Flask Chatbot with LangChain & OpenAI

Ranishree Anegundi

Overview

- Objective:
- - Create a chatbot that answers questions based on PDF documents using OpenAI's GPT-3.5 and LangChain framework.

- Key Features:
- - Load PDFs for document processing.
- - Text splitting for large document handling.
- - Conversational AI with contextual memory.

Architecture Overview

- - Flask Web Framework: Handles frontend and backend interactions.
- - LangChain: Manages document embedding, text splitting, and conversational retrieval.
- - OpenAI API: Powers the chatbot with GPT-3.5 for natural language processing.
- - File Handling: Upload PDFs for chatbot reference.

Technology Stack

- - Flask: Python-based web framework for building the app.
- - LangChain: For document loading, splitting, and retrieval-based question answering.
- - OpenAI GPT-3.5: Language model to generate responses based on document content.
- - PyPDFLoader: To extract text from PDF files.
- - DocArrayInMemorySearch: To store and retrieve document embeddings.

Flowchart: Process Overview

- 1. File Upload: User uploads a PDF document.
- 2. Document Processing: The document is split into chunks for efficient retrieval.
- 3. Embedding and Indexing: The text is embedded using OpenAI embeddings, and stored for search.
- 4. Chat: User asks questions, and the chatbot provides responses based on document content.
- 5. Response Generation: GPT-3.5 model

Key Routes in Flask

- - `/`: Main page to upload PDF and interact with the chatbot.
- - `/load_db`: Uploads a PDF file for processing.
- - `/chat`: Sends user query to the chatbot and gets a response.
- - `/clear_history`: Clears the chat history for a fresh start.

How It Works (Step-by-Step)

- 1. Upload Document: User uploads a PDF file, which is saved in the backend.
- 2. Document Splitting & Embedding: Text is split into manageable chunks for embedding.
- 3. Conversational Querying: The chatbot uses the `qa` object to retrieve context from the documents and generate answers.
- 4. Chat History Management: A list tracks the user’s chat history for conversational context.

Advantages of This Approach

- - Scalable: Can handle large documents by splitting them into smaller chunks.
- - Interactive: Chatbot gives contextual responses based on user queries.
- - Extensible: New document types or language models can be integrated easily.

Challenges and Future Improvements

- - Performance: Optimizing for very large documents.
- - Error Handling: Improve resilience against invalid file formats.
- - Advanced Features: Adding support for other document formats like DOCX or TXT.

Conclusion

- - This Flask-based application demonstrates a seamless integration of LangChain and OpenAI's GPT-3.5 to create a powerful, document-based conversational AI.
- - Future enhancements will focus on improving scalability and feature expansion.