Mental Health RAG Chatbot

Name: Jiawen Li
NUID: 002755483

1. System architecture diagram

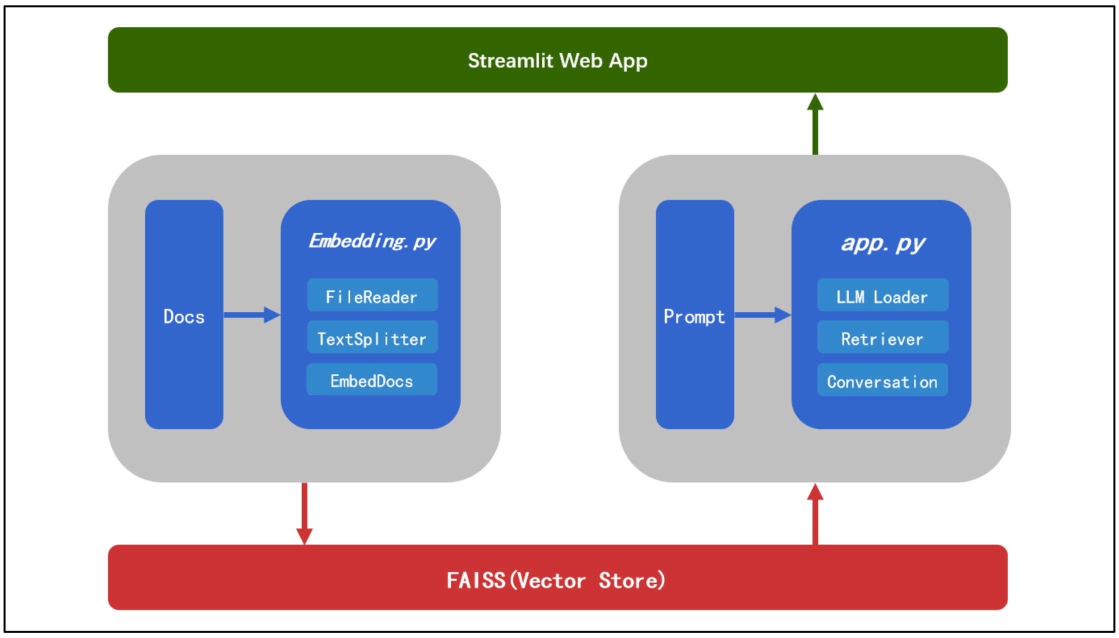

Fig1. System architecture diagram

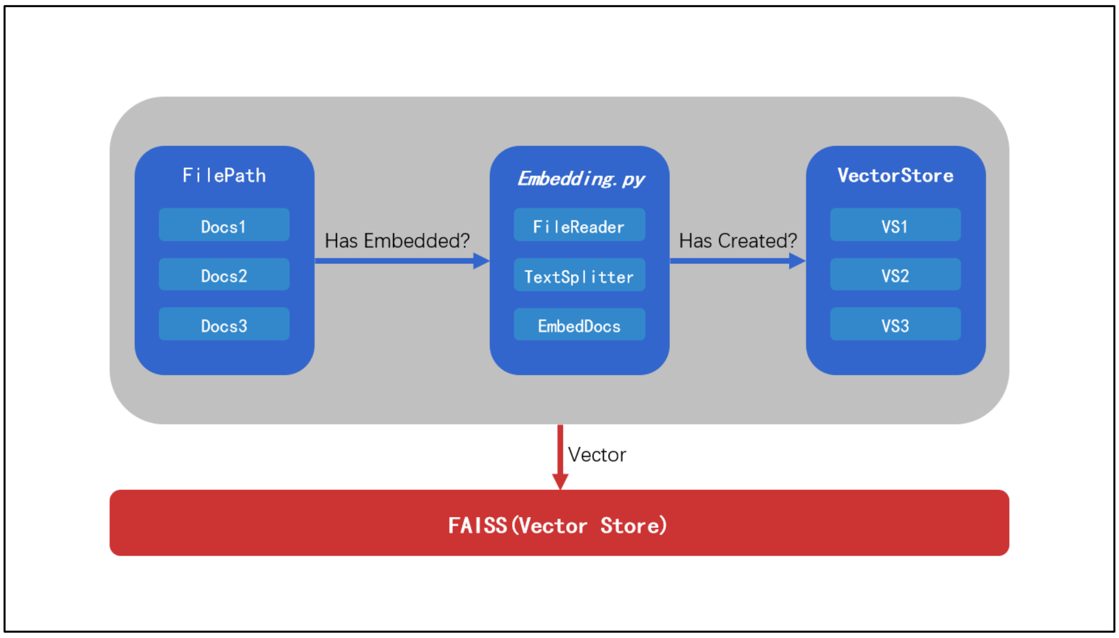

Fig2. Embedding

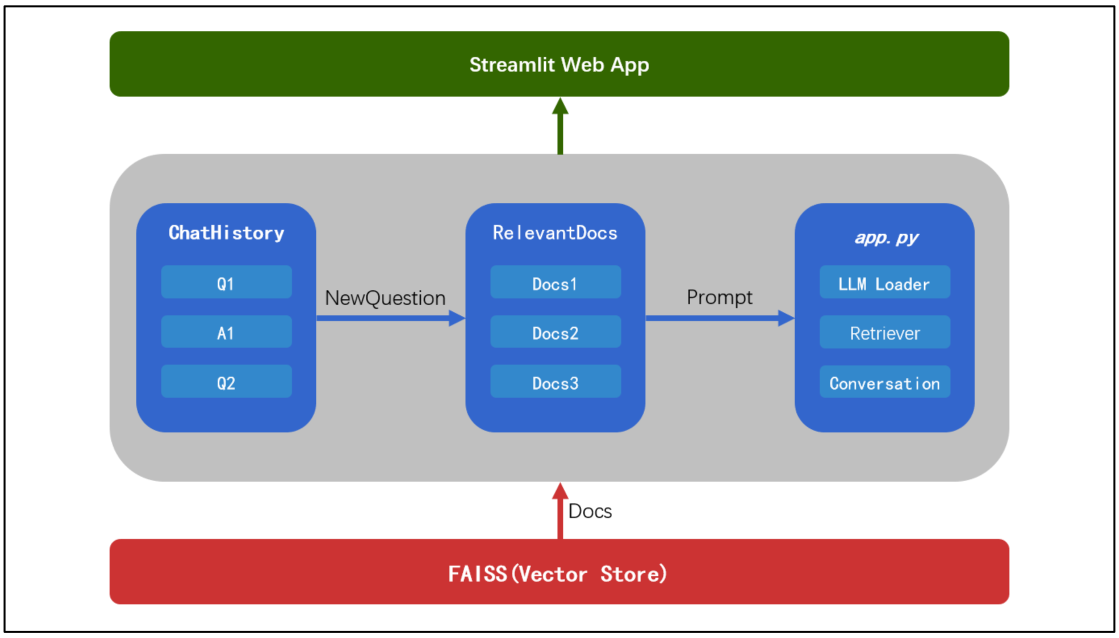

Fig3. Streamlit App

2. Implementation details

2.1. Text Preprocessing and Vectorization

Traverse Documents in the Directory: The program first traverses all the documents in the specified folder and checks whether each document has already been vectorized.

Check Embedding Records: For documents that have already been vectorized, the information is recorded in the embedded.json file. If a document is already listed, it is skipped; otherwise, it undergoes the vectorization process.

Vectorization Process: For documents that are not yet vectorized, the OpenAI Embeddings model is used to generate corresponding document embeddings.

FAISS Vector Store Handling: After vectorization, the program checks whether the corresponding FAISS vector store exists. If it does not, a new vector store is created; if it does, the existing vector store is loaded.

Store Vectors: Finally, the vectorized documents are stored in the corresponding FAISS vector store for subsequent querying and retrieval.

2.2. Question-Answer Processing

User Interaction: The user selects an LLM through the Streamlit interface and inputs a question.The app.py is responsible for receiving these inputs.

Conversation Management: Based on the chat history, the system selectively generates new questions, enabling support for multi-turn conversations.

Embedding Model Processing: The input question is processed using the Embedding model, and FAISS is queried to retrieve the most similar vectors to the question.

Document Retrieval: The system retrieves the corresponding document fragments based on the similarity scores from FAISS.

Answer Generation: The retrieved [Document] is passed to a predefined prompt template (located in the Prompts.py file), which further processes the text to generate the final answer.

Answer Display: The generated answer is then displayed on the Streamlit interface.

3. Performance metrics

To assess the system's performance, the following key metrics are considered:

Retrieval Speed: The FAISS vector search is used, with an average retrieval time of under 1 second (actual measurement values can be added). The efficient indexing structure of FAISS allows for rapid retrieval of the most relevant results even from large-scale document libraries.

Accuracy: The system retrieves documents based on the similarity between the user's query and the documents. The retrieved documents are highly relevant to the user’s question.

Response Time: The time taken from when the user submits a question to when the system returns an answer. Under normal conditions, the system’s response time is within 3 seconds.

System Stability: The system has maintained good stability during prolonged use, with no crashes or major issues observed.

4. Challenges and solutions

Challenge: How to ensure that each segment maximizes the quality of question-answering when splitting large text blocks in document.txt?

Solution: Initially, we used the RecursiveCharacterTextSplitter with appropriate settings for chunk_size, separator, and overlap. However, it did not perform well in testing. Considering that the content in the original document is not composed of long, continuous blocks of text, we opted to use line breaks for segmentation. This approach allows us to limit the length of each segment while preserving the key information in the document, thus improving the quality of the question-answering process.

Challenge: Different LLMs provide poor answers in certain problem scenarios.

Solution: We iteratively modified the prompt templates and adjusted the parameters of the similarity_search to improve the accuracy of the question-answering system.

Challenge: The aesthetic quality of the Streamlit interface.

Solution: We enhanced the Streamlit interface by incorporating streaming output when receiving responses from the LLM. This improves the visual appeal of the answers presented on the interface.

5. Future improvements

5.1. In the future, we hope to support more document formats, such as PDFs. This will likely require the integration of OCR and layout analysis to achieve more accurate information extraction.

5.2. We also aim to add support for multiple vector databases. This would allow users to select different knowledge domains for question-answering based on the specific database they wish to query.

5.3. Moving forward, we plan to further improve the accuracy of document retrieval. This could involve employing different algorithmic search strategies, adding re-ranking techniques, and other optimizations to enhance search performance.