Chetan Kumar Nageshwarrao Chitturi
B00963787

Termination Project
Git Assistant

Overview

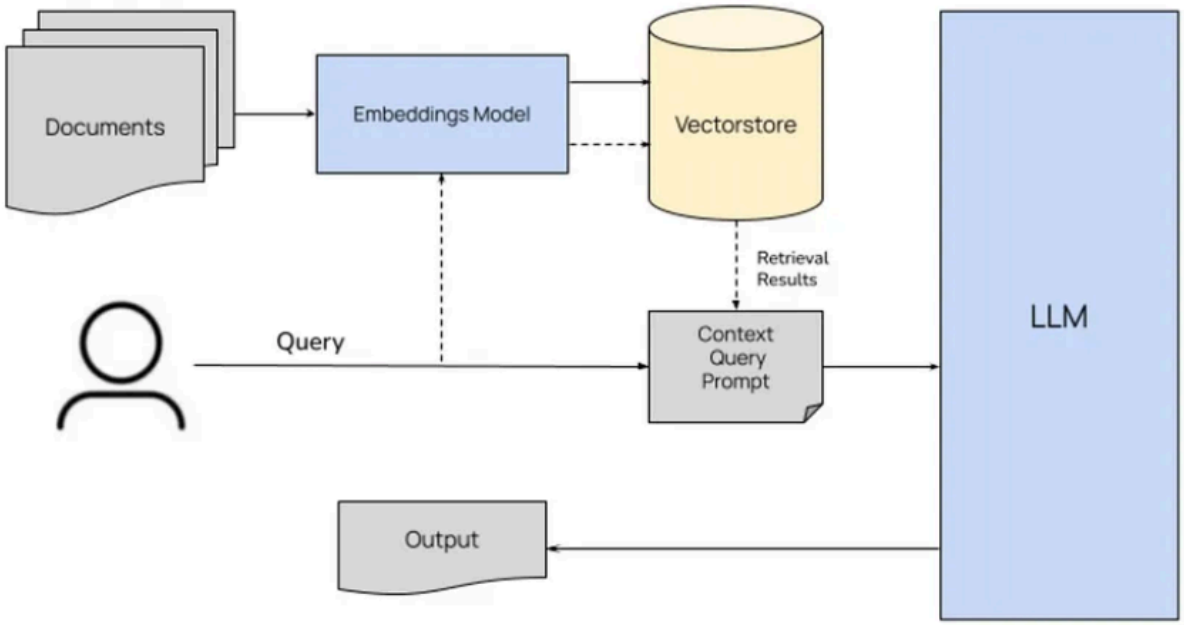
This application serves as a specialized tool designed to provide immediate, interactive support for users seeking help with Git documentation. By leveraging a conversational AI interface, the application allows users to pose questions and receive precise, context-aware responses derived directly from the Git documentation. The system utilizes advanced natural language processing techniques to extract, process, and understand content from a PDF document containing Git instructions, transforming it into an interactive and user-friendly Q&A service. This not only enhances the learning experience for users but also streamlines the search for specific information within the document, promoting an efficient way to grasp complex Git concepts and commands.

Technical Components

Libraries and Frameworks

- Streamlit: Used for building the web application interface.
- dotenv: Manages environment variables.
- PyPDF2: Extracts text from PDF documents.
- langchain: Facilitates the creation and management of conversational models and text processing.
- langchain_openai: Provides OpenAI embeddings and conversational models.
- langchain_community: Includes additional support for vector storage and retrieval.

Application Architecture



Key Functionalities

PDF Text Extraction:

The function get_pdf_text utilizes the PdfReader class from PyPDF2 to extract all text from a specified PDF file. The text from each page is concatenated into a single string.

Text Chunking:

The get_text_chunks function employs the RecursiveCharacterTextSplitter to divide the extracted text into smaller segments with a specified character limit and no overlap, making the text more manageable for processing.

Vector Store Creation:

Using the get_vectorstore function, text chunks are converted into embeddings using OpenAIEmbeddings and then stored in a FAISS vector store. This setup enables efficient retrieval of relevant text chunks for answering user queries.

Conversation Model with Memory:

The get_conversation_chain function sets up a conversational retrieval model using the vector store. A custom prompt template is used to generate responses based on the retrieved text and chat history. The model, inspired by the Retrieval Augmented Generation (RAG) concept, uses a vector store to retrieve relevant information that augments the chatbot's context, making it more informative and contextually relevant for the LLM from which it can generate response.

User Interaction:

The function handle_use_input handles user input from the Streamlit chat interface, managing the state of the conversation and updating the user interface with responses.

Accessing the Application

To run the application, first install the required dependencies using the command "**pip install requirements.txt**" and then simply execute the command "**streamlit run app.py"** from the terminal. This command starts the Streamlit server and automatically opens the default web browser to the application's URL, where users can interact with the conversational AI. The application provides an intuitive and interactive platform for querying Git documentation, utilizing advanced AI to ensure the responses are both accurate and relevant.

Conclusion

This Streamlit application harnesses the power of conversational AI to provide a dynamic and interactive user experience for individuals seeking to understand and utilize Git documentation effectively. By facilitating real-time, context-sensitive dialogues about Git, the application helps users to navigate and assimilate complex technical information with ease.