ChatterMate: Chat with Multiple PDFs

ChatterMate is a Streamlit application that allows users to upload multiple PDF documents and then interact with a conversational AI assistant to ask questions about the content of the uploaded PDFs.

Introduction

ChatterMate is a powerful tool that enables users to quickly and efficiently retrieve information from a collection of PDF documents. By leveraging the capabilities of large language models and vector search, the application provides a conversational interface that allows users to ask questions and receive relevant responses based on the content of the uploaded PDFs.

Features

- PDF Upload: Users can upload multiple PDF documents to the application.

- Text Extraction: The application extracts the text content from the uploaded PDF documents.

- Text Chunking: The extracted text is split into smaller chunks to enable efficient vector search.

- Vector Store Creation: The text chunks are embedded and stored in a vector store using the OpenAI Embeddings and FAISS library.

- Conversational Interface: Users can ask questions through a text input field, and the application will provide relevant responses based on the content of the uploaded PDFs.

- Conversation History: The application maintains a conversation history that is displayed to the user.

Installation

1. Clone the repository or download the source code.

2. Navigate to the project directory in your terminal or command prompt.

3. Create a new virtual environment and activate it.

4. Install the required dependencies by running `pip install -r requirements.txt`.

5. Obtain an API key from OpenAI and set it as an environment variable (e.g., `OPENAI\_API\_KEY=your\_api\_key`).

Usage

1. Run the Streamlit application using the command `streamlit run app.py`.

2. The application will open in your default web browser.

3. In the sidebar, click the "Upload your PDFs here and click on 'Process'" button to upload your PDF documents.

4. Once the documents have been processed, you can start asking questions in the main panel.

5. The application will display the conversation history, with the user's questions and the AI's responses.

Code Structure

The main components of the application are:

1. `app.py`: The main Streamlit application file that handles the user interface and orchestrates the various functions.

2. `htmlTemplates.py`: A separate file that contains the HTML templates for the user and bot message displays.

3. `get\_pdf\_text(pdf\_docs)`: A function that extracts the text content from the uploaded PDF documents.

4. `get\_text\_chunks(text)`: A function that splits the extracted text into smaller chunks for efficient vector search.

5. `get\_vectorstore(text\_chunks)`: A function that creates the vector store using the OpenAI Embeddings and FAISS library.

6. `get\_conversation\_chain(vectorstore)`: A function that sets up the conversational retrieval chain using the vector store and the ChatOpenAI language model.

7. `handle\_userinput(user\_question)`: A function that processes the user's question, retrieves the relevant information from the vector store, and displays the conversation history.

Dependencies

- `streamlit`: A Python library for building interactive web applications.

- `dotenv`: A library for loading environment variables from a `.env` file.

- `PyPDF2`: A library for extracting text from PDF documents.

- `langchain`: A framework for building applications with large language models.

- `langchain-community`: A community-contributed package for the LangChain framework.

- `openai`: The official OpenAI Python library for accessing the OpenAI API.

- `faiss-cpu` (or `faiss-gpu`): A library for efficient vector search and clustering.

Customization

You can customize the application by modifying the following elements:

1. HTML Templates: Update the `htmlTemplates.py` file to change the appearance of the user and bot message displays.

2. Conversation Chain Configuration: Modify the parameters and settings of the `ConversationalRetrievalChain` in the `get\_conversation\_chain` function to customize the behavior of the conversational AI assistant.

3. Text Chunking: Adjust the parameters of the `CharacterTextSplitter` in the `get\_text\_chunks` function to change the size and overlap of the text chunks.

4. Vector Store: Experiment with different vector store implementations or configurations in the `get\_vectorstore` function.

Deployment

To deploy the ChatterMate application, you can use a cloud platform like Streamlit Sharing, Heroku, or AWS. The specific deployment steps will depend on the platform you choose, but the general process involves:

1. Preparing a deployment-ready version of your application.

2. Configuring the necessary environment variables (e.g., `OPENAI\_API\_KEY`).

3. Setting up the deployment platform and pushing your application.