Chatbot Project Documentation

Overall Approach

The chatbot is designed to assist users by answering their questions based on a pre-defined corpus of questions and answers. The approach involves the following steps:

1. Loading the Corpus: The corpus is loaded from a JSON file containing a set of questions and answers. Additionally, text is extracted from a PDF file to expand the chatbot's knowledge base.
2. Encoding Sentences: The questions and sentences from the corpus are encoded into embeddings using a pre-trained SentenceTransformer model.
3. Chatbot Interaction: A Tkinter-based GUI is used to interact with the user. The user's input is processed to find the most similar question in the corpus, and the corresponding answer is provided.
4. Text-to-Speech: The responses are also read aloud using the pyttsx3 library.

Frameworks/Libraries/Tools Used

1. sentence-transformers: Used to encode sentences into embeddings. The SentenceTransformer class from this library helps in creating vector representations of text for similarity comparison.
    - Usage: Encoding corpus questions and user queries.
2. scikit-learn: Provides the cosine_similarity function to compute similarity between vectors.
    - Usage: Calculating similarity between user query embeddings and corpus embeddings.
3. numpy: Used for numerical operations and handling arrays.
    - Usage: Managing and processing embeddings and similarity scores.
4. PyMuPDF (fitz): Used to extract text from PDF files.
    - Usage: Reading and extracting text content from the corpus.pdf file.
5. tkinter: Provides the GUI for the chatbot.
    - Usage: Creating the chat window, input box, and send button for user interaction.
6. pyttsx3: A text-to-speech conversion library.
    - Usage: Reading the chatbot responses aloud.
    -

Problems Faced and Solutions

1. Extracting Text from PDF:
    - Problem: Ensuring accurate text extraction from the PDF file.

- Solution: Used PyMuPDF (fitz) for reliable and efficient text extraction.

2. Sentence Segmentation:
    - Problem: Splitting text from the PDF into meaningful sentences.
    - Solution: Implemented a simple split by period (.). This can be improved with more advanced segmentation techniques.
3. Embedding Large Corpus:
    - Problem: Encoding a large number of sentences could be time-consuming and memory-intensive.
    - Solution: Used the all-MiniLM-L6-v2 model from sentence-transformers for efficient encoding with a balance between speed and accuracy.
4. GUI Responsiveness:
    - Problem: Ensuring the GUI remains responsive during text-to-speech processing.
    - Solution: Used tkinter to handle user interactions and updated the GUI before performing text-to-speech operations.

Future Scope

1. Enhanced NLP Capabilities:
    - Integrate more advanced NLP models such as GPT-3 or GPT-4 for generating more accurate and context-aware responses.
2. Improved Text Extraction:
    - Use advanced sentence segmentation techniques to better handle text extraction from PDFs.
3. Knowledge Base Expansion:
    - Allow dynamic updates to the knowledge base with new questions and answers without requiring a restart.
4. Multilingual Support:
    - Add support for multiple languages in both text processing and text-to-speech.
5. Contextual Understanding:
    - Implement context tracking to provide more coherent responses over multiple interactions.
6. Voice Recognition:
    - Integrate speech-to-text functionality to allow voice input from users.
7. User Authentication and Customization:

- Implement user authentication to provide personalized responses based on user history and preferences.

8. Deployment:
    - Deploy the chatbot as a web application or integrate it with messaging platforms like Slack or Microsoft Teams.