**THE REPORT OF THE QUESTION & ANSWERING (FAISS+RAG+LLM) PROJECT**

**Süleyman Emre PARLAK**

**Computer Engineering Student at Muğla Sıtkı Koçman University**

**INTRODUCTION**

In web analytics, effectively answering queries about web traffic logs can provide crucial insights into user behavior and site performance. To address this need, we are developing a Question-Answering (Q&A) system based on the Retrieval-Augmented Generation (RAG) model. This system is designed to handle natural language questions, retrieve relevant log data, and generate accurate responses.

The RAG model integrates information retrieval with generative capabilities, allowing the system to first find pertinent documents from web traffic logs and then produce contextually relevant answers. This report outlines the system's architecture and implementation, focusing on data processing, indexing, and response generation to enhance insights from web traffic data.

The first five index of our data:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Our data is in txt structure, and when we examine each log structure; We can see that it is divided into ip, date-time, request method, path, protocol, status code, and response size.

It was successfully separated using Regular Expressions.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Then, it was checked to see if there was a faulty transaction or a missing value.



Considering that better results would be obtained in vectorization operations and response operations, sentences were created with features extracted with regular expression.

For instance : **User with IP 10.223.157.186, made request of type GET, on date and time 15/Jul/2009:14:58:59 -0700, accessed request path /, used HTTP/1.1 protocol, returned 403 status code, received 202 bytes of data.**

**DIAGRAM**

**metin, ekran görüntüsü, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**DESCRIPTIONS AND STRUCTURE OF METHODS**

**class LogVectorizer()**

**convert\_it\_to\_text() :** The convert\_it\_to\_text method processes a pandas DataFrame by extracting the contents of its 'text' column and writing each entry as a separate line into a designated text file specified by self.text\_file. This method iterates through all entries in the 'text' column, converting each to a string (if not already one) and appending a newline character to ensure proper formatting in the output file

**split\_it() :** The split\_it method loads text from a specified file (or a default file if no path is provided) and splits it into smaller chunks of 250 characters using the RecursiveCharacterTextSplitter. It then returns the resulting split documents.

**create\_and\_save\_faiss\_index() :** The create\_and\_save\_faiss\_index method generates embeddings for the provided documents using the specified embedding model (defaulting to HuggingFaceBgeEmbeddings), creates a FAISS index from these embeddings, and then saves the index to a file for future use.

**load\_faiss\_index() :** The load\_faiss\_index method reads and returns a previously saved FAISS index from a file, allowing it to be reused without needing to recreate the index from scratch.

**class ResponseGenerator()**

**generate\_response() :** The generate\_response method takes a question and context, combines them into a single input string, and then encodes this string using the tokenizer. It uses the pre-trained model to generate a response based on the encoded input, applying beam search to improve the response quality. After generating the output, the method decodes the response back into text and returns it. This process enables the model to provide answers based on the provided context.

**class System()**

**process\_and\_index\_data() :** The process\_and\_index\_data method processes a DataFrame by first converting its text column into a text file. It then splits this text into smaller parts and uses these parts to create and save a FAISS index. This indexing allows for efficient searching and retrieval of relevant documents based on the text data.

**give\_us\_answer() :** The give\_us\_answer method starts by loading the FAISS index for efficient document retrieval, then initializes the HuggingFaceBgeEmbeddings model to create an embedding for the question. It reshapes this embedding and searches the FAISS index to find the closest matching documents. It then checks the validity of the retrieved index to fetch the relevant context; if valid, it retrieves the document content, otherwise, it sets a default message. Finally, it generates a response using the generate\_response method with the question and the retrieved context, and returns this response.

**EVALUATION**

**FIRST TRY**

I chose a line from the raw data to try it



and I asked the system a question about it.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Output





**SECOND TRY**

****

****

**THIRD TRY**

****

****

Let’s check the result .



**INFERENCES :** Apparently, our system isn't working too badly at all. It is clearly seen in the second example that the sentence structures I chose while preprocessing the data greatly affected the accuracy of the outputs. However, looking at other outputs, it is obvious that the system mostly works correctly. Additionally, it seems that the model we use is insufficient to produce answers in natural language and produces monotonous answers.

**WHAT CAN WE DO FOR BETTER RESULTS?**

- Stronger LLMs can be used.

- It can be developed with systems such as "Contextual Embedding", "Concatenation of Prompts", "Session State Management", "Memory Networks", "External Storage Systems", where the model will keep previous questions and answers together and take this history into account with each new question.

- We can explore different model architectures or embedding techniques that can better fit our data.

- We can apply more sophisticated methods to ensure that the context returned is highly relevant to the question.

- GPU can be used to increase speed.

- Instead of beam search, more diverse and accurate answers can be produced by using techniques such as top-k sampling or nucleus sampling.

- More efficient solutions can be produced by examining powerful frameworks such as LangChain in more detail.

- Attention should be paid to the compatibility of the model and the embedding tool.

**WHAT I DID STEP BY STEP AND THE CHALLENGES I FACED**

- Since I had no previous experience with LLMs, RAG and Vector Databases, I did long research.

- I examined the structure of the server logs and did research to understand it.

- I repeated the Regular Expressions topic.

- I tried a lot to decide which size should be when splitting the data.

- I researched the structure of FAISS and how to embed it.

- Since the data was very large, I reduced the data size as it took a long time to convert it to vector and save it to FAIS.

- It was difficult for me to figure out which method should be located where and how they should be connected to each other.

- When I ran the main class, I received warnings that there were incompatibilities in some connections or that the embedding architecture was not compatible with input. These were later corrected through various attempts.

- Establishing connections for the model to search for input from the database and then generate a response was a bit confusing.

**CONCLUSION**

In this project, a Question and Answer (Q&A) system based on web traffic logs was developed. The system uses the Retrieval-Augmented Generation (RAG) model to respond to questions in natural language, search for relevant log data, and produce correct answers. During the project process, since there was no previous experience with LLMs, RAG and vector databases, extensive research was conducted and the structure of the server logs was examined in detail. Due to the long processing times encountered in the data processing and FAISS indexing stages, it was necessary to reduce the data size. Additionally, the integration of methods and incompatibility problems have been overcome through various attempts.

As a result, the system provides generally satisfactory results and a significant improvement in response accuracy has been achieved. However, the variety and naturalness of the answers offered by the model remain limited. In this context, the use of more powerful language models and alternative embedding techniques can improve response quality. Additionally, GPU support and implementation of techniques such as top-k sampling can significantly improve the performance of the system. Examining powerful frameworks such as LangChain in more detail and paying attention to the compatibility of the model and the embedding tool can contribute to obtaining more efficient results. Overall, this project is considered an important step in gaining deeper insights from web traffic data.

Thank you…

Süleyman Emre PARLAK