks/Libraries/Tools Used

- **Streamlit**: Used to create the web application interface.
- **PyPDF2**: Used for extracting text from PDF files.
- **python-pptx**: Used for extracting text from PPTX files.
- **langchain**: Provides tools for text splitting and creating the question-answering chain.
- **langchain-google-genai**: Used for Google Generative AI embeddings.
- **google-generativeai**: Library to configure and use Google Generative AI.
- **FAISS**: Used for creating and managing the vector store for efficient similarity search.
- **python-dotenv**: Used to load environment variables from a `.env` file.

Problems Faced and Solutions

1. **Text Extraction Issues**:
    - **Problem**: Some PDF files had complex structures that made text extraction challenging.
    - **Solution**: Proper documentation of the libraries is followed to tackle this problem.
2. **Large File Handling**:
    - **Problem**: Processing large files could lead to memory and performance issues.
    - **Solution**: Implemented text chunking to split large texts into smaller, manageable chunks.
3. **Asynchronous Operations**:
    - **Problem**: Ensuring smooth asynchronous processing without blocking the Streamlit UI.

    - **Solution**: Used `asyncio.run` and considered `asyncio.create_task` for better performance.
4. **Integration of Multiple Libraries**:
    - **Problem**: Integrating various libraries and ensuring compatibility.
    - **Solution**: Carefully managed dependencies and used well-documented libraries.

Future Scope

1. **Enhanced File Support**:
    - Support for additional file formats like PPTX, DOCX, TXT, and HTML.
2. **Improved Text Processing**:
    - Advanced text processing techniques to handle more complex document structures.
3. **Scalability**:
    - Optimization for handling very large documents and multiple simultaneous users.
4. **Interactive Features**:
    - Adding more interactive elements like highlighting the relevant text in the document for the provided answer.
    - Visualizing the document structure and allowing users to navigate through it.
5. **Natural Language Understanding**:
    - Improving the chatbot's ability to understand and respond to more complex queries using advanced NLP techniques.
6. **Customization**:
    - Allowing users to customize the chatbot's behavior and appearance.
7. **Integration with Other Services**:
    - Integrating with cloud storage services like Google Drive or Dropbox for direct file uploads.
    - Connecting with other AI services for enhanced functionalities.

By incorporating these features, the chatbot can become a more powerful tool for interacting with and extracting information from various document types, providing a seamless and efficient user experience.