Brance AI engineer Task

Name: Hrishikesh Chougule

LinkedIn Profile: https://www.linkedin.com/in/hrishikesh-chougule-7937401b6

Date Challenge Received:22-7-23

Date Solution Delivered:24-7-23



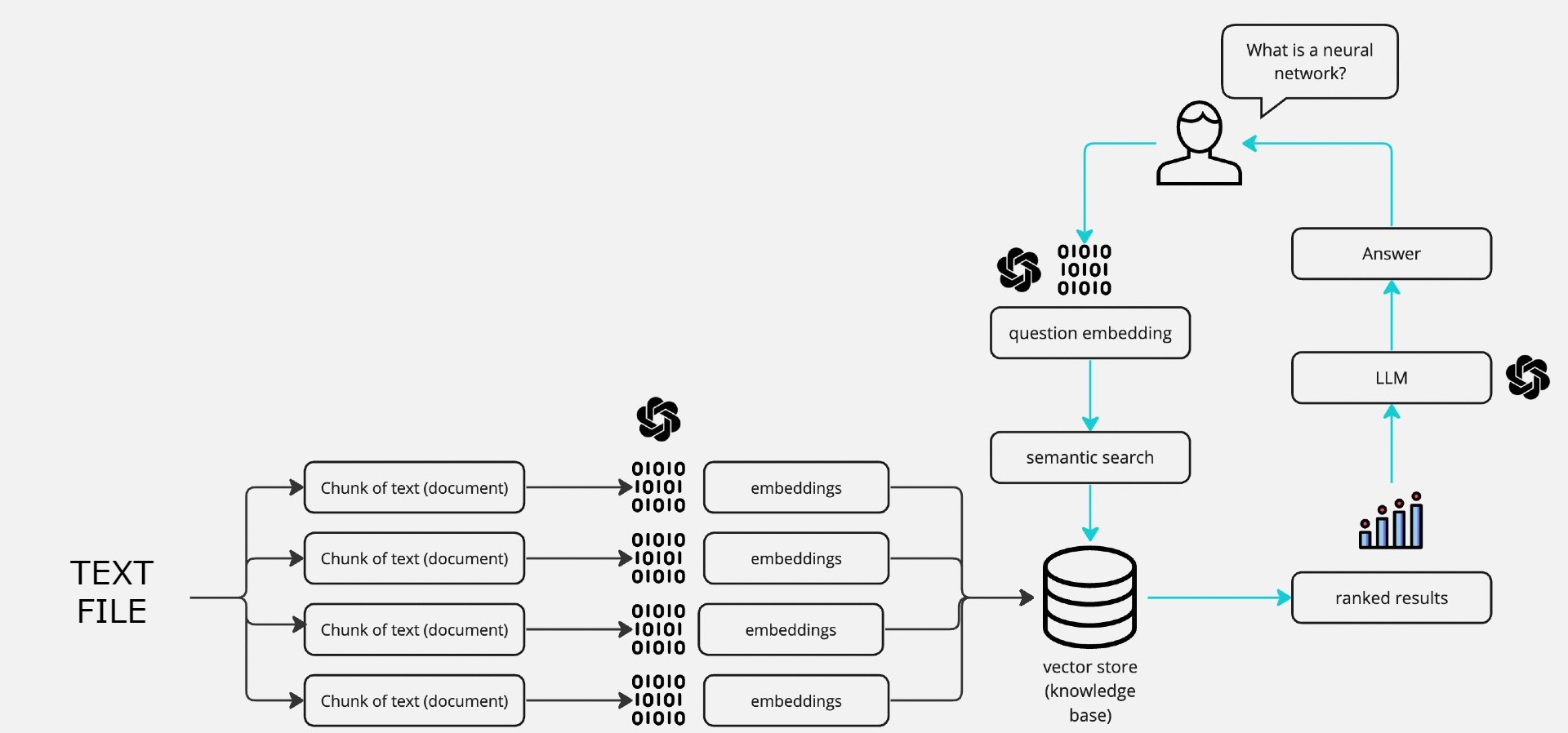
1. Problem Statement

Making a chatbot implementing the rag module finding out about the various embedding techniques applying the best one and feeding the data to any given llm to generate meaningful content.

Llms usually tend to give diverse and generic answers based on all the knowledge they are trained with. It becomes rather difficult for the individual to get focused data from a given dataset.

This is where the RAG technique comes in handy. This technique provides the user with the ability to focus on the restricting the knowledge base of the llm to a selected document. Thus, only relevant data from the document is matched and then sent to the llm to generate an articulated response from the relevant knowledge.

2. Approach



The general structure of our chat bot is seen in the figure above -

Text folder is inputted and read

It is then broken down into chunks taking into account the overlaps so as to ensure that no information is lost.

These chunks are then converted into embeddings (using OpenAI API calls)

The question from the user is inputted and also converted into embeddings.

Now the relevant knowledge is extracted by matching the embeddings.

These embeddings are then used to pass the relevant knowledge to the llm which generates the relevant output

3. Solution

The code is a Streamlit application that performs the following steps:

* **Input and Read Text**: The application takes a text file as input (specified by the **knowledge\_file\_path**) and reads its content.
* **Text Chunking**: The text is then broken down into smaller chunks to ensure that no information is lost. The chunking is done with consideration for overlaps using the **CharacterTextSplitter** from the **langchain.text\_splitter** module.
* **Embedding Generation**: The chunks of text are converted into embeddings using the OpenAI API calls (**OpenAIEmbeddings**). These embeddings are vector representations that capture the semantic meaning of the text.
* **User Question Input and Embedding**: The user can input a question. The question is also converted into embeddings using the same OpenAI API calls.
* **Knowledge Extraction**: The application matches the embeddings of the user's question with the embeddings of the text chunks to find the most relevant knowledge that answers the user's question. This is done using the **ConversationalRetrievalChain** from the **langchain.chains** module, which uses a language model (**ChatOpenAI**) to retrieve relevant information.
* **Chat Interaction**: The relevant knowledge is extracted and used to generate a response to the user's question. The conversation history is stored in **st.session\_state.chat\_history**, and the messages are displayed on the Streamlit web interface using templates (**user\_template** and **bot\_template**) to show user and bot messages alternately.
* **Processing and Initialization**: The Streamlit web application is initialized with a title, icon, and CSS styling. The user's conversation history and chat messages are stored in the **st.session\_state** to maintain state between user interactions.
* **Sidebar Processing**: The application has a sidebar that contains a "Process" button. When clicked, it processes the text file and sets up the conversation chain for responding to user queries. The processing includes reading the text file, chunking the text, generating embeddings, and creating the conversation chain.

The application allows users to ask questions about the contents of the input text file, and it responds with relevant information extracted from the text file using language model-based retrieval.

4. Future Scope

Looking for free alternatives for apis  
adding additional features like voice to text

Evaluation of answers and extending

model to beyond one document

Working with the embeddings to work on better alternatives that can be deployed without relying on OpenAIs embeddings.