Chat with PDFs using Streamlit and LangChain

Introduction

This project involves creating a Streamlit application that allows users to upload PDFs and interact with them through a chat interface. The application leverages the LangChain library to process and embed the PDF content, and utilizes OpenAI's GPT-3.5-turbo model to generate responses based on user queries. The primary goal is to provide a seamless experience for users to search and extract information from their PDF documents using natural language queries.

Technology Stack

1. **Streamlit**: A framework used for creating interactive web applications in Python.
2. **PyPDF**: A Python library to read and extract text from PDF files.
3. **LangChain**: A suite of tools to work with language models, including text splitting, embeddings, vector stores, and conversational agents.
4. **OpenAI GPT-3.5-turbo**: The language model used for generating responses based on the embedded PDF content.
5. **FAISS (Facebook AI Similarity Search)**: A library for efficient similarity search and clustering of dense vectors, used here to create a vector store for the PDF embeddings.

Workflow

1. PDF Upload and Text Extraction

- **File Upload**: Users can upload multiple PDF files via the Streamlit interface.
- **Text Extraction**: The `PdfReader` from the `PyPDF` library is used to read and extract text from the uploaded PDFs.

2. Text Preprocessing

- **Text Splitting**: The extracted text is split into manageable chunks using `CharacterTextSplitter` from the `LangChain` library. This ensures that the text can be efficiently processed and embedded.
    - **Chunk Size**: 1000 characters.

- **Chunk Overlap**: 200 characters.

3. Embedding and Vector Store Creation

- **Embeddings**: The text chunks are converted into embeddings using `OpenAIEmbeddings`.
- **Vector Store**: The embeddings are stored in a FAISS vector store, which allows for efficient similarity search and retrieval.

4. Conversational Agent Setup

- **LLM Initialization**: The `ChatOpenAI` class initializes the GPT-3.5-turbo model with the provided API key.
- **Retriever Tool**: A retriever tool is created using the vector store, enabling the agent to search through the PDF content.
- **Agent Creation**: An OpenAI Functions agent is created and configured to handle multiple consecutive queries.
- **Message History**: The chat history is maintained using `ChatMessageHistory`, which allows the agent to keep track of the conversation context.

5. User Interaction

- **User Query**: Users can input their queries through the Streamlit text input widget.
- **Response Generation**: The agent processes the user query and generates a response based on the PDF content.
- **Display**: The chat history and responses are displayed in the Streamlit application.

Implementation

Code Structure

1. **Main Script**: Contains the Streamlit application logic, including file upload, text extraction, embedding creation, and chat handling.
2. **Helper Functions**: Separate functions for text extraction, text splitting, vector store creation, and agent setup to modularize the code and improve readability.

Streamlit Application

- **Header**: Displays the title "Chat with PDFs".

- **Text Input**: Allows users to input their queries.
- **Sidebar**:
    - API Key Input: Users can input their OpenAI API key.
    - File Uploader: Users can upload multiple PDF files.
    - Process Button: Triggers the processing of the uploaded PDFs.

Processing Pipeline

1. **Upload PDFs**: Users upload PDFs via the sidebar.
2. **Extract Text**: Text is extracted from the PDFs.
3. **Split Text**: The extracted text is split into chunks.
4. **Create Embeddings**: Embeddings are generated for the text chunks.
5. **Create Vector Store**: A FAISS vector store is created from the embeddings.
6. **Initialize Agent**: The conversational agent is initialized with the vector store and chat history.
7. **User Interaction**: Users input queries and receive responses based on the PDF content.

Conclusion

This project demonstrates the integration of various libraries and technologies to create an interactive application for querying PDF documents. By leveraging Streamlit for the user interface, PyPDF for text extraction, LangChain for text processing and embeddings, and OpenAI's GPT-3.5-turbo for natural language understanding, the application provides a powerful tool for users to interact with their PDF documents in an intuitive and efficient manner.