HPE CTY Project Report

Team 04

Project Title:

AI powered log parsing tool

Objective:

To develop a robust AI-powered solution that utilizes LLMs to parse device logs, identify patterns and anomalies, and provide actionable insights to diagnose network issues.

Team Members:

Dhanushree B	4VV21IS033
Karthik R	4VV21CS073
Khushi Kashinath	4VV21CS078
Rohan Vijay	4VV21CI044
Yashas J Kumar	4VV21CI059

College Mentor:
Neeti Shukla
Assistant Professor, Dept of ISE, VVCE

HPE Mentor:
Mr. Sribharath Doopati
sribharath.doopati@hpe.com

Table of Contents :

Table of Figures :

1. Abstract:

The rapid growth of networked devices has led to an exponential increase in the volume and complexity of device logs. Effective analysis of these logs is crucial for diagnosing network problems and ensuring optimal performance. This project aims to harness the power of Generative AI and Large Language Models (LLMs) to automate the parsing and interpretation of device logs, transforming raw data into meaningful insights.

2. Development / Runtime environments

Streamlit Cloud Environment

3. Outcome of the Project

We have developed a robust AI-powered solution that utilizes LLMs to parse device logs, identify patterns and anomalies, and provide actionable insights to diagnose network issues.

4. Project Requirements

- Python 3.11
- Python Libraries:
    - streamlit~=1.35.0
    - langchain~=0.2.1
    - python-dotenv~=1.0.1
    - langchain-groq~=0.1.4
    - langchain-community~=0.2.1
    - faiss-cpu~=1.8.0
    - langchain-google-genai~=1.0.6
- GROQ API KEY
- Google API KEY
- OS Compatible to run python 3.11

5. Technical Approach: Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an advanced technique that combines information retrieval and generative modeling to enhance the performance of AI systems. RAG leverages the strengths of both approaches: it retrieves relevant documents or data points from a large corpus and then uses a generative model to process and produce contextually appropriate and coherent outputs based on the retrieved information.

RAG is particularly effective in complex tasks such as question answering, content creation, and problem-solving scenarios where access to diverse and structured information is crucial. Moreover, by dynamically selecting and integrating retrieved data into the generation process, RAG models can adapt to different domains and specific user queries, making them highly versatile and powerful tools in modern AI applications.

6. Importance of RAG in Meeting the Project Objectives

RAG is particularly well-suited for our objective of parsing device logs and diagnosing network issues for several reasons:

- **Contextual Understanding:** The retrieval component fetches relevant log entries, ensuring that the generative model has access to pertinent information, which enhances its ability to understand and interpret the context of the logs.
- **Enhanced Accuracy:** By combining retrieval with generation, RAG improves the accuracy of the analysis. The retrieval ensures that the generative model works with the most relevant data, reducing the chances of generating irrelevant or incorrect outputs.
- **Scalability:** RAG can handle large volumes of log data efficiently. The retrieval step narrows down the data to only the most relevant entries, making it feasible to process and analyze extensive log files.
- **Anomaly Detection and Pattern Recognition**: The generative model can identify patterns and anomalies within the context of the retrieved log entries, providing deeper insights into potential network issues.
- **Actionable Insights:** RAG produces detailed and actionable insights by contextualizing the retrieved log data, helping network administrators quickly understand and address network problems.

7. Implementation of the RAG in the Project

1. Load the log or text file.
2. Split the data into a number of chunks.
3. Use Embedding-001, an embedding model, to convert the text chunks into vectors.
4. Store the vectors inside a FAISS vector database.
5. Configure a Llama3-70B-8192, a Llama 3 model, as the generative model.
6. Define the prompt template suitable for the corresponding model.
7. Define a document chain using the LLM and the prompt.
8. Configure and use a retriever to retrieve the top k (2) semantically similar chunks as context.
9. Configure a retrieval chain using the retriever and document chain to answer any query related to the provided logs using a streamlit UI.

Each of these steps are explained in details in the below sessions,

7.1 Step 1: Load the Log or Text file

Users initiate the process by uploading either a log or text file via a dedicated user interface. The system automatically identifies the file type by comparing its content against predefined log templates**: Kernel Logs, OVS Logs, dmesg Logs, and sys Logs.**

- If the uploaded file matches one of these templates, it undergoes a transformation into a structured format, typically a CSV file. This structured data format ensures that specific fields and attributes relevant to each log type are organized systematically.
- Conversely, if the uploaded file does not match any of the predefined templates, it is recognized as unstructured data and remains in its original format.

The loading process involves utilizing two distinct loaders:

- **CSV Loader:** This loader is employed specifically for handling structured files, ensuring efficient loading and preparation of data into the system.
- **Text Loader:** Designed for unstructured files, the text loader manages the ingestion of raw text data, preserving its original format while preparing it for further analysis.

Upon completion of the loading process, both structured and unstructured data are processed and made available for subsequent stages of analysis and manipulation. This initial step ensures that all incoming log and text data is accurately categorized and prepared for further analysis according to project requirements.

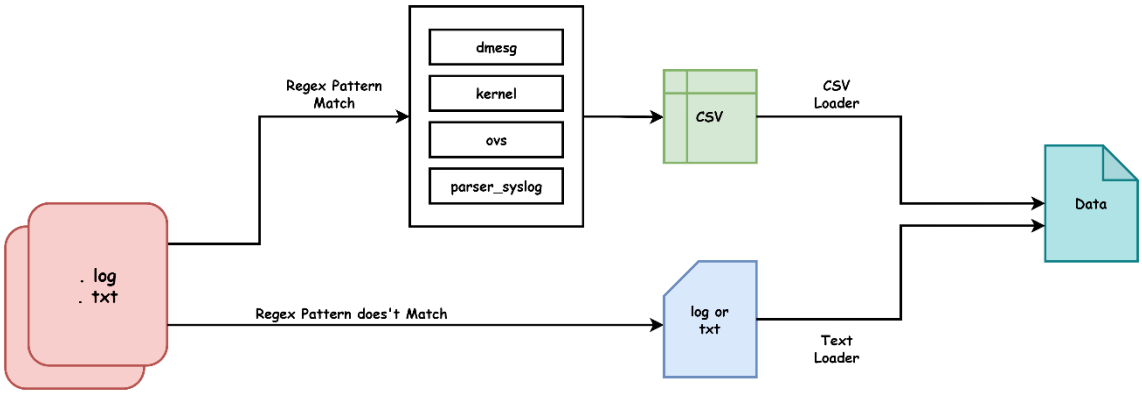

Fig 1. Loading the Data – Information Extraction

Code snippet:

```
loader = CSVLoader(file_path)
or
loader = TextLoader(file_path)
```

7.2 Step 2: Split the data into a number of chunks

To split the data into chunks of 1024 characters each, with a 60-character overlap between consecutive chunks, we have utilized the **RecursiveCharacterTextSplitter** with the specified parameters:

- **chunk_size:** Each chunk of data is defined to contain up to 1024 characters.
- **chunk_overlap:** Subsequent chunks overlap by 60 characters to ensure continuity and context preservation across chunks.
- **length_function=len:** This indicates that the length of the data (in characters) determines how it is split into chunks.

This approach ensures that the original data is segmented into manageable chunks suitable for further processing, such as embedding and subsequent analysis tasks.

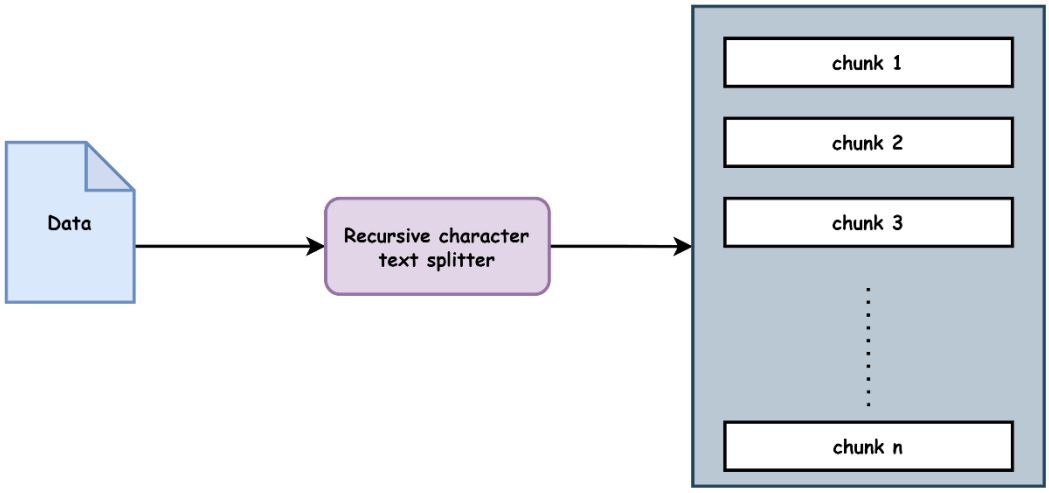

Fig 2. Chunking

Code snippet:

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1024,
    chunk_overlap = 64,
    length_function = len
)
texts = text_splitter.split_documents(docs)
```

7.3 Step 3: Use Embedding-001, an embedding model, to convert the text chunks into vectors

In this step, we use **embedding-001**, an advanced model designed to convert text chunks into numerical vectors. This model, **embedding-001 by GoogleGenerativeAIEmbeddings**, is specifically configured to optimize the embedding process by capturing semantic and contextual information from the text.

These vectors facilitate efficient analysis and retrieval tasks, capturing nuanced relationships and meanings within the original text chunks. This transformation into vectorized format supports tasks such as similarity calculations, classification, and content generation based on text properties.

Overall, this step plays a pivotal role in preparing textual data for advanced processing and insights extraction, adhering to best practices in natural language processing (NLP) to enhance accuracy and efficiency in data-driven applications.

Code snippet:

```
embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001" )
```

7.4 Step 4: Store the vectors inside a FAISS vector database

In step 4, the vectors generated from the text chunks are stored within a FAISS vector database.

FAISS, or Facebook AI Similarity Search, is a library designed for efficient similarity search and clustering of dense vectors. By storing the vectors in a FAISS vector database, the system optimizes retrieval operations based on similarity metrics.

This approach enables rapid querying and retrieval of vectors, supporting tasks such as nearest neighbor search and similarity-based analysis.

Storing vectors in FAISS ensures that the system can handle large-scale vector data effectively, leveraging its indexing and search capabilities to enhance performance in applications requiring fast and scalable vector retrieval. This step is integral to maintaining the efficiency and effectiveness of subsequent stages in the data analysis pipeline.

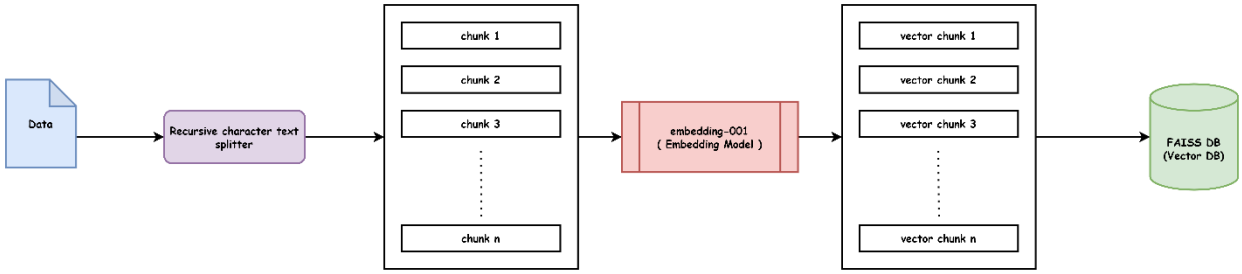

Fig 3. Storing the Embeddings in a Vector Storage

Code snippet:

```
db = FAISS.from_documents(texts, embeddings, persist_directory="db")
```

7.5 Step 5: Configure a Llama3-70B-8192, a Llama 3 model, as the generative model.

In this step, we configure a Llama3-70B-8192, specifically a Llama 3 model, to serve as our generative model. This model deployment is hosted on GROQ Cloud, accessible via a GROQ API Key.

The Llama3-70B-8192 model represents a state-of-the-art generative AI model capable of producing text based on given prompts. By configuring it within GROQ Cloud and utilizing the API Key for access, we ensure seamless integration and reliable performance for generating outputs based on specific inputs.

This setup allows us to leverage the capabilities of Llama 3 for tasks such as text generation, content creation, and contextual understanding, supporting diverse applications in natural language processing (NLP) and AI-driven content generation.

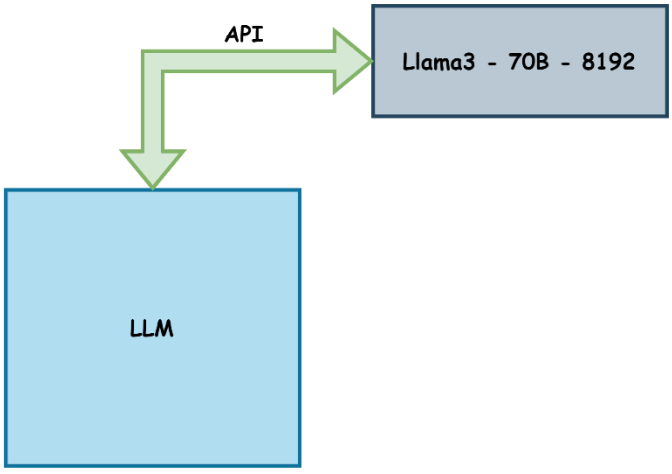

Fig 4. Accessing the Remotely Deployed LLM using an API

Code snippet:

```
llm = ChatGroq (groq _api_key = groq_api_key, model_name = "Llama3-70b-8192")
```

7.6 Step 6: Define the prompt template suitable for the corresponding model

In this step, we define a prompt template tailored to the characteristics and capabilities of the Llama3-70B-8192 model deployed on GROQ Cloud. The prompt template serves as the structured input format that guides the generative model in producing coherent and contextually relevant outputs based on specific queries or requirements.

This step ensures that the model receives clear and structured prompts, optimizing its ability to generate accurate and meaningful responses aligned with the intended use case.

The Below Prompt Template is used:

SYSTEM	""" You are an advanced AI assistant integrated with a RAG (Retrieval-Augmented Generation) system, "specialized in log analysis. Suggest next steps or further investigations when appropriate. If you don't know the answer just say that you don't know. Don't try to make up an answer based on your assumptions. "Response Format: Structure your responses clearly, using sections or bullet points for complex analyses. Include relevant log messages when explaining your findings. Clearly distinguish between information from logs, retrieved knowledge, and your own analysis. Provide the final answer to the question first in bold. Clarification and Precision: If log formats or contents are unclear, ask for clarification. """
USER	"The Log Data is as follows: {context}. User Question: {input}"

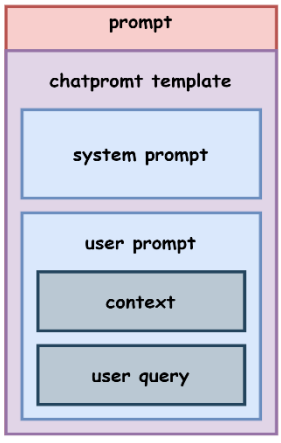

Fig 5. Structure of Prompt Template

Code snippet:

```
prompt = ChatPromptTemplate.from_messages(
   [
      ("system", """You are an advanced AI assistant integrated with a RAG (Retrieval-Augmented Generation) system, "specialized in log analysis. Suggest next steps or further investigations when appropriate. If you don't know the answer just say that you don't know.Don't try to make up an answer based on your assumptions.
"Response Format:
Structure your responses clearly, using sections or bullet points for complex analyses.
Include relevant log messages when explaining your findings.
Clearly distinguish between information from logs, retrieved knowledge, and your own analysis.
Provide the final answer to the question first in bold.
Clarification and Precision: If log formats or contents are unclear, ask for clarification.
"""
       ),

      ("user", "The Log Data is as follows : {context}. User Question : {input}")
   ]
)
```

7.7 Step 7: Define a document chain using the LLM and the prompt

In step 7, we establish a document chain utilizing the LLM (Large Language Model) and the predefined prompt.

The **document_chain** variable is initialized using a function **create_stuff_documents_chain,** configured to incorporate the LLM and the specified prompt. This chain formation links the capabilities of the LLM with the structured prompt, facilitating a streamlined process for document analysis, content generation, or other tasks aligned with the project objectives.

This integration ensures that the LLM interacts effectively with the provided prompt, enabling comprehensive document handling and insightful data processing in subsequent stages of the workflow.

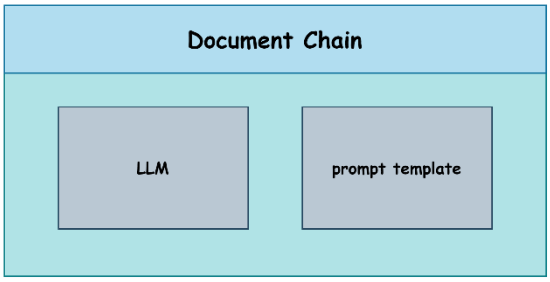

Fig 6. Structure of the Document Chain

Code snippet:

```
document_chain = create_stuff_documents_chain(llm=llm, prompt=prompt)
```

7.8 Step 8: Configure and use a retriever to retrieve the top k (2) semantically similar chunks as context.

In step 8, we configure and utilize a retriever to fetch the **top k (2)** semantically similar chunks, serving as context for further analysis or processing.

The configuration involves initializing the retriever using the `**as_retriever**` method from the database (`**db**`), specifying search parameters through `**search_kwargs = {"k": 2}**`. This setup ensures efficient retrieval of chunks that closely match the semantic content of the query or input, enhancing the relevance and accuracy of subsequent operations.

By leveraging the retriever, the system can dynamically gather contextually relevant information, supporting tasks such as content summarization, contextual understanding, and response generation based on the retrieved chunks. This step

plays a crucial role in enriching the analysis process by providing meaningful context derived from the dataset or document repository.

First, the user query is converted into a vector representation. The retriever then utilizes this vector to identify the top 2 semantically similar vectors from the FAISS database. These retrieved vectors serve as contextual information provided to the LLM (Large Language Model) for further processing or analysis. This approach ensures that the LLM receives relevant and contextually aligned data, enhancing its ability to generate accurate responses or insights based on the user query.

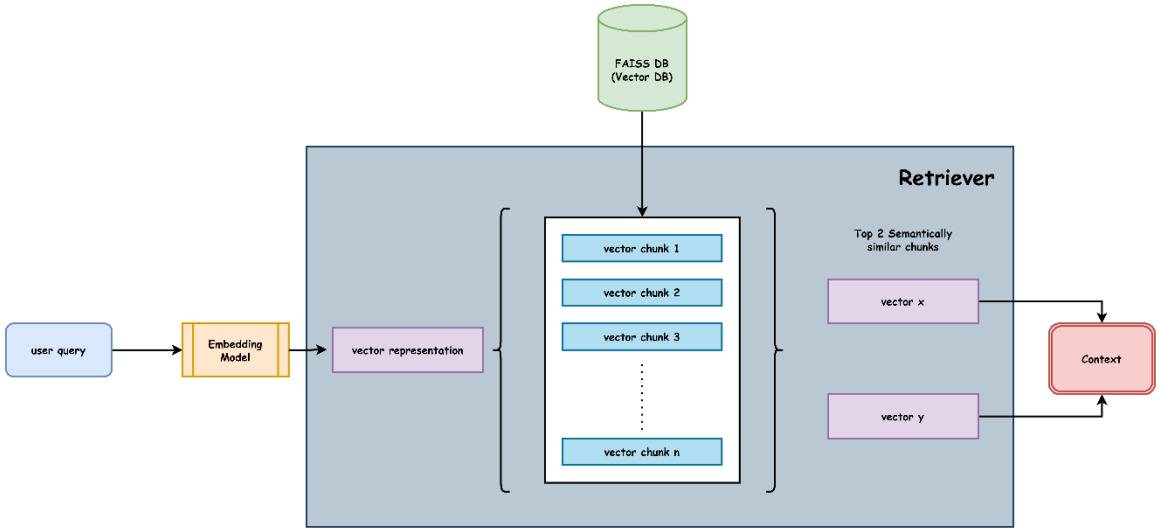

Fig 7. Working of the Retriever

Code snippet:

```
retriever=db.as_retriever(search_kwargs={"k": 2})
```

7.9 Step 9: Configure a retrieval chain using the retriever and document chain to answer any query related to the provided logs using a streamlit UI.

In step 9, we establish a retrieval chain integrating the retriever and document chain to handle queries related to the provided logs, utilizing a Streamlit user interface (UI).

The retrieval_chain is initialized using the **create_retrieval_chain** function, which incorporates both the retriever and document_chain components. This setup enables seamless processing and retrieval of information relevant to log-related queries through the **Streamlit UI**.

By integrating these components, the retrieval chain ensures efficient querying and retrieval of contextually relevant data, empowering users to interactively explore and analyze log information using intuitive and responsive UI interactions provided by Streamlit.

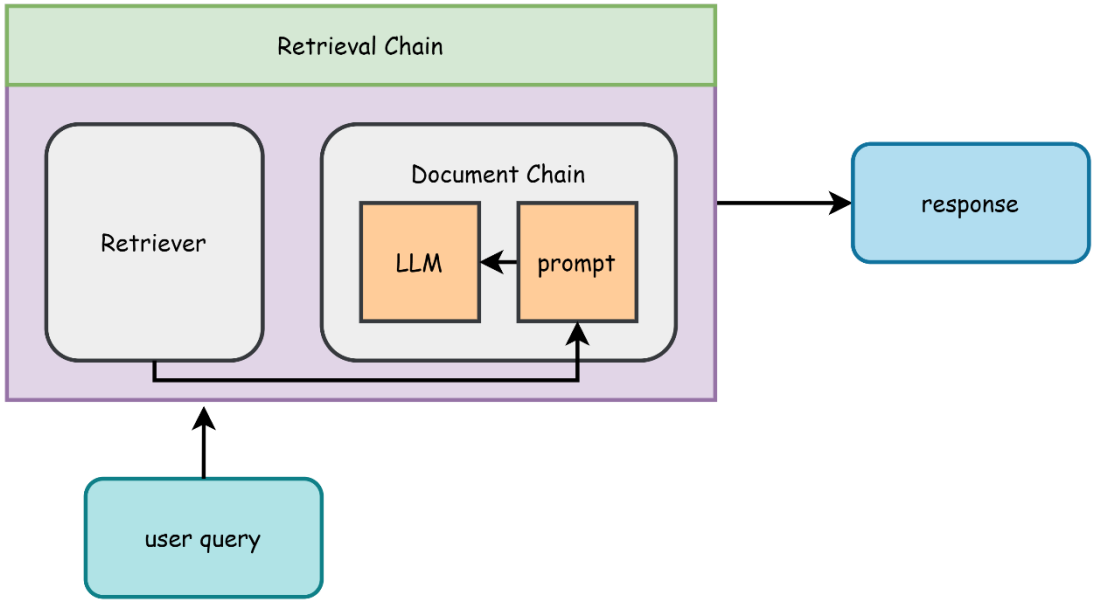

Fig 8. Working of the Retrieval Chain

Code snippet:

```
retrieval_chain = create_retrieval_chain(retriever, document_chain)
```

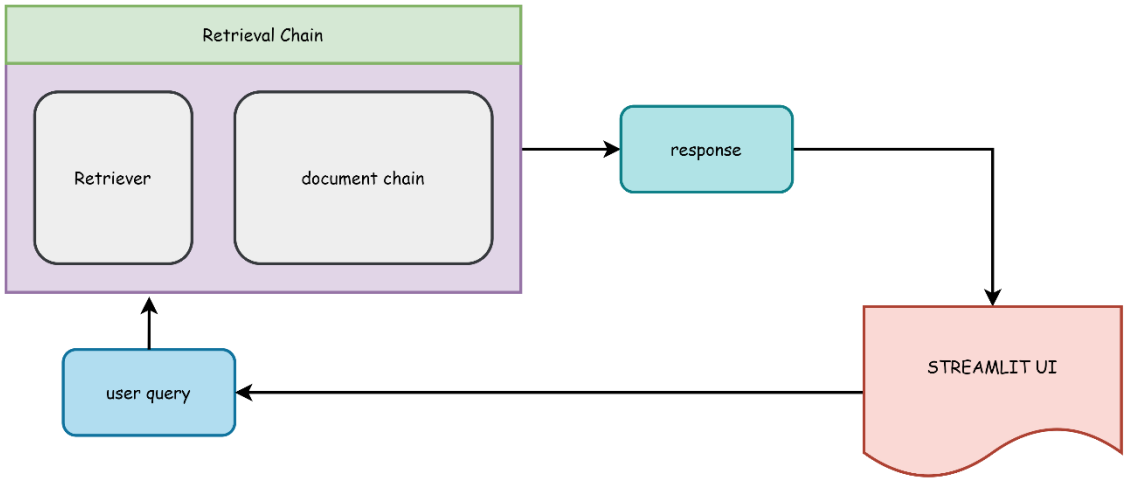

Fig 9. working of the Retrieval Chain with streamlit UI

8. Architecture

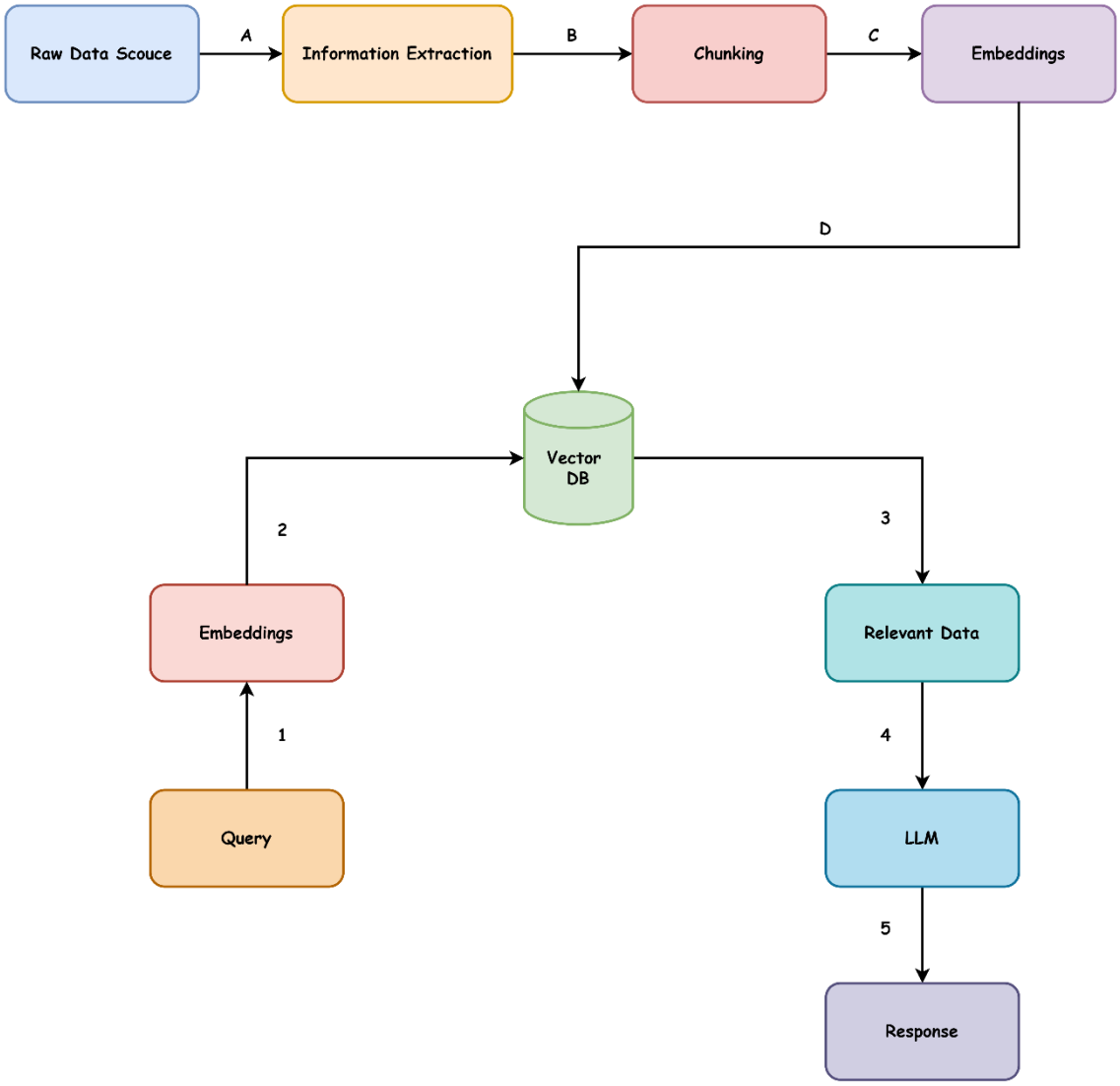

Fig 10. Abstract Architecture of the RAG

9. Limitations

Scalability and Performance: The project relies heavily on embedding models and FAISS for vector storage and retrieval, which can become resource-intensive as the size of the data increases. Large-scale datasets may lead to increased latency in query processing and retrieval times. Additionally, the deployment of the Llama3-70B-8192 model on GROQ Cloud may have constraints related to computational resources and throughput, potentially impacting performance during peak usage.

Model and API Dependency: The project's reliance on the Llama3-70B-8192 model and the GROQ Cloud API key introduces potential vulnerabilities related to external dependencies. Any disruptions in service availability, changes in API access policies, or limitations in the cloud provider's infrastructure could impact the system's functionality and reliability. This dependency on third-party services may also introduce challenges in maintaining consistent performance and ensuring long-term sustainability of the project.

Contextual Answer Limitation: The Retrieval-Augmented Generation (RAG) approach used in the project is designed to provide answers based on the context retrieved from semantically similar chunks. However, if the relevant information is spread across multiple chunks, the efficiency of the model decreases. The system may struggle to generate accurate responses when the necessary context is distributed, leading to incomplete or less precise answers. This limitation highlights the challenge of ensuring comprehensive context coverage, especially when dealing with fragmented or dispersed information within large documents.

10. How to Run the Application

Setup:

- **Install python version 3.11 from the official python website.** https://www.python.org/downloads/
- **Create an account on groq cloud and get a Groq API Key** https://console.groq.com/keys
- **Create a google account and get a Google API Key** https://aistudio.google.com/app/apikey

Run:

Steps to run the Application on Windows 10 or 11

- Download a Zip file or clone the Project using

```
https://github.com/k-arthik-r/ai_powered_log_parsing_tool.git
```

- UnZip the file or check the cloned Project for all the files.
- Navigate to the root Directory of the project.
- Create a python virtual environment.

```
python -m venv venv
```

- Activate the python environment. (for windows cmd)

```
.\venv\Scripts\activate.bat
```

or for power shell

```
.\venv\Scripts\Activate.ps1
```

- Install all the required libraries

```
pip install -r requirements.txt
```

- paste your google and groq API keys inside .env file in the root directory

```
GOOGLE_API_KEY = <paste your google api key here>
GROQ_API_KEY = <paste your groq api key here>
```

- run your application using,

```
streamlit run app.py
```

11. Results

Upon successfully following the outlined steps, the Streamlit application will be live on the local host, allowing users to interact with the application seamlessly. Users can input their queries directly into the interface to obtain detailed responses generated by the integrated LLM.

For example, as depicted in the figure below, a user query such as "At what speed is the sched_clock running?" results in the application generating a comprehensive answer, which is displayed in the interface. This user-friendly setup ensures that complex log analysis and query handling are accessible and intuitive, providing users with valuable insights and streamlined interactions.

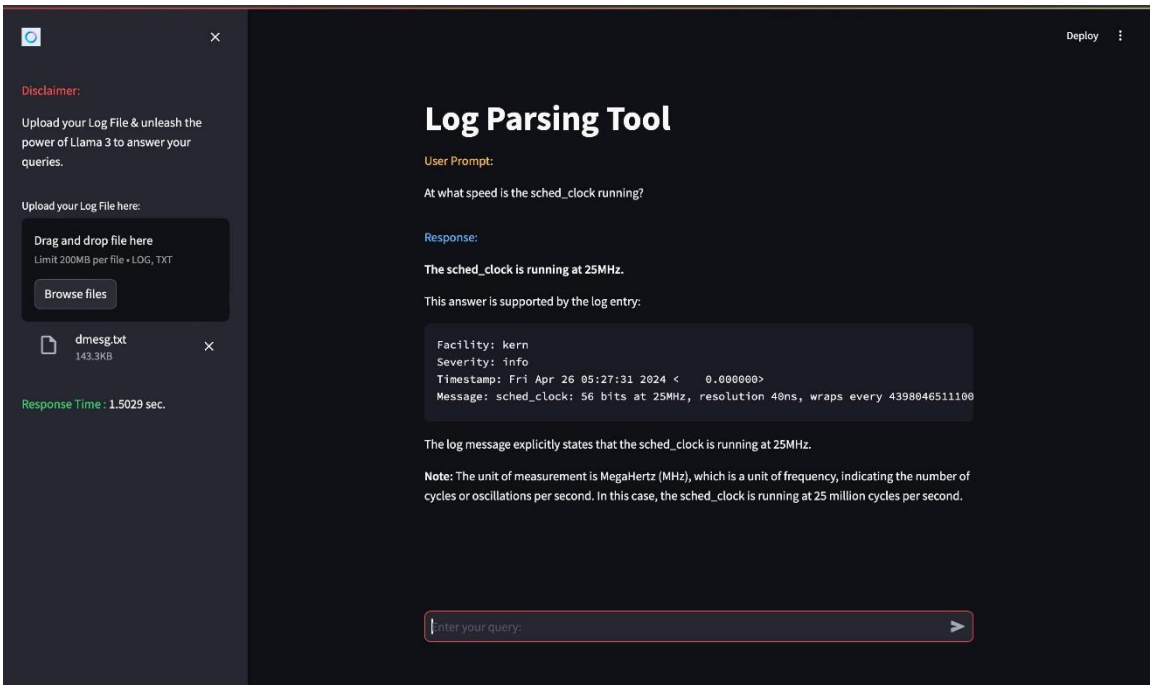

Fig 11. Snapshot of the Application Results

12. Conclusion

In conclusion, this project leverages advanced natural language processing techniques and modern machine learning models to efficiently analyze and process log data. By converting logs into structured formats and utilizing embedding models for vectorization, the system facilitates effective retrieval and contextual understanding of log information. The integration of a powerful LLM, coupled with a responsive Streamlit UI, enables interactive querying and insightful analysis. However, the project faces challenges related to scalability and external dependencies, which must be addressed to ensure robust and sustainable performance. Overall, this project represents a significant step forward in automating and enhancing log analysis through cutting-edge AI technologies.