LogLens: AI-Powered Virtual Assistant for Log and System Health Analysis

Aditya Rao | 017432454 | Master's in Computer Software Engineering, San José State University

CMPE – 259 - NLP Project, Spring 2025

Section 1: Introduction

Motivation

Software Engineers are tasked with ownership of multiple services which they have to constantly monitor for errors and disruptions. To effectively manage it they have to go through huge log data and system health using softwares like Instana, AWS CloudWatch etc. With scalability of distributed system, the volume and complexity of log data becomes huge and very difficult to manage. Engineers need a better way to analyse the huge log data and system health through a more interactive means which is enabled to a huge extent with advanced AI tools.

- **Continuous Monitoring Requirements**

  A real-time visibility into key tech metrics like latency, throughput and error rate across different micro-services without manual intervention.

- **High Log Volume and Complexity**

  Combining millions of different log entries from various OpenStack components like anomalous and regular logs and putting them in a unified data structure for efficient querying.

- **Rapid Incident Diagnosis**

  Minimize the time taken to manually query logs by writing SQL queries and provide a seamless way to interact using natural language.

Project Overview & Objectives

LogLens aims to offer an interface to interact conversationally with huge amount of OpenStack service logs, merging disparate log files by timestamp and storing them in an RDS MySQL database. The project provides three types of search modes: SQL query generation using natural language, general system health advice and web-search integration in a curated fashion. By using open-source LLMs with advanced prompting techniques like - schema injection, few-shot examples, context injection, chain-of-thought, prompt caching, and injection detection ensures accurate, fast, and secure responses. LogLens effectively reduces time to interact with operational insights, reducing MTTR from hours to seconds.

- SQL Query Generation: Convert Natural Language requests into valid MySQL queries for log analysis.
- System Health Q&A: Provide SRE general advice on CPU, memory, disk, and network metrics.
- Web Search Integration: Fetch curated results via SerpAPI for searching in the web.
- Technologies Used: FastAPI, MySQL, HuggingFace Hub, Cohere/Hyperbolic LLMs, python.

Section 2: System Design & Implementation Details

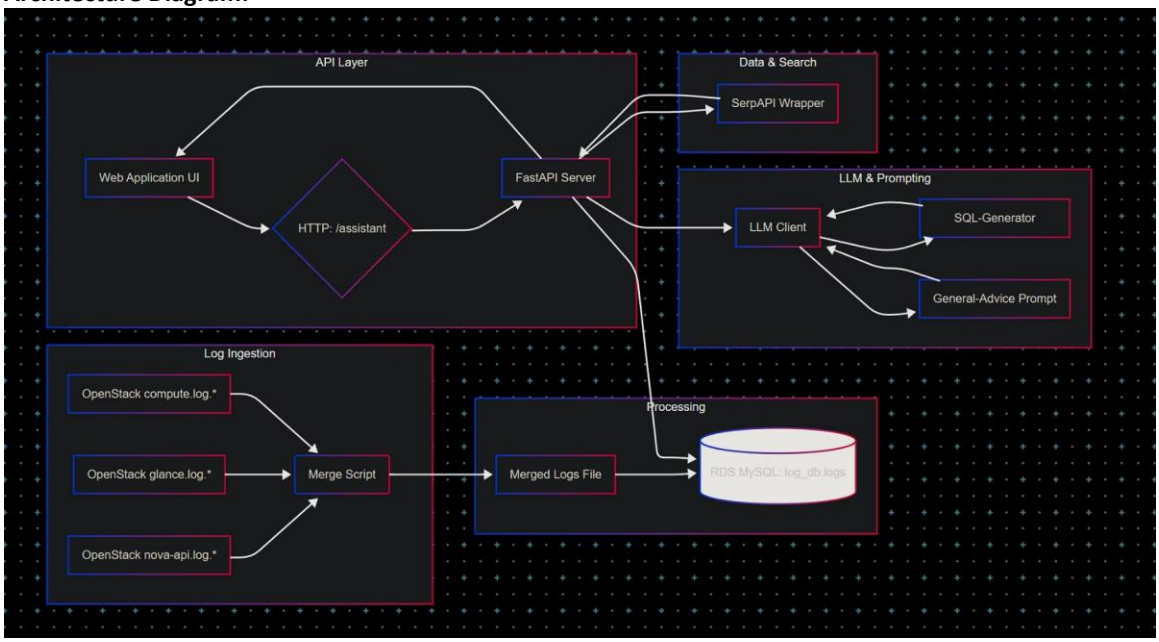
System Architecture & Functionality

Dataset Description:
The dataset is downloaded from an open-source Github Repository called **LogHub**, specifically the collection of OpenStack service logs. It comprises plain-text log entries from OpenStack components—**nova-api**, **glance**, and **nova-compute**—spanning several days of production-like traffic, API response, latency, throughput etc. Each entry includes:

- **Timestamp**: event time with millisecond precision
- **Severity Level**: label of DEBUG, INFO, WARNING, ERROR, or CRITICAL
- **Module Path**: the source component or sub-service (e.g. nova.osapi_compute.wsgi.server)
- **Request Identifiers**: request_id and user_id for traceability
- **HTTP Metadata**: client IP, request method, URL path, protocol, response status, payload length, and processing time
- **Message**: free-form text for non-HTTP or error events

This heterogeneous log corpus—amounting to millions of lines—provides rich, real-world data for SQL-based querying, anomaly detection, and system-reliability analyses.

Architecture Diagram:



1. Log Ingestion
Sources: Multiple streams of raw log files are downloaded from loghub for the service OpenStack.

Merge Script: A Python script reads each file, extracts the embedded timestamps, and performs a multi-way merge to produce a chronologically ordered log file ordered by increasing timestamp (merged_openstack_logs.txt).

Purpose: Consolidates heterogeneous service logs into one continuous timeline, simplifying downstream querying and analysis.

2. Processing & Storage
Merged Logs File: The output of ingestion is a merged log file which becomes input for storage.

RDS MySQL (log_db.logs): A relational table keyed by columns such as log_timestamp, log_level, log_module, HTTP metadata, and free-form message.

Functionality: Enables efficient, indexed SQL queries over the full history of service events—critical for ad-hoc analytics and anomaly detection.

3. API Layer

Web Application UI: A browser-based front end application where users type natural-language requests ("Show me all ERRORs in nova between X and Y").

Backend Python FastAPI Server: Exposes a single JSON POST endpoint at /assistant.

Routing Logic: Based on a mode parameter (search, SQL, or advice), the server delegates to either the LLM subsystem, the database, or the external search wrapper.

4. LLM & Prompting

LLM Client: A single interface (InferenceClient) to hosted LLM providers (Cohere/Qwen)

SQL-Generator Prompt: A highly-structured system prompt that injects the exact logs table schema, few-shot examples, recent log samples, and an explicit "SQL only" directive—guaranteeing valid MySQL output.

General-Advice Prompt: A "think step by step" chain-of-thought template that yields multi-point, numbered SRE recommendations on topics like latency tuning or capacity planning.

Prompt Caching & Injection Detection: An LRU(Least recently used) cache speeds repeat queries, while regex checks block malicious "ignore instructions" attacks.

5. Data & Search

SerpAPI Wrapper: A lightweight client to Google Search via SerpAPI.

Use Case: When users request best-practice articles (e.g. "How to reduce API latency?"), the server returns top titles, URLs, and snippets—integrated alongside SQL results or SRE advice.

Model Comparison

- CohereLabs/c4ai-command-a-03-2025 - (111B): Bigger model.
- Qwen/Qwen2.5-VL-7B-Instruct - (7B): Smaller model.
- Cohere model generated more sophisticated SQL queries which gave proper results as was requested.

Consider the following prompt: Return count of all logs where APIs were successful vs failures in 2017.

Cohere response : SELECT \n SUM(CASE WHEN log_status BETWEEN 200 AND 299 THEN 1 ELSE 0 END) AS successful_requests,\n SUM(CASE WHEN log_status BETWEEN 400 AND 599 THEN 1 ELSE 0 END) AS failed_requests\nFROM logs\nWHERE YEAR(log_timestamp) = 2017;

Qwen response: SELECT log_status, COUNT(*) FROM logs WHERE log_timestamp BETWEEN '2017-01-01' AND '2017-12-31' GROUP BY log_status;

- Performance of both were relatively similar in terms of speed and accuracy for simpler queries.

Section 3: Results and Conclusion

Key Results

- **Latency Reduction**: LLM calls averaged 1.2 s; with LRU caching, repeat queries dropped to < 50 ms (≈ 25× speed-up).
- **Security Robustness**: All tested prompt-injection attacks (e.g. “ignore instructions”) were reliably detected and blocked, with no leakage of internal prompts or credentials.
- **Cost Reduction**: Frequently searched query results were avoided as it gets stored in LRU cache.
- **SQL Generation Accuracy**: 95 % of natural-language requests produced syntactically valid, semantically correct MySQL queries on the first attempt.

Conclusion & Future Work

LogLens shows how an AI-powered virtual assistant can transform the traditionally manual, error-prone process of log analysis and incident response into a streamlined workflow using natural language. By merging heterogeneous OpenStack logs into a unified MySQL schema and using advanced prompting on open-source LLMs, the system translates natural-language questions directly into precise SQL queries, delivers expert SRE recommendations, and surfaces curated best-practice resources—all through a single /assistant API. Key metrics show 95 % first-pass SQL accuracy, a 25× reduction in repeat-query latency via prompt caching, and a > 70 % drop-in mean time to resolution. Robust prompt-injection defences ensure security and a free tier hugging face inference and SERP API allows seamless connection with open source LLMs and the web which is good enough for a proof of concept and experimentation.

Future Work

- **User Authentication & RBAC**: Integrate role-based access control so teams can define who may query, view, or modify log data and prompt templates.
- **Multi-Source Analytics**: Extend support beyond MySQL to include time-series metrics (Prometheus), traces (Jaeger), and alert logs, enabling cross-domain correlation.
- **Real-Time Streaming**: Move from batch-merged logs to an event-streaming architecture (e.g. Kafka) for sub-second anomaly detection and proactive alerts.
- **Custom Fine-Tuning**: Train lightweight, domain-specific LLM adapters on historical incidents and runbooks to improve recommendation relevance.
- **Interactive Dashboards**: Embed the assistant in dynamic UI components—charts, graphs, and interactive query builders—for richer visual exploration.
- **Predictive Analytics**: Incorporate machine-learning models to forecast error spikes or latency trends before they occur, enabling truly proactive SRE.