BCSE497J Project-I

NATURAL LANGUAGE INTERFACE FOR LOG DATA ANALYSIS

22BCE0219	**ANSHUMAN ARYAN**
22BCE2576	**LAUKIK WADHWA**

Under the Supervision of

Dr Durgesh Kumar

Assistant Professor Sr. Grade 1

School of Computer Science and Engineering (SCOPE)

B.Tech.

in

Computer Science and Engineering

School of Computer Science and Engineering

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

September 2025

ABSTRACT

Extracting insights from log data is essential for maintaining and improving software systems. However, doing so typically requires familiarity with complex query languages, which slows down tasks like debugging and incident management. This project proposes a more intuitive, conversational interface that simplifies log analysis for technical users of all experience levels. We are building an agent-driven system that uses a large language model to understand plain English questions. Our approach involves incrementally expanding the context from log data until enough relevant information is gathered to answer a user's query. This method avoids the need to process the entire log file, making the system scalable and efficient for both simple and complex questions

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background

Modern software systems are known to generate vast quantities of log data. These logs play a critical role in the maintenance and improvement of software, as they are essential for tracking performance issues, identifying bugs, and detecting potential security threats. The ability to extract meaningful insights from this data is fundamental to system reliability and security.

1.2 Motivations

Despite the importance of logs, analyzing them is often a difficult and time-consuming task. Traditionally, this process requires engineers to write detailed queries using domain-specific languages like Splunk Processing Language or Elasticsearch Query DSL. This reliance on complex query languages can slow down critical tasks such as debugging and incident management. This presents a significant challenge for engineers who may lack the necessary expertise, particularly when they are under pressure during incidents. The goal of this project is to address this inefficiency by proposing a solution that allows users to ask questions in everyday language and receive relevant answers directly from the logs, eliminating the need for technical queries.

1.3 Scope of the Project

This project proposes the development of an intuitive, conversational interface designed to simplify log analysis for technical users of all experience levels. We are building an agent-driven system that leverages a large language model to understand questions posed in plain English. The core of our approach is a method that incrementally expands the context from log data, starting with a small portion and gathering more information only as needed to answer a user's query. This technique avoids the need to process an entire log file at once, ensuring the system is both scalable and efficient for simple and complex questions. The ultimate aim is to make log analysis as straightforward as having a conversation with a colleague.

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

The concept of using natural language to query log data is a developing field. A significant advancement was made by He and colleagues in 2022 with the introduction of the LogQA dataset. This dataset contains real-world log entries and complex questions that necessitate advanced reasoning across multiple entries. Researchers have generally taken two main approaches to this problem.

The first are parsing-based methods, which utilize regular expressions or machine-learned templates to give structure to log data. A key weakness of these methods is their fragility; if a log line does not conform to a known format, it might be missed completely. The second, more recent strategy involves using large language models to translate natural language directly into executable queries. An example of this is the NL to Splunk system, which can produce queries with minimal training. However, these systems often face difficulties with raw or diverse log formats and typically support only specific platforms.

2.2 Gaps Identified

Despite recent progress, several challenges in the field remain unresolved. A major issue is that many existing systems rely on rigid log parsing methods, which are prone to failure when they encounter new or unusual log formats. Furthermore, errors that occur in the initial stages of analysis can propagate through all subsequent steps, leading to inaccurate final answers. Another significant limitation is that most available solutions are tied to specific platforms such as Splunk or ELK, which restricts their application in varied technology environments. This highlights the need for a more flexible, platform-independent solution capable of understanding any type of log file.

2.3 Objectives

The primary objectives of this project are as follows:

- To design and develop an agent-based, self-correcting, and platform-independent natural language interface for analyzing log data.
- To implement a "Context Resolver" component that, instead of processing an entire log file at once, starts with a small portion of the data and gradually expands its view until sufficient context is gathered to answer a query.
- To utilize a "Language Model Agent" that interprets both log content and user questions, allowing it to request additional data if the current context is insufficient and avoiding the need for fixed parsing rules.
- To build a system that converts plain language questions into accurate, executable responses, which may take the form of custom Python scripts for raw files or structured queries for platforms like Splunk.

2.4 Problem Statement

Modern software systems generate enormous volumes of log data. These logs are crucial for tracking performance issues, identifying bugs, and detecting security threats. However, the process of analyzing these logs is often difficult because it traditionally requires writing detailed queries in domain-specific languages like Splunk Processing Language or Elasticsearch Query DSL. This requirement poses a significant challenge for engineers who may not have the necessary expertise, especially when under pressure during critical incidents. Our project aims to solve this by creating a solution that allows users to simply ask questions in everyday language and receive relevant answers from the logs.

2.5 Project Plan

The project will be executed by developing an agent-based architecture designed to learn from feedback and improve its accuracy over time. The development will be centered around two key components: the Context Resolver and the Language Model Agent.

First, the Context Resolver will be engineered to manage data ingestion efficiently. It will be designed to start with a small portion of the beginning and end of the log data and then incrementally expand its view only until it collects enough context to address the user's question.

Second, the Language Model Agent will be implemented to interpret the log data and the user's query. A key feature of this agent will be its ability to request additional data if the initial context is insufficient, making the system highly adaptable to variations in log formats without relying on rigid parsing rules. The final system will be capable of generating executable responses, such as Python scripts or structured queries, based on the user's plain language question.

3. REQUIREMENT ANALYSIS

This section outlines the functional and non-functional requirements for the Natural Language Interface for Log Data Analysis. These requirements define the system's capabilities and operational characteristics.

3.1 Functional Requirements

- **Natural Language Query Input:** The system must accept user queries in plain, everyday English, eliminating the need for complex query languages.
- **Multi-Source Connectivity:** The system must be able to connect to various log sources, including cloud storage, Elasticsearch clusters, Splunk instances, databases, and monitoring tools.
- **Incremental Context Processing:** The system will not process entire log files at once. Instead, it must start with a small data sample and incrementally expand its context until enough information is gathered to answer the query.
- **LLM-Based Interpretation:** An agent-driven large language model will interpret both the user's question and the content of the log data to understand the query's intent.
- **Executable Response Generation:** The system must convert the user's question into an accurate, executable response, which could be a custom Python script for raw files or a structured query for platforms like Splunk.
- **Self-Improvement:** The system must incorporate a feedback loop, allowing it to learn and improve its accuracy over time.

3.2 Non-Functional Requirements

- **Efficiency:** The system must be highly efficient by avoiding the need to process entire log files for every query, making it scalable for both simple and complex questions.
- **Platform Independence:** The solution must be flexible and not tied to a specific platform or log format, making it applicable across varied technology environments.
- **Usability:** The interface should be simple and intuitive, featuring a basic search bar so it can be used by anyone on a technical team, including junior engineers and non-specialists.
- **Accuracy:** The system is designed to be self-correcting, with an iterative correction loop to ensure the final answers are accurate.
- **Speed:** The tool must significantly reduce the time needed to extract insights from logs, which is especially critical during outages or performance issues.

4. SYSTEM DESIGN

This section describes the architectural design of the proposed system, detailing its core components, workflow, and the technologies required for implementation.

4.1 Architecture and Workflow Model

The system is built on an **agent-based, self-correcting architecture**. This design allows the system to learn from feedback and improve its accuracy over time. The workflow is iterative and designed for efficiency.

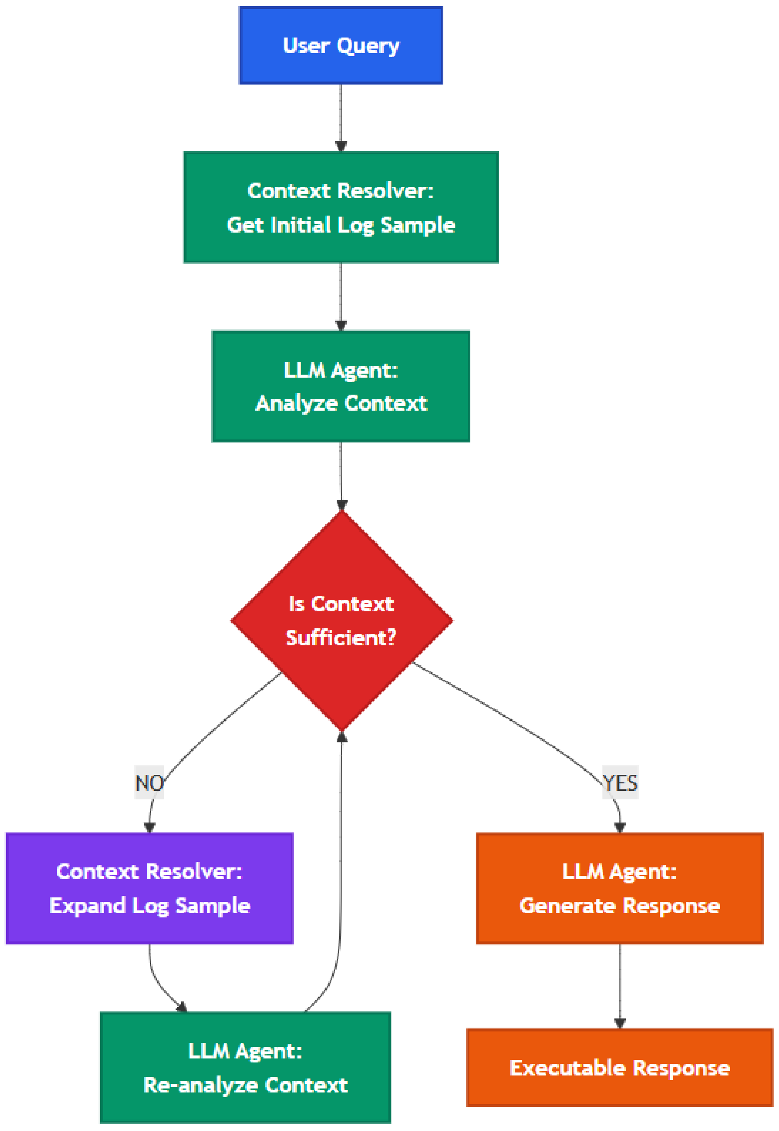
The user interaction and data processing flow is as follows:

1. **Query Submission**: A user asks a question in plain English through a simple interface.
2. **Initial Context Retrieval**: The **Context Resolver** module fetches a small, initial portion of the log data, such as the beginning and end of the file, rather than loading the entire log.
3. **Agent Analysis**: The **Language Model Agent** receives the user's question and the initial log context. It analyzes both to determine if the available information is sufficient to provide an accurate answer.
4. **Iterative Context Expansion**:

- If the context is **insufficient**, the agent requests additional data.
- The Context Resolver then expands its view, providing a larger portion of the log data to the agent.
- This loop continues until the agent has gathered enough relevant information to confidently answer the user’s query.

5. **Response Generation**: Once the context is sufficient, the agent formulates a final answer and generates an executable response, such as a Python script or a structured query (e.g., for Splunk).

   This iterative process is visualized below:



User Query
Context Resolver: Get Initial Log Sample
LLM Agent: Analyze Context
Is Context Sufficient?
NO
YES
Context Resolver: Expand Log Sample
LLM Agent: Re-analyze Context
LLM Agent: Generate Response
Executable Response

4.2 Module Design

The system comprises two primary modules:

- **Context Resolver**: This module's main purpose is to manage data ingestion efficiently. Instead of processing the entire log file at once, it starts with a small portion and gradually expands its view on demand. This strategy makes the system scalable and avoids the high computational cost of analyzing large files from the start.

- **Language Model Agent**: This is the core intelligence of the system. It interprets the user's natural language question and the content of the logs. A key feature is its ability to request more data when the context is insufficient, which makes the system adaptable to various log formats without needing rigid parsing rules. The agent operates within a self-correcting loop, allowing it to refine its understanding and improve answer accuracy over time.

4.3 Implementation Status (40%)

The project is currently in the implementation phase, with approximately 40% of the work completed.

- **Completed**:

  - Thorough literature review, gap analysis, and finalization of objectives.
  - Detailed requirement analysis and SRS documentation.
  - Design of the high-level system architecture and workflow.
  - Development of a prototype **Context Resolver** module capable of sampling log files and loading data incrementally.

- **In Progress**:

  - Integration with a pre-trained Large Language Model (LLM).
  - Development of the core logic for the **Language Model Agent** to evaluate context sufficiency.
  - Implementation of the iterative loop for context expansion.
  - Building the initial user interface for query input.

4.4 Hardware and Software Specifications

The following specifications are required for the development and deployment of the project:

- **Hardware**:
  - **Development Machine**: A computer with a multi-core processor (e.g., Intel Core i7/AMD Ryzen 7), 16 GB RAM, and a 512 GB SSD.
  - **Deployment/Training Server**: Cloud-based server with GPU support (e.g., NVIDIA A100 or V100) for hosting and fine-tuning the language model.
- **Software**:
  - **Programming Language**: Python 3.9+
  - **Core Libraries**:
    - LangChain/LlamaIndex: For orchestrating the LLM agent and data connections.

        - OpenAI/Hugging Face Transformers: For interacting with LLMs.
        - Pandas: For handling and manipulating log data.
        - Flask/FastAPI: For creating an API to serve the model.
    - **Language Model**: A pre-trained LLM such as GPT-4, Llama 3, or a fine-tuned open-source model.
    - **Version Control**: Git / GitHub.

5. PROJECT PLAN AND TIMELINE

This chapter outlines the project's execution plan, including a breakdown of the work into manageable tasks and a timeline for their completion.

5.1 Work Breakdown Structure (WBS)

1. **Phase 1: Research and Planning**

1.1. Conduct Literature Review and Identify Research Gaps

1.2. Finalize Project Objectives and Scope

1.3. Develop Software Requirement Specification (SRS)

1.4. Design High-Level System Architecture

2. **Phase 2: Core Development**

2.1. Set Up Development Environment (Python, Git, LLM API Access)

2.2. Implement the Context Resolver Module

2.3. Implement the Language Model Agent

2.4. Develop the Iterative Context Expansion Logic

3. **Phase 3: Integration and Testing**

3.1. Integrate the Context Resolver and Language Model Agent

3.2. Implement the Self-Correction and Feedback Loop

3.3. Perform Unit and Integration Testing

3.4. Refine Model and Agent Logic Based on Test Results

4. **Phase 4: Finalization**

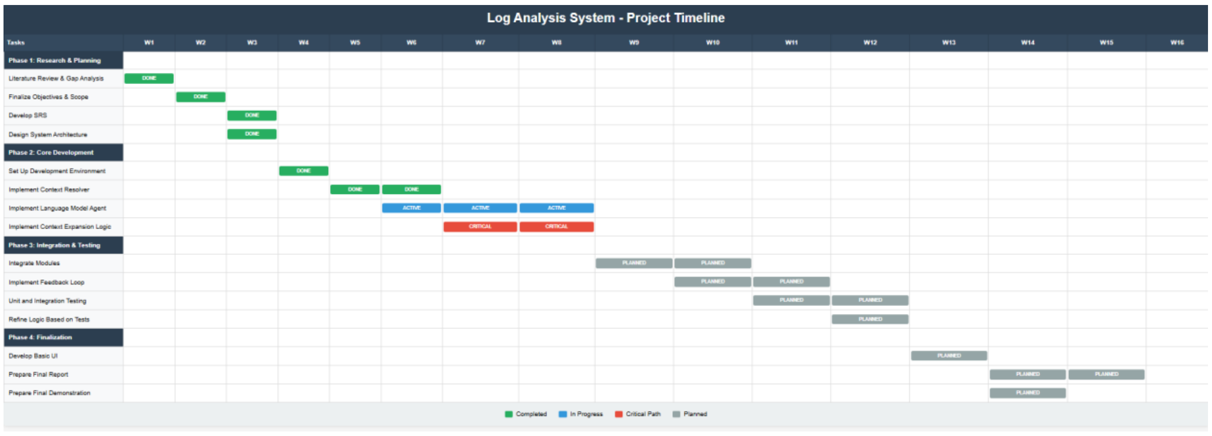
4.1. Develop a Basic User Interface (UI) for Demonstration

4.2. Prepare the Final Project Report

4.3. Prepare and Rehearse the Final Project Demonstration

5.2 Gantt Chart

The project is planned over a 14-week semester. The following table represents the timeline.



Log Analysis System - Project Timeline

Tasks	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16
Phase 1: Research & Planning																
Literature Review & Gap Analysis	DONE															
Finalize Objectives & Scope		DONE														
Develop SRS			DONE													
Design System Architecture			DONE													
Phase 2: Core Development																
Set Up Development Environment				DONE												
Implement Context Resolver					DONE	DONE										
Implement Language Model Agent						ACTIVE	ACTIVE	ACTIVE								
Implement Context Expansion Logic							CRITICAL	CRITICAL								
Phase 3: Integration & Testing																
Integrate Modules									PLANNED	PLANNED						
Implement Feedback Loop										PLANNED	PLANNED					
Unit and Integration Testing											PLANNED	PLANNED				
Refine Logic Based on Tests												PLANNED				
Phase 4: Finalization																
Develop Basic UI													PLANNED			
Prepare Final Report														PLANNED	PLANNED	
Prepare Final Demonstration														PLANNED		

🟩 Completed 🟦 In Progress 🟥 Critical Path ▨ Planned

6. REFERENCES

[1] He, P., et al. (2022). LogQA: A Question Answering Dataset on A Large Scale Log Dataset. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics.*

[2] Aryan, A., Wadhwa, L., Eshwar D, K., Sinha, A., & Kumar, D. (2025). AlphaPro at SemEval-2025 Task 8: A Code Generation Approach for Question-Answering over Tabular Data. In *Proceedings of the The 19th International Workshop on Semantic Evaluation (SemEval-2025)* (pp. 2358-2367). Association for Computational Linguistics.