Chatbot Project Documentation

1. Overall Approach

Loading Q&A Pairs

I start by loading a set of pre-defined question and answer pairs from a JSON file. This helps in quickly setting up a basic knowledge base for the chatbot.

Extracting Text from PDF

Next, I extract text from a provided PDF file using the `PyPDF2` library. This text is added to the knowledge base to improve the chatbot's ability to answer questions related to the content of the PDF.

Combining Data

The Q&A pairs and the extracted text from the PDF are combined into a single corpus. This corpus forms the foundation of the chatbot's knowledge.

Encoding Data

I use the `SentenceTransformer` library to encode the questions from the Q&A pairs into embeddings. These embeddings are used to find the best match for user queries based on cosine similarity.

Flask API

I created a Flask API endpoint to handle user queries. The API takes a user message, encodes it, and then finds the most similar question in the corpus. If the similarity score is above a certain threshold, the corresponding answer is returned; otherwise, a default message is provided.

Frontend Interface

I design a simple chat interface using HTML, CSS, and JavaScript. The interface displays a greeting message from the chatbot upon loading and allows users to type their queries.

2. Frameworks/Libraries/Tools Used

Flask

- **Usage**: To create the web API.

- **Purpose**: Handles HTTP requests and serves the chatbot responses.

SentenceTransformer

- **Usage**: For encoding and finding the best match for user queries.
- **Purpose**: Provides pre-trained models that generate embeddings for sentences.

PyPDF2

- **Usage**: For extracting text from PDF files.
- **Purpose**: Enables text extraction from various PDF formats to enrich the chatbot's knowledge base.

Flask-CORS

- **Usage**: For handling Cross-Origin Resource Sharing (CORS).
- **Purpose**: Allows the frontend and backend to communicate even when they are hosted on different servers.

HTML/CSS/JavaScript

- **Usage**: For creating the frontend interface.
- **Purpose**: Provides a user-friendly interface for interacting with the chatbot.

3. Problems Faced and Solutions

PDF Text Extraction

- **Problem**: Ensuring that text is correctly extracted from various PDF formats can be challenging.
- **Solution**: I used the `PyPDF2` library, which reliably extracts text from most PDF files.

Model Performance

- **Problem**: Ensuring the chatbot provides accurate and relevant answers.
- **Solution**: I tuned the cosine similarity threshold to improve matching accuracy. Further improvements can be achieved by experimenting with different pre-trained models or fine-tuning a model specifically for our domain.

CORS Issues

- **Problem**: Cross-origin requests between the frontend and backend.
- **Solution**: Implemented `Flask-CORS` to enable proper communication between the frontend and backend.

4. Future Scope

Improving Model Accuracy

- **Description**: Experiment with different pre-trained models or fine-tune a model for specific domains to improve the chatbot's accuracy.
- **Benefit**: More accurate and relevant responses to user queries.

Adding More Data Sources

- **Description**: Incorporate additional data sources like databases or APIs to enhance the chatbot's knowledge base.
- **Benefit**: Provides more comprehensive answers to a wider range of queries.

Enhancing Frontend Interface

- **Description**: Create a more interactive and user-friendly frontend interface.
- **Benefit**: Improves user experience and engagement with the chatbot.

Voice Interaction

- **Description**: Implement voice input and output capabilities using tools like Web Speech API.
- **Benefit**: Makes the chatbot more accessible and engaging for users who prefer voice interaction.