AI Chatbot

a. Overall Approach

The chatbot was developed with the goal of answering user questions based on the content of an uploaded PDF document. The chatbot maintains conversation history to provide context for follow-up questions, ensuring coherent and relevant responses even when queries are related to previous interactions.

The overall approach involved the following steps:

1. **PDF Content Extraction**: Extract text content from the uploaded PDF file.
2. **Text Chunking**: Split the extracted text into manageable chunks to facilitate efficient semantic search.
3. **Semantic Search**: Use a pre-trained model to perform semantic search and find the most relevant text chunks for a given query.
4. **Question-Answering**: Utilize a language model to generate answers based on the relevant text chunks and the query.
5. **Conversation History**: Maintain and display conversation history to provide context for follow-up questions and improve user experience.

b. Frameworks/Libraries/Tools Used

- **Streamlit**: Used for creating the web interface, handling file uploads, and maintaining session state for conversation history.
- **PyPDF2**: Utilized for extracting text from PDF files.
- **LangChain**: Used for text chunking with CharacterTextSplitter.
- **Sentence-Transformers**: Employed for semantic search to find relevant text chunks.
- **Requests**: Used to make API calls to Hugging Face for feature extraction and AI21 Studio for the question-answering model.
- **Torch**: Used to handle embeddings and perform semantic search operations.

c. Problems Faced and Solutions

1. **Latency in API Calls**:
    - **Problem**: Making API calls to Hugging Face and AI21 Studio introduced latency, affecting response times.
    - **Solution**: Optimized the number of API calls by performing batch processing for text chunks and embeddings. Also, used the wait_for_model option to ensure model readiness, reducing unnecessary delays.
2. **Maintaining Conversation Context**:
    - **Problem**: Ensuring the chatbot could refer back to previous questions and answers required effective management of conversation history.

- **Solution**: Utilized Streamlit's st.session_state to store and manage conversation history, concatenating previous queries and responses to provide context for new questions.

3. **Handling Large Text Data**:
    - **Problem**: Managing large text data from PDF files and ensuring efficient semantic search was challenging.
    - **Solution**: Implemented text chunking to split large documents into smaller, manageable chunks, making semantic search more efficient and accurate.

d. Future Scope of the Chatbot

The chatbot can be enhanced with several additional features and improvements to further improve its functionality and user experience:

1. **Use of OpenAI Models**:
    - Integrate OpenAI's GPT-4 or future models for more advanced and accurate question-answering capabilities.
    - Use OpenAI's fine-tuning capabilities to specialize the model based on specific document types or domains.
2. **Retrieval-Augmented Generation (RAG) Model**:
    - Implement a RAG model that combines retrieval-based and generation-based approaches for more accurate and contextually relevant answers.
3. **Improved Text Processing**:
    - Enhance text chunking algorithms to better handle complex document structures, such as tables, figures, and multi-column layouts.
4. **Multimodal Capabilities**:
    - Extend the chatbot to handle multimodal inputs, such as images and scanned documents, using OCR (Optical Character Recognition) and image captioning models.
5. **Real-Time Collaboration**:
    - Enable real-time collaboration features where multiple users can interact with the chatbot simultaneously, useful for team-based document analysis.
6. **Interactive Visualizations**:
    - Integrate interactive visualizations to display document content, search results, and conversation history in a more intuitive manner.