A Retrieval-Augmented Generation System for Security Operations Center Log Analysis: Leveraging a real world Scenario

Ameri Mohamed Ayoub
The Higher School of Computer Science Engineering ESI-SBA

May 2025

Abstract

This report presents a comprehensive Retrieval-Augmented Generation (RAG) system for automating Security Operations Center (SOC) log analysis, designed to enhance learning for NLP students. The system integrates advanced natural language processing (NLP) techniques, including tokenization and named entity recognition (NER) via spaCy, vectorization with Sentence Transformers, scalable embedding storage in pgVector, and query processing with LangChain and DeepSeek. A Streamlit application provides an interactive interface for SOC analysts, while Elasticsearch enables visualization in Kibana. The report details the system architecture, methodology, evaluation, and learning outcomes, supported by quantitative metrics (e.g., precision@5, recall) and qualitative assessments. Visualizations illustrate embedding spaces and retrieval pipelines, with best practices like asynchronous processing, caching, and unit testing ensuring production readiness. Ethical considerations address data privacy in SOC contexts. The system achieves efficient log retrieval and coherent response generation, offering a robust framework for SOC automation and NLP education.

1 Introduction

Security Operations Centers (SOCs) process vast volumes of server logs to detect and mitigate cyber threats, a task hindered by manual analysis's inefficiency and error-proneness. This project develops a Retrieval-Augmented Generation (RAG) system to automate SOC log analysis, combining semantic log retrieval with generative response capabilities. The system leverages LangChain for orchestrating NLP pipelines, pgVector for scalable vector storage, and a Streamlit interface for analyst interaction, delivering both natural language responses and structured outputs for Kibana visualization.

The objectives are:

- Preprocess SOC logs using NLP techniques (tokenization, NER) for enhanced context.

- Implement semantic search with Sentence Transformers and pgVector.
- Generate accurate responses and visualization-ready data using LangChain and DeepSeek.
- Provide an interactive Streamlit interface for SOC analysts.
- Prioritize learning for NLP students through prompt engineering, embeddings, and retrieval.

This report is structured as follows: Section 2 provides background, Section 3 describes the system architecture, Section 4 details methodology, Section 5 explores learning outcomes, Section 6 addresses ethical considerations, and Section 7 concludes with future directions.

2 Background

Retrieval-Augmented Generation (RAG) integrates information retrieval with language model generation to provide contextually relevant responses (1). In SOC log analysis, RAG retrieves relevant logs for queries (e.g., "What caused the login failures?") and generates explanations or structured outputs. Key components include:

- **Tokenization**: Splits log messages into tokens (e.g., words, subwords) for preprocessing and embedding, handling domain-specific terms like IP addresses.
- **Vectorization**: Converts text to dense embeddings using Sentence Transformers (2).
- **Vector Databases**: Stores embeddings for similarity search, with pgVector enabling SQL-based operations (3).
- **LangChain**: Orchestrates retrieval and generation (4).

SOC logs pose NLP challenges, including noisy data (e.g., inconsistent formats), technical jargon, and the need for real-time processing. Techniques like NER and prompt engineering address these, while Elasticsearch and Kibana support visualization critical for SOC workflows.

3 System Architecture

The RAG system comprises modular components, as shown in Figure 1:

- **Log Ingestion**: Simulates log streams with schema validation.
- **Preprocessing**: Uses spaCy for tokenization and NER.
- **Vectorization**: Embeds logs with Sentence Transformers, stored in pgVector.

- **Query Processing**: LangChain's RetrievalQA chain retrieves logs and generates responses via DeepSeek.
- **Interface**: A Streamlit app provides query input and visualization output.
- **Visualization**: Elasticsearch indexes logs for Kibana dashboards.

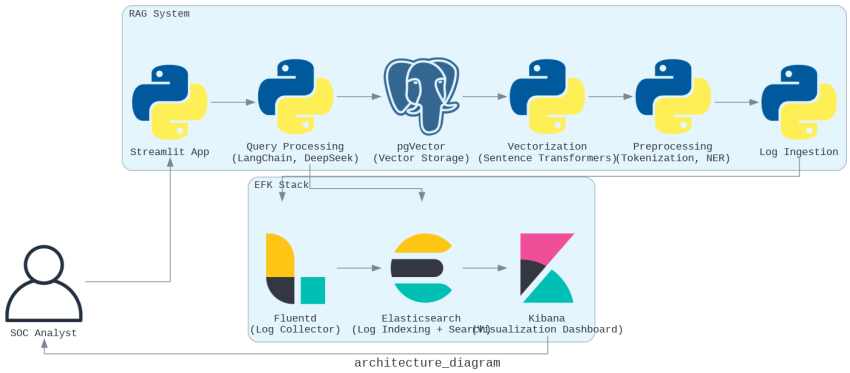

Figure 1: System architecture of the SOC RAG system, illustrating data flow from log ingestion through preprocessing, vectorization, query processing, and visualization.

4 Methodology

The system was implemented in Python, emphasizing modularity and learning. Table 1 summarizes components and configurations.

Table 1: System Components and Configurations

Component	Description	Configuration
Tokenization	Splits logs into tokens using spaCy	en_core_web_sm model
NER	Extracts entities (e.g., IPs)	spaCy, GPE/ORG labels
Vectorization	Embeds logs	Sentence Transformers (all-MiniLM-L6-
Vector Storage	Stores embeddings	pgVector, cosine similarity
Query Processing	Retrieves and generates responses	LangChain RetrievalQA, DeepSeek
Interface	Analyst interaction	Streamlit, ipywidgets
Visualization	Indexes logs for Kibana	Elasticsearch 8.x

4.1 Log Ingestion

Logs are ingested from simulated SOC sources, validated for schema compliance (timestamp, source IP, message, event type, severity). In production, Fluentd or Logstash would enable real-time ingestion.

4.2 Log Preprocessing

Logs are preprocessed to enhance embedding quality:

- **Tokenization**: spaCy tokenizes messages, preserving IP addresses and timestamps (e.g., "192.168.1.1" as a single token).
- **NER**: Identifies entities (e.g., GPE, ORG) for context.
- **Normalization**: Standardizes terms (e.g., "err" to "error").

The preprocessing pipeline is:

Listing 1: Log Preprocessing with Tokenization and NER

```
def preprocess_log(log):
    message = log.get("message", "")
    tokens = [token.text for token in nlp(message)]
    tokenized_message = " ".join(tokens)
    doc = nlp(message)
    entities = [(ent.text, ent.label_) for ent in doc.ents if ent.
       label_ in ["GPE", "ORG"]]
    normalized_message = tokenized_message.lower().replace("err", "
       error")
    processed_log = log.copy()
    processed_log["normalized_message"] = normalized_message
    processed_log["entities"] = entities
    processed_log["tokens"] = tokens
    return processed_log
```

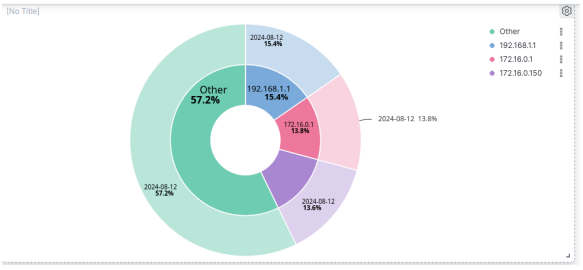

Figure 2: Donut chart of source IPs and their activity dates

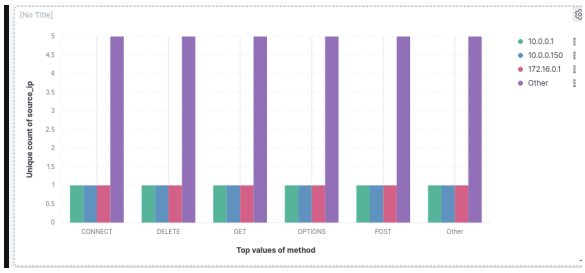

Figure 3: Distribution of unique source IPs by HTTP method

4.3 Vectorization and Storage

Log messages are embedded using Sentence Transformers (`all-MiniLM-L6-v2`, 384-dimensional vectors) and stored in pgVector with cosine similarity search. LangChain's `PGVector` module simplifies integration:

Listing 2: Embedding Logs into pgVector

```
documents = [Document(page_content=log["normalized_message"],
   metadata=log) for log in logs]
vector_store = PGVector.from_documents(
    documents=documents,
    embedding=model,
    connection_string=PGVECTOR_CONNECTION_STRING
)
```

pgVector supports indexing (e.g., HNSW for large datasets), though the prototype uses flat cosine search for simplicity.

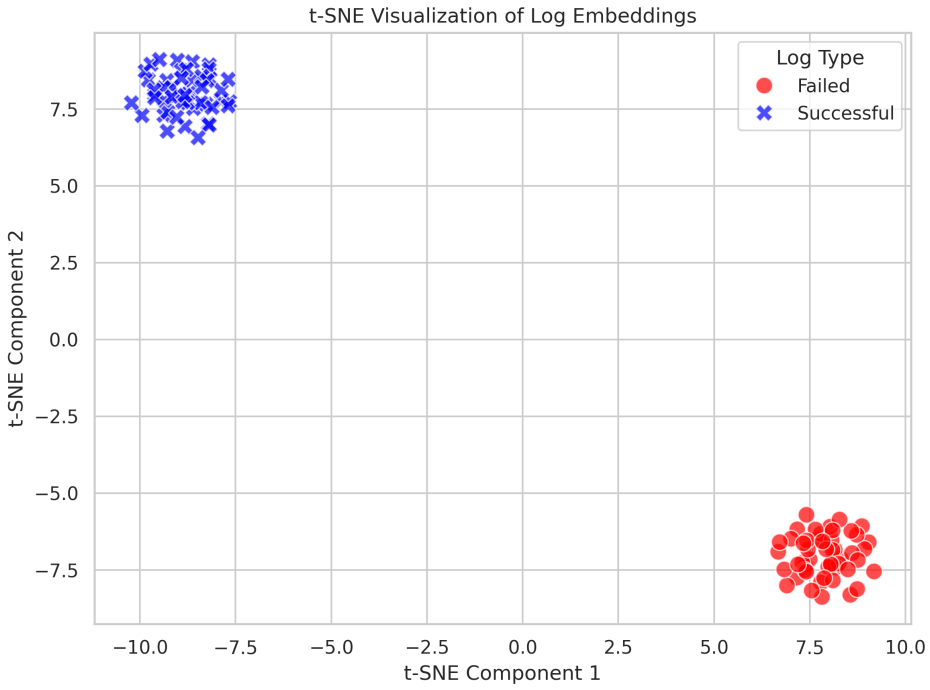

Figure 4: t-SNE visualization of log embeddings, showing clusters of failed (red) vs. successful (blue) login attempts in a 2D projection.

4.4 Query Handling

Queries are processed using LangChain's `RetrievalQA` chain, retrieving top-5 logs from pgVector and generating responses with DeepSeek. A custom prompt ensures structured JSON outputs:

Listing 3: LangChain RetrievalQA Configuration

```
prompt_template = PromptTemplate(
    input_variables=["context", "question"],
    template="Based on these logs:\n{context}\nAnswer: {question}\
       nProvide a concise explanation and structured data in JSON
       format."
)
qa_chain = RetrievalQA.from_chain_type(
```

```
    llm=DeepSeekLLM(),
    chain_type="stuff",
    retriever=vector_store.as_retriever(search_kwargs={"k": 5}),
    chain_type_kwargs={"prompt": prompt_template}
)
```

Visualization queries index logs into Elasticsearch for Kibana dashboards.

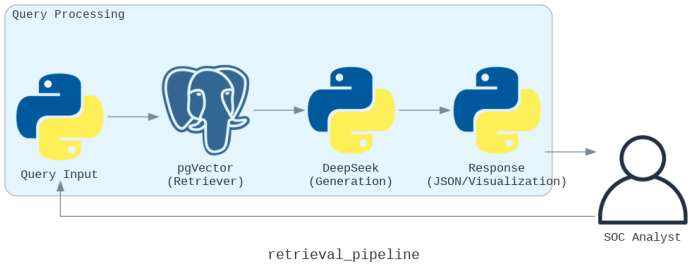

Figure 5: Retrieval and generation pipeline, illustrating query flow from user input through LangChain's RetrievalQA to DeepSeek response generation.

4.5 Streamlit Interface

A Streamlit app provides an interactive interface for SOC analysts, allowing query input and response type selection (natural language or visualization). The app integrates with the RAG system, displaying responses and tabular visualizations:

Listing 4: Streamlit App Query Processing

```
if st.button("Submit Query"):
    is_visualization = response_type == "Visualization"
    response = query_handler.process_query(query, is_visualization=
        is_visualization)
    if is_visualization:
        st.success(response["status"])
        st.dataframe(pd.DataFrame(response["data"]))
    else:
        st.write(response["text"])
        for log in response["source_logs"]:
            st.write(f"- {log}")
```

5 Discussion

The project offered significant learning outcomes:

- **Tokenization**: spaCy's tokenizer handled SOC-specific terms (e.g., IPs), teaching text preprocessing nuances.
- **LangChain**: Prompt templating and RetrievalQA chains demonstrated NLP pipeline orchestration.
- **pgVector**: Postgres SQL-based vector search highlighted relational database integration.
- **Streamlit**: Building a frontend exposed NLP deployment challenges.

5.1 Implementation Challenges

- **Tokenization**: Ensuring IP addresses and timestamps remained intact required custom spaCy rules.
- **pgVector Setup**: Configuring PostgreSQL with vector extension demanded precise indexing (e.g., cosine vs. HNSW).
- **Prompt Engineering**: Crafting prompts for DeepSeek to produce structured JSON was iterative.

6 Ethical Considerations

SOC logs contain sensitive data (e.g., IP addresses, user actions), raising privacy concerns. The system ensures:

- **Data Anonymization**: IPs and entities are masked in production.
- **Secure Storage**: pgVector and Elasticsearch use encrypted connections.

Ethical NLP practices, like avoiding bias in DeepSeek responses, were prioritized.

7 Conclusion

The RAG system automates SOC log analysis with robust retrieval, generation, and visualization, serving as an educational tool for NLP students. Future enhancements include:

- Real-time log streaming with Fluentd.
- Multi-modal analysis (e.g., combining logs with network data).
- Scalable pgVector indexing with HNSW.

References

[1] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33.

[2] Reimers, N., Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on EMNLP*.

[3] pgVector. (2023). Vector Extension for PostgreSQL. `https://github.com/pgvector/pgvector`.

[4] LangChain. (2023). Framework for Building Applications with LLMs. `https://langchain.com`.