**Project Title**: RAG-based Web Logs LLM  
**Author**: Mehmet Özkaya  
**Date**: 22.08.2024

**Contents**:

* Project Summary and Description
* Methods
* Findings
* Discussion
* Conclusion
* References

**Project Summary and Description:**

This project was created to integrate data retrieved using the RAG system into a large language model, enabling it to answer questions about web server logs. After processing the user-entered query, the most relevant records in the database are found and provided to the large language model to generate a response. The aim of the project is to enable any user to perform an operation that would typically require programming expertise, simply by entering a query.

* **Methods:**

1. **Data Preprocessing and Vectorization:**

The dataset for this project consists of web server access logs from an Iranian e-commerce website, obtained from Kaggle (link in the references). The log data needs to be properly analyzed to be used in the project. Each log should be read from the file, parsed according to the log pattern, and stored in a dictionary as needed. During this process, popular Python data preprocessing and manipulation libraries such as re, Pandas, and Numpy can be used, and have been used in this project. Storing each log in dictionary format allows easy access to desired attributes via keys. After creating dictionaries, an index number is added to each log. For RAG-based systems to function, it is necessary to incorporate semantic knowledge and meaningfully combine elements within the dictionary. Therefore, using dictionary data, each log is converted into a sentence and assigned to the ‘context’ key of the dictionary. The vectorization process then begins.

The primary goal of vectorization is to convert sentences stored in a ‘string’ data type, which the machine cannot understand, into numerical data that the machine can comprehend. In this project, the SentenceTransformer class, obtained from Python's sentence\_transformers library, was used to perform this task. This class facilitates loading pre-trained language models that can convert sentences into vectors. The model ‘all-mpnet-base-v2’ was used in this project for vectorization, and it was transferred to the GPU for faster processing. All sentences are converted into vectors and then assigned to the ‘embedding’ key of each log dictionary. Subsequently, the ‘embedding’ data of all logs is collected into a single variable called ‘embeddings’, and our vectorized sentences are ready.

1. **Vector Database:**

In this project, FAISS, a vector database, was used to store and search the vectorized sentences. FAISS is a vector database developed by Facebook for storing vector data and researching vector similarities using fast and optimal methods. It is highly effective in storing large amounts of data and finding similar vectors.

The FAISS vector database takes the vectors you previously created, creates indexes based on these vectors, and stores them. FAISS can find similarities between vectors using multiple methods. Some of the most commonly used methods include ‘dot scores’, ‘cosine similarity’, or ‘Euclidean similarity’. Depending on the method used, calculations are performed among high-dimensional vectors, and as many similar vectors as desired can be retrieved. In this project, in addition to the similar vectors obtained with the FAISS library, there are also vectors calculated with the dot\_scores function in the sentence\_transformers.util module, and it was concluded that all vectors were identical.

1. **RAG (Retrieve-Augmented Generation):**

The project consists of a large language model containing the Retrieval Augmented Generation (RAG) method. RAG-based large language models can produce much more consistent and solid responses on a specific topic. These models refer to a specific external information source before generating their responses and then produce their answers with the data obtained from there. They consist of two separate components and can produce more accurate and comprehensive answers by combining these two components.

1. **Retriever:**

Responsible for finding and retrieving relevant information from a large document or data set (database, web pages, PDF files, etc.). The data to be used in answer generation can be stored in a vector-based database. It selects the most appropriate and closest results from the database based on the user's query.

1. **Generator:**

This component takes the data retrieved by the Retriever component from the database and generates a response using a large language model. Examples of large language models include GPT, T5, and BERT. It turns the data retrieved from the database into natural language responses.

The RAG model we created will retrieve the closest vectors in the FAISS vector database based on the user’s query, then, after the decoding process of these vectors, it will provide them to the LLM model for generating answers.

1. **Retrieval:**

The retrieval process begins when the user enters a query, and the query is vectorized using the language model stored in the ‘embeddings’ variable. Using the same language model is essential because each language model performs the embedding process in a way unique to its structure. Then, the nearest vectors to this query vector are retrieved from the index created by the FAISS vector database. This search provides the similarity/distance of the vectors and their indices. Without needing the decode phase, the indices obtained from the dictionary data are fetched, and the ‘string’ data type sentences are taken from their ‘context’. Thus, the sentences most relevant to the query are accessed from a database.

1. **LLM (Large Language Model):**

Large language models are AI models trained on vast datasets and designed to answer natural language questions. LLMs are highly successful in various language tasks such as text generation, understanding, summarization, translation, question-answering, and more. A large number of parameters are optimally adjusted during the lengthy training phase and designed to produce the best possible answers to incoming questions.

In this project, the large language model used is Google’s Gemma 7B Instruction-tuned model. The 7B refers to 7 billion parameters, and Instruction-tuned (it) indicates that it has been tuned to perform better on a specific task or set of tasks. The 2 billion parameter version of the same model was tried, but it was concluded that the answers were insufficient. The 7 billion parameter version produced answers at the desired level but increased the cost. High-parameter language models run on GPUs and require a significant amount of computational resources. GPU VRAM capacity must be checked before use; otherwise, the model may stop working or produce irrelevant results. To reduce computational costs, the number of bits in the large language model was reduced, and the ‘quantization\_config’ parameter was adjusted.

The large language model in the project was downloaded from HuggingFace using the transformers library. The tokenizer for this model was also obtained in the same way.

1. **Augmentation (Integration):**

The integration phase combines the Retrieval and Generation phases. The records obtained during the Retrieval phase are given to the large language model, which analyzes them and generates a response. In this project, the user’s query is first taken, vectorized, compared with the vectors in the vector database, and the indices and similarities of the closest vectors are accessed. Then, the log records stored in the dictionary data type with the determined index values are retrieved with all their information and given to the large language model along with the query.

1. **Generation:**

A special query preparation function has been created for the large language model to produce accurate answers, and this function sets the Instruction-tuned language model on how to generate a response, which records to use, and the query. The large language model generates responses based on the query and the records.

* **Findings:**

When the generated responses were examined, it was concluded that the large language model had sufficient capacity to produce responses and analyze the retrieved data. If the retrieved records are the most relevant and correct records for the given query, the large language model produces correct answers as desired. However, depending on the depth of the query, incorrect data may sometimes be retrieved during the data retrieval phase. The reasons for this are discussed in the ‘Discussion’ section below.

* **Discussion:**

One of the most important reasons for incorrect data retrieval is the similarity between the log data and the sentences generated from this data, which are compared for similarity. This is because all log data has a specific format, and only a few words in the sentence create a difference. For example, during the processing of log records, sentences are created by placing different words into a specific template. While hundreds of different sentences can be found in a PDF file, and similar sentences can be found with high accuracy for the query made, log data does not have this feature. The same browser or operating system appears in hundreds of log records. Also, the model was tested with dictionary datasets instead of sentences, but it was concluded that success was higher with sentences. The inability of the LLM model to fully understand the structure in the retrieved results may also be a reason for incorrect answers. LLM models with more parameters, much more comprehensive versions of the language models performing the embedding process, and a more consistent and understandable dataset could produce much better results, but the computational cost will increase.

* **Conclusion:**

In conclusion, the desired RAG-based LLM model has been created and can produce output. Although most of the retrieved data is correct, there is a possibility of making mistakes depending on the depth of the query. If the issues discussed in the ‘Discussion’ section can be overcome, a much better model can be created.

* **References:**
* <https://chatgpt.com/>
* <https://www.kaggle.com/datasets/eliasdabbas/web-server-access-logs>
* https://www.youtube.com/watch?v=qN\_2fnOPY-M&t=14036s