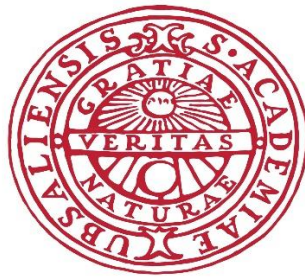


Uppsala University
Department of Informatics and Media

Optimizing LLMs for Microservices Logs Analysis through Prompt Engineering

Sahar Ahmed



UPPSALA
UNIVERSITET

Master's Degree Project
September 10th, 2025
Supervisor: Steve McKeever

Abstract

The increasing complexity of cloud-native and microservices-based architectures has made log analysis a critical task for ensuring system reliability and operational efficiency. Traditional rule-based approaches to log analysis are limited in scalability and adaptability, particularly within fast-paced environments of continuous integration and continuous deployment. In industrial settings such as Ericsson, microservices testing generates over 1,300 issue tickets each month, each linked to large volumes of distributed logs. Manually inspecting and diagnosing these logs is time-consuming and resource-intensive, creating a significant operational burden.

Recent advances in large language models have introduced promising opportunities for automating log interpretation, summarization, and failure diagnosis. This study investigates the impact of prompt engineering on the performance of large language models in summarizing microservices logs, with a specific focus on Ericsson's continuous integration and deployment pipeline and logs produced by the Java Common Auto Tester framework. Prompt engineering is defined here as the strategic design of input text to guide a language model toward producing accurate and contextually relevant outputs. Six prompting strategies were evaluated, including minimal instruction, role reframing, few-shot prompting, and chain-of-thought prompting, using real issue tickets and their associated log data.

To evaluate performance, the research combined automatic text comparison methods—such as recall-oriented gisting evaluation and semantic similarity scoring—with expert feedback from engineers working in continuous integration and deployment. The results showed that minimal instruction and role reframing prompts produced the clearest, most relevant, and most diagnostically useful summaries, while few-shot and chain-of-thought approaches were less effective due to verbosity and limited generalization.

This thesis contributes to ongoing research on the use of large language models in software operations by demonstrating that the design of prompts has a decisive influence on output quality. The findings have practical implications for enhancing automated log analysis in industrial environments and provide a foundation for future exploration of adaptive and hybrid prompting strategies.

Acknowledgements

I would like to express my deepest gratitude to my industrial supervisor at Ericsson, Pawan Sarika, for his invaluable support, guidance, and encouragement throughout this research. His insights and expertise played a vital role in shaping the direction and execution of this thesis. I am also sincerely thankful to the CI/CD team at Ericsson for their collaboration, and for generously providing the technical resources, domain knowledge, and access required to carry out this work. It has been a privilege to contribute to and learn from such a dedicated and knowledgeable team of professionals.

I would also like to extend my sincere appreciation to my academic supervisor, Steve McKeever, for his continuous mentorship, constructive feedback, and unwavering support throughout the course of this thesis. His guidance was instrumental in helping me maintain academic focus and rigor.

Finally, I am profoundly grateful to my family and friends for their constant support, encouragement, and patience. Their belief in me, especially during challenging times, made this journey possible.

Table of contents

1	Introduction	4
1.1	Background	4
1.2	Problem Formulation.....	5
1.3	Purpose and Research Question	6
1.4	Delimitation.....	6
1.5	Key Concepts	7
1.6	Thesis Outline	9
2.	Research context	11
2.1	Collaboration with Ericsson	11
2.3	Microservices at Ericsson.....	11
2.3	Large Language Model Log Analysis Tool	12
	How Microservices Log Analysis Tool Works.....	12
2.4	Research Focus: Prompting Engineering for LLM based log summarization	14
3	Existing Research.....	16
3.1	Microservices Log Analysis Theories and Concepts:	16
3.2	Natural Language Processing (NLP) and LLM Theories:	16
3.3	Automatic Microservices Failure testing	17
3.4	Prompt Engineering and Its Influence on the Effectiveness of LLM Responses.....	18
3.5	Automatic Evaluation Metrics for Large Language Models.....	19
4.	Method	20
4.1	Research strategy.....	20
4.2	Research Design.....	20
4.2.1	Design Science	21
4.3	Data Collection.....	26
4.4	Analysis Methods.....	27
4.4.1	Large Language Models Automatic Evaluation Metrics	28
4.4.2	Human-Centered Evaluation Using the Bradley-Terry Model and Elo rating.....	30
4.5	Research Object and Experimental Setup	32
4.5.1	Software Environment.....	33
4.6	Ethical Considerations.....	33
5.	Results	35
5.1	Experimental Datasets and Setup.....	35
5.1.1	Dataset.....	35
5.1.2	Data Processing and Evaluation Metrics.....	39

5.1.3 Prompt engineering configurations	39
5.2 Performance Evaluation Results	44
5.2.1 Expert user Assessment Results	44
5.2.2 Automatic Evaluation Results	45
6. Discussion	49
6.1 Research questions	49
6.3 Knowledge contribution	49
6.4 Practical Implications	50
6.4 Limitations	50
7. Conclusion and Future Research	51
7.1 Future work	52
References:	53
Appendix A - Practitioner Survey Protocol (Qualitative Evaluation)	57
Appendix B: Illustrative Example of Microservices Log Data	63

Table of Figures

Figure 1: Microservices Log Analyzer Workflow	13
Figure 2: Workflow of the Microservices Log Analysis Tool Integrated with JCAT and LLM API.	14
Figure 3 BERT SCORE Calculation	29
Figure 4 Precision equation	30
Figure 5 Recall equation	30
Figure 6 F1 equation	30
Figure 7 Bradely-Terry Model equation	31
Figure 8 LOG Data EXAMPLE for Rank 5 ticket	36
Figure 9 JIRA ticket details For Rank 5 Example	37
Figure 10 Data EXAMPLE for Rank 5 ticket	38
Figure 11 JIRS TICKET Rank 1 EXAMPLE	38
Figure 12 The monthly average Jira Tickets	39
Figure 13 Bradley Model survey Results	45
Figure 14 ELO Rating Survey Results	45
Figure 15 Rouge Metric Results	46
Figure 16 BERTSCORE Metric Results	48

Table of Tables

Table 1 ROUGE Metric Results	47
Table 2 BERT SCORE Results	48

1 Introduction

This chapter presents the motivation and scope of the study, focusing on the challenges of log analysis in microservices-based Continuous Integration and Continuous Deployment (CI/CD) environments. It introduces the use of Large Language Models (LLMs) and prompt engineering as a solution for improving log summarization. The chapter also outlines the research problem, questions, limitations, and the overall structure of the thesis.

1.1 Background

Modern software engineering has increasingly adopted cloud-native and microservices-based architectures, driven by the demand for scalability, modularity, and rapid deployment (Balalaie et al., 2016; Jamshidi et al., 2018; Mauro, 2015). In this context, Continuous Integration and Continuous Deployment (CI/CD) pipelines have become essential for enabling rapid and reliable software delivery. These pipelines automate the building, testing, and deployment processes, allowing teams to detect issues early and iterate quickly.

At Ericsson, the JCAT (Java Common Auto Tester) framework plays a critical role in the CI/CD pipeline by enabling automated testing of complex software systems. The logs generated during these tests—referred to as JCAT logs—contain detailed information on test executions, including success and failure events, performance metrics, and system behavior (C. Waite, 2008). These logs are vital for ensuring code quality and stability, as they allow engineers to verify that new changes do not introduce regressions.

However, the shift to distributed, service-oriented systems has significantly reshaped the log analysis landscape. In microservices environments, each service may generate its own logs independently, in varying formats and at different times. As (Cinque et al., 2019) observe, this heterogeneity, combined with frequent replication and runtime migration of services, makes faulty localization and monitoring particularly difficult. The lack of centralized log schemas and the dynamic interactions between services render manual log inspection inefficient and unsustainable on a scale.

In industrial settings like Ericsson, the volume of logs can be immense—with over 1,300 issue tickets generated monthly, each tied to large, distributed log sets. Manually sifting through this data to identify the root causes of test failures is both time-consuming and resource-intensive, adding substantial cognitive and operational burden on engineering teams.

To address these challenges, there is a growing need for automated tools that can not only parse and analyze logs but also summarize issues effectively and aid in diagnosing failures. Traditionally, such tasks have relied on rule-based or template-driven methods. While these approaches are useful in controlled settings, they are often rigid, maintenance-heavy, and inadequate in the face of novel or unforeseen failure scenarios (Karlsen et al., 2024).

In response, recent research has begun exploring the potential of Large Language Models (LLMs)—such as BERT, DistilRoBERTa, and GPT variants—for log interpretation and summarization. LLMs are capable of capturing deep semantic relationships within unstructured text, enabling them to support tasks like anomaly detection, log classification, and root cause analysis. For instance, (Karlsen et al., 2024) show that combining LLMs with autoencoding techniques enhances anomaly detection, while (Gopinath, 2024.) demonstrates that decoder-

based models like MistralLite and Mixtral can be fine-tuned to infer failure causes from complex logs.

Building upon these advancements, this thesis explores how to enhance the quality and usefulness of LLM-generated summaries in microservices log analysis. In particular, it investigates the role of prompt engineering—the strategic design of input text to guide LLM outputs—as a lightweight yet powerful method for improving summarization outcomes in real-world CI/CD contexts.

1.2 Problem Formulation

At Ericsson, a large language model—LLaMA 3.1–8B—has been integrated with the JCAT testing framework to enhance automated log analysis. This tool processes extensive volumes of unstructured log data, extracts meaningful patterns, and generates summaries intended to assist CI/CD engineers in diagnosing failures. While the integration has significantly reduced manual effort, the clarity and accuracy of the summaries generated remain insufficient. In many cases, the outputs lack the contextual precision required to fully support root cause identification and resolution.

Enhancing the diagnostic value of these summaries could lead to substantial operational benefits. For instance, Ericsson’s CI/CD pipelines produce over 1,300 issue tickets per month, each linked to large and complex log files. More accurate and relevant summaries could help engineers resolve issues more quickly and with less cognitive effort—ultimately improving efficiency and reducing downtime.

Although LLMs have shown great promise in processing unstructured logs, their performance in summarization tasks is highly sensitive to prompt engineering—defined as “the practice of strategically crafting input text that effectively communicates the task to an LLM, enabling it to produce outputs that meet specific objectives within a given application context” (Park et al., 2023). In instruction-tuned models, even minor variations in prompt formulation can significantly influence the clarity, relevance, and completeness of the generated output (Rodriguez et al., 2023).

Despite growing interest in LLM-based solutions for software logs, there is limited empirical research that examines how different prompting strategies affect summarization quality in real-world CI/CD environments—particularly within microservices architectures, where logs are heterogeneous and span multiple interdependent services, and manual inspection is impractical due to the volume and complexity of the data.

In such environments, automated summarization holds the potential to improve fault localization and accelerate issue resolution. However, evaluating summary quality remains challenging due to its subjective nature, domain-specific expectations, and the absence of a universal standard for what constitutes a “good” summary (Davoodijam & Alambardar Meybodi, 2024).

This study addresses this gap by investigating how prompt engineering techniques can be optimized to enhance the summarization quality of LLM outputs for microservices log analysis. The research begins with an assessment of the existing LLM-based summarization tool, then

evaluates refined prompting strategies aimed at producing more accurate, structured, and contextually useful outputs—ultimately supporting CI/CD engineers in faster and more effective failure diagnosis.

1.3 Purpose and Research Question

The purpose of this thesis is to evaluate and optimize prompt engineering techniques for LLM-based summarization of microservices logs within Ericsson’s CI/CD pipeline. The study focuses specifically on logs generated by the JCAT testing framework and containerized microservices, investigating how variations in prompt formulation influence the clarity, accuracy, and practical usefulness of the generated summaries.

A central objective is to understand how effectively these summaries support failure diagnosis from the perspective of CI/CD engineers. By aligning LLM-generated outputs with real-world diagnostic workflows, the study aims to improve operational efficiency and reduce time-to-resolution. As more accurate and contextually relevant summaries reduce the time and resources required to identify and address failures, optimizing prompt strategies becomes a critical factor in enhancing the overall value of LLM-assisted log analysis tools.

To address this purpose, the study is guided by the following research questions:

RQ1: How effective are LLM-generated summaries in conveying microservices log information to support accurate and efficient failure diagnosis within CI/CD workflows?

RQ2: What is the impact of different prompting strategies on the clarity, relevance, and diagnostic utility of LLM-generated log summaries?

1.4 Delimitation

This research is limited to the use of large language models (LLMs) for summarizing application-level logs generated by Ericsson’s cloud-native microservices architecture. The study focuses exclusively on systems deployed using containerization technologies (e.g., Docker) and orchestration platforms (e.g., Kubernetes). Architectures such as monolithic, serverless, or hybrid models fall outside the scope of this work.

The analysis is restricted to logs generated by the JCAT testing framework, which is used to validate software functionality during CI/CD processes. Logs related to infrastructure, system-level monitoring, or network telemetry are excluded. Additionally, this research does not involve training or fine-tuning LLMs. Unlike prior studies that explore model adaptation for tasks such as root cause prediction, this work concentrates solely on prompt engineering as the method for optimizing summary quality—particularly in terms of clarity, relevance, and diagnostic utility.

The evaluation is based on a curated sample of Jira issue tickets and their corresponding JCAT logs. This sample was selected to capture a range of common error types and failure scenarios

while keeping the analysis manageable. Therefore, the results are meant to provide indicative insights rather than generalizable conclusions for all log types or ticket categories.

Finally, the study does not consider real-time operational constraints, such as system latency, inference time, or computational efficiency during deployment. These aspects fall outside the scope of this thesis.

1.5 Key Concepts

Given the technical and interdisciplinary nature of this study, several specialized terms and acronyms are used throughout the thesis. To ensure clarity and accessibility for readers from different backgrounds, this section provides concise definitions of the core concepts most central to the research.

Microservices

Microservices are modular architectural components that are loosely coupled and can be independently deployed using automated pipelines, often implemented through containerized infrastructure such as Docker. While they enhance scalability and modularity, they also generate large volumes of heterogeneous logs, complicating debugging and monitoring (Cinque et al., 2019; Jamshidi et al., 2018; Karlsen et al., 2024).

Logs

Logs are structured or unstructured textual records produced by software systems that capture operational data, including test outcomes, performance metrics, and error events. In Ericsson’s environment, logs provide the main diagnostic source for test failures, supporting anomaly detection, security monitoring, and fault diagnosis (C. Waite, 2008; Karlsen et al., 2024).

Continuous Integration and Continuous Deployment (CI/CD)

CI/CD pipelines automate software building, testing, and deployment to accelerate delivery while ensuring quality. They play a central role in modern cloud-native development environments, enabling fast feedback and minimizing integration risks (Balalaie et al., 2016; Jamshidi et al., 2018).

Large Language Models (LLMs)

LLMs are deep learning models trained on massive text corpora to understand and generate natural language. They are increasingly applied to log analysis, where they can parse unstructured text, detect anomalies, and generate summaries to support efficient diagnosis (Karlsen et al., 2024; Pang et al., 2024).

Prompt Engineering

Prompt engineering refers to the practice of carefully designing and refining input instructions to guide LLMs in producing relevant, accurate, and task-specific outputs. Studies show that prompt formulation significantly affects output clarity, relevance, and diagnostic value (Rodriguez et al., 2023; Xia et al., 2024).

Summarization (Extractive vs. Abstractive)

Summarization is the task of producing concise versions of longer texts while retaining key information. Extractive summarization selects and compiles important sentences or phrases directly from the source text, while abstractive summarization generates new sentences that paraphrase and condense the content. The latter is more flexible and closer to human-written summaries but also more challenging to evaluate (Nenkova & McKeown, 2011).

JCAT (Java Common Auto Tester)

JCAT is an automated testing framework used at Ericsson to execute system-level tests and generate structured log files. These logs provide detailed information about test runs, including errors, warnings, and performance metrics, and were used as the primary data source in this thesis (C. Waite, 2008).

Jira

Jira is an issue-tracking and project management tool designed to support software development at scale. It helps teams document and track activities such as bug fixes, feature requests, code reviews, and project documentation. Because it can be customized to match different workflows, Jira has become a core system for coordinating and managing complex development projects (Fisher et al., 2013).

ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE is a widely used set of metrics for automatic evaluation of text summarization. It measures overlap between generated and reference summaries through n-grams, longest common subsequence, and skip-bigram matches (Lin, 2004).

BERT Score

BERT Score evaluates semantic similarity between generated and reference texts using contextual embeddings from transformer models such as BERT. It provides a more nuanced measure of alignment with human judgment compared to purely lexical metrics like ROUGE (Davoodijam & Alambardar Meybodi, 2024; Zhang et al., 2020).

Evaluation Metrics (Precision, Recall, F1-score)

These are common measures in classification and information retrieval tasks. Precision reflects the proportion of correctly identified positive cases among all predicted positives, while recall reflects the proportion of actual positives correctly identified. The F1-score is the harmonic means of precision and recall, balancing both measures into a single indicator of performance (Rezaei & Jabbari, 2022).

Bradley–Terry Model

The Bradley–Terry model is a probabilistic model used to derive rankings from pairwise comparison data. It assumes that the probability of one item being preferred over another

depends on the relative strength of their underlying parameters. It is widely applied in psychology, sports, and human preference studies, and in this thesis, it was used to analyze practitioner survey results (Bradley & Terry, 1952).

Elo Rating System

The Elo system is a dynamic ranking method originally developed for chess, where players' scores are updated based on match outcomes and expectations. In this study, the Elo method was applied to survey data to provide a complementary ranking of prompt strategies, capturing both static preference and dynamic performance trends (Albers & De Vries, 2001).

1.6 Thesis Outline

This section provides an overview of the structure of the thesis, which is divided into seven main chapters, each with respective sub-chapter to provide for a structured disposition.

1.5.1 Introduction

The first chapter consists of background, problem formulation, purpose and research question, and delimitation. This chapter will give an overview and introduction about the research.

1.5.2 Research Context

This chapter describes the practical and organizational context in which the research is situated. Specifically, it focuses on the operational challenges faced by CICD teams at Ericsson when diagnosing failures in microservices-based systems. It highlights the relevance of log analysis in this setting and explains why optimizing LLM-based solutions is valuable for supporting real-world troubleshooting processes.

1.5.3 Existing Research

This chapter presents the theoretical foundation of the research by reviewing existing studies relevant to the field. Its purpose is to offer a comprehensive understanding of the domain, establish the academic context, and underscore the significance of the current study.

1.5.4 Method

This chapter presents the methodology employed in this study, outlining the overall research approach and design. It describes the processes used for data collection, the structure of the experimental setup, and the methods applied for analysis.

1.5.5 Results

This chapter introduces the evaluation phase of the study, describing the experimental setup and presenting the results. It also includes the implementation of enhancement techniques and a comparative analysis of their impact on system performance.

1.5.6 Discussion

The discussion chapter interprets the experimental findings within the context of the research objectives. It examines the evaluation metrics; datasets utilized and provides a detailed analysis of the results in relation to the research questions and performance benchmarks.

1.5.7 Conclusion

This final chapter summarizes the key outcomes of the thesis, including the methodology, implementation, and experimental results. It also highlights the main conclusions drawn from the study and outlines potential directions for future work.

2. Research context

This chapter presents the operational and technical context in which the research is situated, focusing on Ericsson’s cloud-native software engineering environment and its microservices-based CI/CD pipeline. It outlines the main challenges of log management in large-scale distributed systems and describes the current LLM-powered log analysis workflow used within the company.

In addition, the chapter introduces key concepts related to prompt engineering—a technique for optimizing LLM outputs—highlighting its relevance alongside other strategies such as model fine-tuning and few-shot learning. The chapter concludes by defining the research scope and objectives, laying the groundwork for the experimental evaluation of prompting strategies in log summarization.

2.1 Collaboration with Ericsson

This study was conducted in collaboration with Ericsson and serves as a case study within the company’s cloud-native software development environment. Ericsson employs microservices architecture combined with Continuous Integration and Continuous Deployment (CI/CD) pipelines to enable scalable and efficient software delivery.

The research addresses a real-world operational challenge: the complexity of analyzing the large volumes of logs generated during automated microservice testing. By exploring and optimizing Large Language Model (LLM)-based summarization techniques, this work seeks to solve a concrete industry problem while offering insights applicable to other large-scale, distributed systems.

2.3 Microservices at Ericsson

Microservices are modular architectural components that are loosely coupled and can be independently deployed using fully automated pipelines. This architecture is typically implemented through containerized infrastructure, such as Docker (Jamshidi et al., 2018). While microservices enhance modularity and scalability, they also introduce challenges in debugging and monitoring due to the distributed nature of services and the high volume of logs they generate (Cinque et al., 2019; Karlsen et al., 2024).

To manage this complexity, Ericsson has adopted a structured CI/CD pipeline integrated with several tools and frameworks. JCAT (Java Common Auto Tester) is the primary automated testing framework, used for both functional and non-functional validation of microservices. It produces structured application-level logs—referred to as JCAT logs—that are essential for diagnosing test failures and verifying service correctness (C. Waite, 2008). These logs often serve as the initial source for identifying pre-deployment issues.

Complementing JCAT, Jira is used as the central platform for issue tracking and fault management. When test failures occur, Jira tickets are created to document the context, reproduction steps, and relevant engineering insights. This establishes a traceable workflow between automated testing and manual resolution processes (Karlsen et al., 2024).

Microservices orchestration is handled by Kubernetes, which automates service deployment, scaling, and operation. Kubernetes ensures consistency across different environments and supports dynamic scaling based on workload demands—essential in Ericsson’s high-throughput deployment settings (Usman et al., 2022).

As microservices architectures scale, the volume and complexity of log data continue to increase. Traditional rule-based log analysis approaches are becoming inadequate. In response, Ericsson has begun integrating LLMs into its workflows to automate log summarization and aid failure diagnosis. A variant of Llama-3.1-8B has been deployed to process JCAT logs and generate concise, human-readable summaries. These summaries are intended to reduce the cognitive load on CI/CD engineers by highlighting relevant information and accelerating root cause identification (Gopinath, 2024; Karlsen et al., 2024).

2.3 Large Language Model Log Analysis Tool

To enhance log analysis and error detection, Ericsson has integrated Llama-3.1-8B, a state-of-the-art LLM, with the JCAT framework. The model processes large volumes of log data, leveraging its natural language understanding capabilities to extract meaningful insights, detect anomalies, and generate concise summaries that support rapid root cause identification.

The primary goal of this tool is to manage unstructured log data and provide context-aware diagnostics. This significantly reduces the need for manual troubleshooting and helps accelerate failure resolution in complex microservices environments.

How Microservices Log Analysis Tool Works

The microservices log analyzer is a specialized tool designed to process JCAT logs and assist in identifying system issues. Using a Python-based preprocessing script, it analyzes logs to detect error patterns or anomalies. Error lines are then sent to the LLM via an internally deployed API, where summaries are generated based on both JCAT logs and related container logs.

The tool identifies the relevant log files, sends them to the LLM for summarization, extracts key insights, and adds this summary to the Jira ticket. These insights are reviewed by CI/CD engineers, who are responsible for diagnosing and resolving the issues. This process streamlines the transition from log analysis to actionable diagnosis, enhancing the efficiency of incident response in Ericsson’s development workflow.

AI tool Workflow:

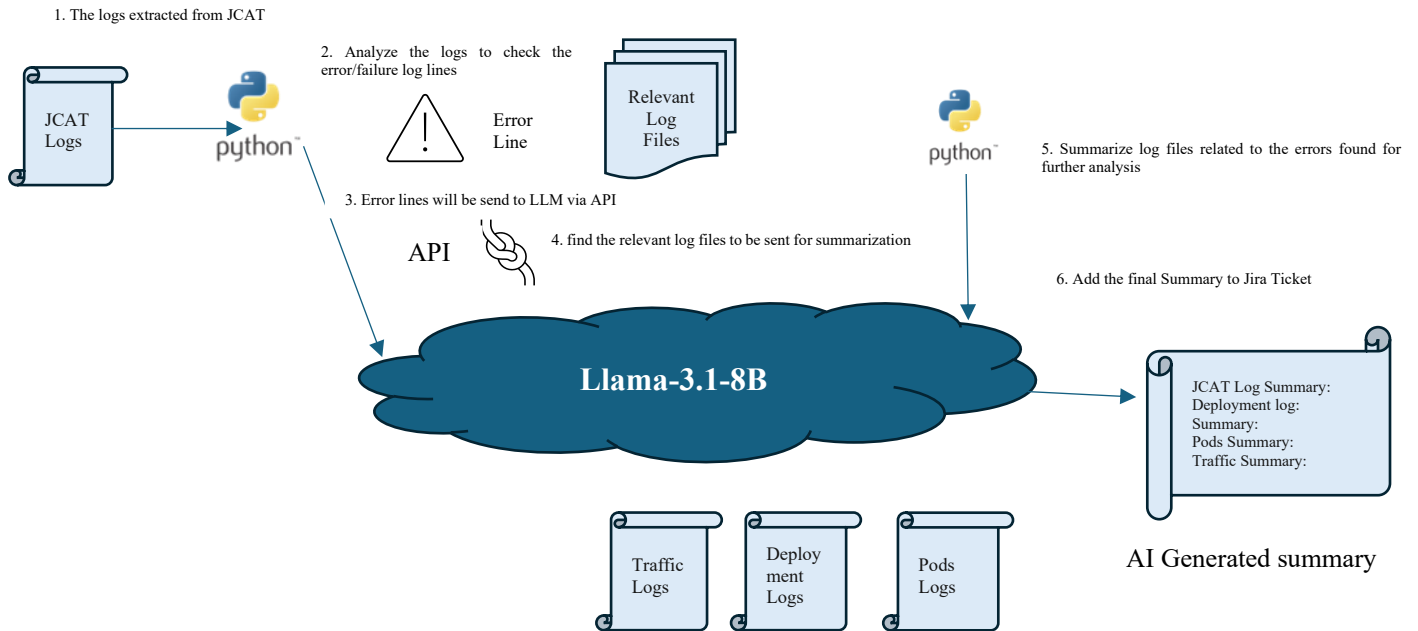


Figure 1: Microservices Log Analyzer Workflow

Figure 1 illustrates the end-to-end process of generating AI-powered summaries from JCAT logs using an LLM-based system. First, logs are extracted from JCAT and processed using a Python script to identify error or failure lines. These error lines are then sent to a large language model (LLaMA 3.1-8B) via an API. The model identifies and selects relevant log files—such as deployment, traffic, and pod logs—for further summarization. These files are individually summarized, and the results are aggregated into a final summary. The final AI-generated summary is then added as a comment to the corresponding Jira ticket, aiding engineers in diagnosing issues efficiently.

Flowchart :

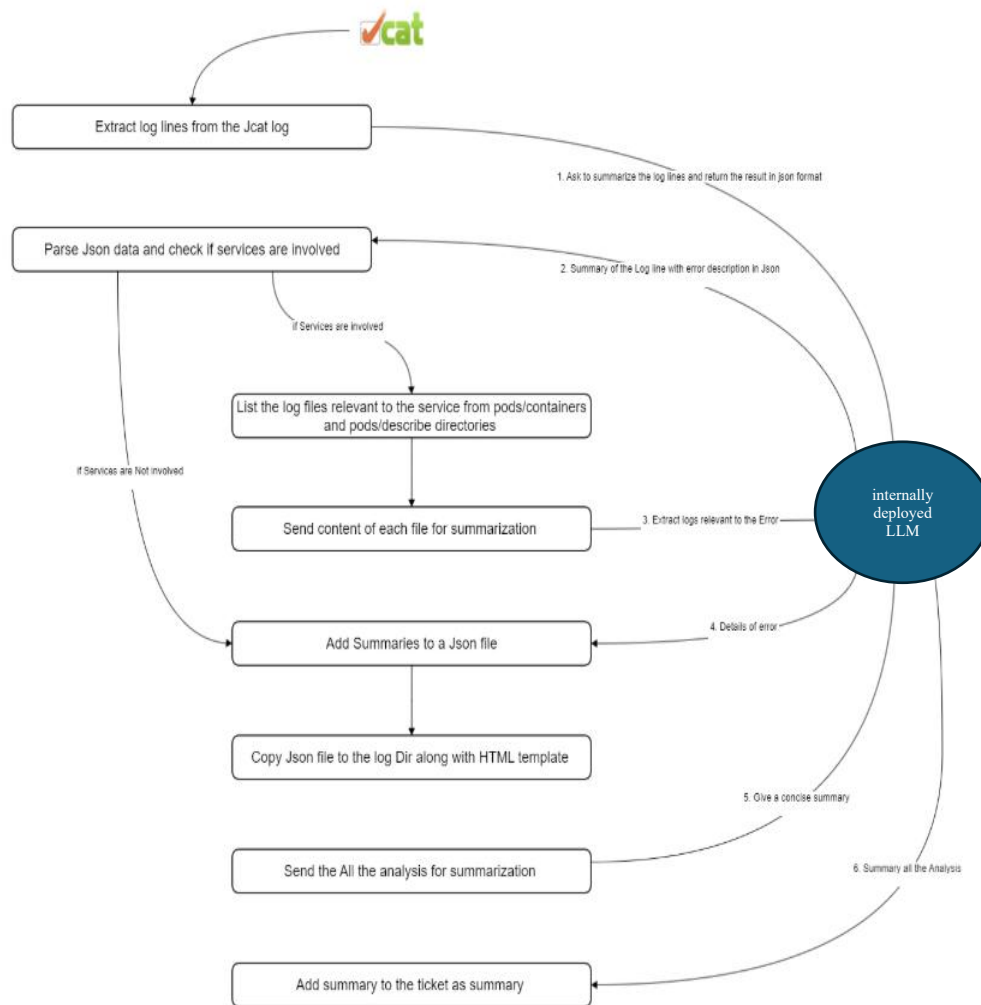


Figure 2: Workflow of the Microservices Log Analysis Tool Integrated with JCAT and LLM API.

Figure 2 illustrates the step-by-step process of how log data is extracted from JCAT, parsed for service involvement, and sent to the LLM-powered API server for summarization. Depending on the context, relevant container logs are also included. The generated summaries are stored in a JSON file, formatted for readability, and appended to the associated ticket, enabling efficient issue diagnosis and resolution.

2.4 Research Focus: Prompting Engineering for LLM based log summarization

While the integration of Llama-3.1-8B into Ericsson's CI/CD pipeline highlights the potential of Large Language Models (LLMs) for automated log summarization, the quality of the generated summaries—particularly their clarity, relevance, and diagnostic value—depends heavily on how the task is framed in the prompt.

This study does not involve model fine-tuning or retraining; instead, it focuses exclusively on prompt engineering as the primary method for improving output quality. In the context of log summarization, effective prompts can guide the model to emphasize key diagnostic insights, filter out noise, and adapt to the heterogeneous structure of microservices logs—all while complying with the operational constraints of Ericsson’s testing environment.

To this end, the research systematically evaluates several prompting strategies, assessing their influence on summary clarity, relevance, and usefulness in diagnosing system failures. The goal is to derive practical, evidence-based recommendations for designing prompts that enhance LLM performance without modifying the underlying model.

By concentrating on prompt engineering, this study seeks to optimize the generation of log summaries to better support real-world diagnostic workflows within complex CI/CD environments.

3 Existing Research

This chapter reviews related research relevant to the focus of this study, it examines prior work across five key areas: log analysis concepts in microservices, NLP and LLM techniques for unstructured log data, automated failure testing and root cause prediction, prompt engineering and its effect on LLM output quality, and evaluation metrics for assessing generated outputs. This review establishes the foundation and motivation for the research presented in this thesis. While each of these areas has advanced significantly in recent years, they are often studied in isolation. By synthesizing insights across them, this chapter highlights both the state of current knowledge and the knowledge gap that motivates this work—namely, the lack of systematic integration of prompt engineering with LLM-based log summarization and evaluation in microservices CI/CD environments.

3.1 Microservices Log Analysis Theories and Concepts:

In large-scale enterprise and networked systems, logs serve as a vital source of real-time operational data, supporting tasks such as security monitoring, anomaly detection, and user behavior analysis (Karlsen et al., 2024).

Within cloud-native microservices architectures, log analysis is critical for maintaining system reliability and performance. However, as these architectures scale in size and complexity, the volume and diversity of log data make manual inspection infeasible. Automating log analysis not only accelerates root cause identification but also reduces system downtime and operational costs. (Sarika et al., 2023).

Recent studies have demonstrated the effectiveness of machine learning methods in this domain. For instance, (Sarika et al., 2023) applied Random Forest and Gradient Boosting models to classify failures from Kubernetes cluster logs, significantly reducing the manual effort required for diagnosis.

Furthermore, integrating automated root cause prediction with LLM-based summarization enhances operational efficiency. Such integration enables not only the detection of failures but also the generation of concise, human-readable explanations (Gopinath, 2024). Therefore, optimizing LLM-driven tools is essential to reducing the cognitive load on engineers and improving overall system resilience.

3.2 Natural Language Processing (NLP) and LLM Theories:

In recent years, various Natural Language Processing (NLP) techniques have been widely applied in the field of log analysis to support tasks such as error detection, security monitoring, anomaly detection, and general log interpretation. These methods aim to transform unstructured log data into structured, actionable insights, thereby improving the efficiency of diagnosing and resolving issues in complex software systems.

A study by (Karlsen et al., 2024) explored the advantages of different LLM architectures for unsupervised learning in the domain of log analysis. The researchers fine-tuned 30 language models using a self-supervised language modeling objective, testing five distinct architectures across six diverse datasets.

(Y. Liu et al., 2024) extended this line of research by applying state-of-the-art LLMs to log analysis. Their work proposed a comprehensive engineering methodology for prompt optimization, offering strategies to improve LLM performance in log interpretation. They also introduced systematic criteria for evaluating the usefulness and readability of LLM-generated summaries, contributing to more practical and explainable AI tools in software operations and maintenance.

(Shan et al., 2024) addressed the challenge of localizing configuration errors through log analysis. Their preliminary study outlined key opportunities and obstacles, and introduced a two-stage LLM-based strategy informed by those insights to improve fault localization accuracy.

In the networking domain, (Pang et al., 2024) presented a method for optical network log analysis using LLMs, specifically utilizing LLaMA2 with instruction tuning. Their approach was tested across three core tasks—log parsing, anomaly classification, and anomaly explanation—demonstrating the adaptability of LLMs in diverse infrastructure settings.

(Enrique & Casado, n.d.) examined the role of NLP and machine learning in industrial maintenance logs. Their study demonstrated the value of LLMs for simplifying robot breakdown logs, automating alert generation, and improving workflow efficiency through the automation of maintenance work order creation.

Together, these studies highlight the growing relevance of NLP and LLMs in automated log analysis. They support the motivation of this thesis, which focuses on improving LLM output quality through prompt engineering—ultimately aiming to enhance diagnostic support within CI/CD environments.

3.3 Automatic Microservices Failure testing

Accurately identifying the root cause of microservices failures is crucial for assigning issues to the appropriate teams and facilitating timely resolution. Automating this process can significantly reduce diagnostic time and minimize the manual effort required in complex software environments. Several studies have demonstrated the potential of machine learning and LLM-based techniques to support this goal.

(Sarika et al., 2023) applied machine learning algorithms to classify logs generated from large-scale Kubernetes clusters. Given that each log file could produce up to 450 MB of data, the study showed that applying expert-driven filtering methods reduced log volume by 91%, making it more feasible to extract relevant information and pinpoint failure causes efficiently.

(Yu et al., 2024) proposed a log parsing framework that leverages the generative capabilities of Large Language Models with zero-shot prompting. This hybrid approach, combining LLMs and unsupervised models, was shown to be highly adaptable across different online log parsing scenarios. Extensive experimental evaluations on real-world alarm log datasets from cloud production environments demonstrated a marked improvement in parsing performance.

In another relevant study, (Xuhui Zhao, 2019) investigated automatic failure analysis within the JCAT testing framework used for RBC (Radio Baseband Control) systems. The study

compared rule-matching and machine learning-based approaches, concluding that while machine learning requires an initial investment in data labeling, rule-based systems offer a faster and more interpretable solution for time-sensitive log analysis tasks.

Collectively, these studies underscore the importance of developing scalable, automated techniques for microservices failure detection and diagnosis. They provide a strong foundation for exploring LLM-based summarization and root cause analysis, particularly in high-volume, production-grade environments such as Ericsson's CI/CD pipeline.

3.4 Prompt Engineering and Its Influence on the Effectiveness of LLM Responses

Prompt engineering is a critical factor in determining the quality and effectiveness of outputs generated by Large Language Models (LLMs). Even minor variations in prompt phrasing or structure can lead to substantial differences in the relevance, clarity, and completeness of the responses. This sensitivity highlights the complexity of designing effective prompts and underscores the importance of prompt formulation in domain-specific applications. (Rodriguez et al., 2023)

(Rodriguez et al., 2023) study demonstrated that carefully crafted prompts significantly improve model performance across tasks by guiding reasoning, enhancing explanatory depth, and aligning outputs with user intent. The study emphasized that prompt formulation is not merely a technical step but a strategic component in LLM-based workflows.

(Pang et al., 2024) introduced a comprehensive framework that integrates multiple prompt engineering strategies to optimize LLM performance across various NLP tasks. Their results showed that well-structured task descriptions improved output relevance and accuracy, and they stressed the importance of balancing specificity and generality depending on the application domain.

In domain-focused use cases such as spacecraft anomaly detection, prompt engineering has demonstrated quantifiable benefits. (Xia et al., 2024) applied techniques including role prompting, tone tuning, few-shot examples, and chain-of-thought prompting after fine-tuning an LLM on historical logs. Their experiments revealed that role-based prompting achieved the highest accuracy and recall, while integrating a Min-Max Ant System (AMMAS) to optimize prompt sequences increased the F1-score to 92%.

(Park et al., 2023) further explored the impact of prompt design in Korean-language LLMs. By introducing a Query Transformation Module (QTM) that restructures user prompts into semantically organized components, the study achieved an 11.46% improvement in performance across several models (e.g., SKT GPT-2 and KoGPT-3). This illustrates the benefit of structured prompting in low-resource language environments.

Additionally, prompt engineering has proven valuable in automating complex software engineering tasks, such as traceability and log analysis. (Rodriguez et al., 2023) report emphasized that iterative prompt refinement and chain-of-thought strategies substantially enhance the semantic accuracy and technical precision of generated outputs, even when models are not explicitly trained on specialized domains.

Collectively, these studies reinforce the central focus of this thesis: optimizing LLM-based summarization of microservices logs through effective prompt engineering. By leveraging prompt strategies that align with diagnostic goals, this research aims to improve the clarity and relevance of LLM-generated summaries, thereby supporting more efficient failure detection and resolution within CI/CD environments.

3.5 Automatic Evaluation Metrics for Large Language Models

Evaluating the quality of outputs generated by Large Language Models (LLMs) remains a critical and ongoing challenge in NLP. Traditional automatic evaluation metrics often rely on comparing the model-generated text against human-written references. Among the most widely used metrics are BLEU (for machine translation) and ROUGE (for summarization), both of which measure surface-level lexical similarity. However, more recent approaches—such as BERTScore—shift focus towards capturing deeper semantic similarity, leveraging contextual embeddings for more nuanced assessment (Zhang et al., 2020).

(Chen et al., 2023) examined the performance of LLMs in generating detailed code comments by employing a combination of evaluation metrics. These included BLEU for lexical similarity and two semantic-based metrics—Sentence-BERT with cosine similarity (SBCS) and Sentence-BERT with Euclidean distance (SBED). To validate the effectiveness of these metrics, human evaluations were also conducted. The study revealed a discrepancy between automatic metric scores and human judgment, especially in cases where BLEU failed to capture the semantic adequacy of the summaries. This finding underscores the limitations of relying solely on surface-level similarity metrics in tasks requiring deeper contextual understanding.

A broader evaluation framework was presented by (Davoodijam & Alambardar Meybodi, 2024), who conducted a comprehensive survey of 42 evaluation methods for automatic text summarization. The study categorized these methods based on several dimensions, including intrinsic vs. extrinsic evaluation, manual vs. automatic processes, and reference-based vs. reference-free approaches. Key metric types covered in the survey include n-gram overlap methods (e.g., ROUGE), semantic similarity methods (e.g., BERTScore), and question-answering-based techniques (e.g., QAEval). The authors concluded that no single metric is sufficient to capture all facets of summary quality, recommending the use of multiple complementary metrics for more reliable and robust evaluation.

Collectively, these studies emphasize the importance of using both lexical and semantic evaluation techniques—alongside human feedback—to effectively assess the diagnostic usefulness and clarity of LLM-generated summaries. This multi-faceted evaluation approach is particularly relevant in the context of this thesis, which aims to optimize LLM outputs for log summarization in CI/CD environments.

In summary, prior research has demonstrated the value of log analysis in microservices, the potential of LLMs for transforming unstructured logs into structured insights, and the critical role of prompt engineering and evaluation metrics in shaping output quality. However, existing work often treats these areas in isolation—focusing either on failure detection, LLM-based summarization, or evaluation—without systematically integrating prompt engineering to enhance the diagnostic usefulness of LLM-generated summaries. This study addresses that gap by investigating how prompt design can be leveraged to improve the clarity, accuracy, and evaluability of log summaries in microservices CI/CD environments.

4. Method

This chapter outlines the research methodology used to address the study’s objectives. It begins by explaining the overall research strategy and design, including the selection of LLM models, the experimental setup, and the use of Design Science as the guiding approach.

It then describes the data collection process, along with the evaluation methods—both automated metrics and human-centered techniques such as the Bradley-Terry model. The chapter concludes with a discussion of ethical considerations relevant to the study.

4.1 Research strategy

This section presents the research strategy used to evaluate and enhance the summaries generated by the LLM model for microservices log analysis. The primary objective is to assess how effectively these summaries capture system failures and to explore methods for improving their quality to better support troubleshooting processes.

As an applied research effort, this study addresses real-world challenges by optimizing the performance of a large language model in analyzing log data within CI/CD pipelines. The goal is to reduce the time and effort required for diagnosing microservices failures by producing clearer, more accurate, and informative summaries. The approach combines both experimental and iterative components, involving systematic testing of different prompting strategies and continuous refinement based on performance outcomes.

To evaluate the effectiveness of the improvements, a mixed-methods approach was employed, integrating quantitative and qualitative assessment techniques. Quantitatively, the study uses BERTScore F1 and ROUGE metrics (ROUGE-1, ROUGE-2, and ROUGE-L) to measure semantic similarity between LLM-generated summaries and human-written reference summaries. Given the limitations of automated metrics in capturing contextual and diagnostic relevance, qualitative evaluation was also conducted with input from domain experts.

Various prompting techniques were applied to generate multiple summary versions for each log. These outputs were evaluated using the selected metrics and assessed by experts to determine which strategies yielded the most relevant and contextually accurate results. This combined approach provided a robust framework for evaluating and refining the summarization capabilities of the model.

4.2 Research Design

This section outlines the research design used in this study, which is based on the Design Science Research (DSR) methodology. DSR is a well-established problem-solving paradigm in information systems research that emphasizes the creation and evaluation of artifacts to address complex, real-world challenges (Hevner et al., 2004). It provides a structured approach grounded in iterative design, experimentation, and evaluation to ensure both practical applicability and academic rigor.

Hevner introduced DSR as one of two primary paradigms in the information systems field—alongside behavioural science. While behavioural science seeks to understand existing systems and behaviours, design science adopts a proactive stance, focusing on the development and assessment of innovative IT artifacts that support critical information tasks within organizations.

4.2.1 Design Science

This subsection outlines how the study applies the seven guidelines of the Design Science Research (DSR) framework. Rather than developing a completely new artifact, the research focuses on enhancing and optimizing an existing one—Large Language Models (LLMs)—for the practical task of log summarization in cloud-native microservices. The methods used here not only improve this particular artifact but also demonstrate how similar approaches could be applied in related contexts.

Following the principles of DSR, the study aims to build a methodologically sound approach for adapting, prompting, and assessing LLMs in a way that improves their diagnostic effectiveness and usability in log analysis. The research process is structured to meet each of the seven DSR guidelines, ensuring that the evaluation of LLM-based summarization techniques is both scientifically grounded and practically relevant.

1. Design As an Artifact

The first guideline of the Design Science Research (DSR) framework, as outlined by Hevner et al. (2004), emphasizes that a key outcome of DSR should be the creation of a viable artifact that addresses a relevant organizational problem. These artifacts may take the form of constructs, models, methods, or instantiations, and are developed within the context of improving or enabling information systems. Importantly, the artifact is not viewed in isolation but is evaluated in terms of its fit with people and organizational processes.

In alignment with this guideline, the artifact central to this study is the Microservices Log Analyzer Tool, a system designed to support the analysis of microservices logs within Ericsson’s CI/CD environment. The tool automatically detects failure points and produces summaries intended to aid engineers in the quick and accurate resolution of issues.

This research supports the “design as an artifact” principle by aiming to enhance the performance of the log analyzer tool, particularly the quality and usefulness of the generated summaries. The enhancement process follows an iterative development approach, where prompt engineering strategies are systematically evaluated and refined to improve the tool’s output. This continuous improvement cycle ensures that the artifact evolves in response to practical needs, fulfilling its intended role in supporting efficient fault diagnosis and resolution in microservices-based systems.

2. Problem Relevance

The second principle of Design Science Research (DSR) emphasizes that the research must address problems that are both practically relevant and impactful within the field of information

systems. Specifically, it asserts that research efforts should result in innovative, technology-driven solutions aimed at resolving real-world business challenges. A problem, in this context, is understood as the gap between an existing state and a desired future state (Hevner et al., 2004).

In this study, the existing state is represented by the initial summaries generated by large language models (LLMs) when analyzing microservices log data—summaries that may lack clarity, precision, or diagnostic value. The desired state, therefore, is the generation of more accurate, informative, and context-aware summaries that can better assist engineers in diagnosing and resolving system issues.

Modern microservices environments produce large volumes of unstructured and complex log data, making it difficult for engineers to monitor system performance and identify failures efficiently. This research tackles this real-world challenge by exploring how LLMs can be optimized—through prompt engineering—to improve log analysis, anomaly detection, and root cause identification. By producing clearer and more actionable summaries, the approach aims to reduce time-to-diagnosis, minimize system downtime, and enhance reliability during software testing within CI/CD pipelines.

An important consideration is cost-effectiveness. While fine-tuning LLMs could potentially improve log summarization, it is often computationally expensive and financially burdensome for organizations. In contrast, prompt engineering provides a lightweight and scalable alternative that leverages the existing capabilities of LLMs without incurring the significant costs of model retraining. This makes it particularly well suited for enterprise settings such as Ericsson’s, where operational efficiency and cost management are critical.

Additionally, DSR requires that the developed artifact be relevant to its intended user community. In this case, the artifact—the enhanced LLM-based log analysis tool—is developed in collaboration with Ericsson’s CI/CD team, who also serve as its primary users and evaluators. This team is responsible for identifying and resolving software failures during microservices testing. Thus, the artifact not only addresses a pressing technical need but is also aligned with the practical requirements of an engaged and relevant stakeholder group.

3. Design Evaluation

The third principle of Design Science Research (DSR) highlights the importance of rigorous artifact evaluation. This assessment must be context-sensitive and aligned with the operational environment in which the artifact is implemented. A key aspect of evaluation involves determining how well the artifact integrates into existing IT infrastructure and how effectively it addresses the intended business problem.

Since artifact development in DSR is inherently iterative, evaluation plays a dual role: it validates the artifact’s current performance and informs its ongoing refinement. In this study, the artifact—a microservices log analysis tool powered by a Large Language Model (LLM)—is already deployed within its intended CI/CD environment at Ericsson. As such, a thorough and systematic evaluation is necessary to assess its performance and identify opportunities for enhancement.

To this end, the research adopts a mixed-methods evaluation strategy, combining both quantitative and qualitative techniques. Quantitatively, the tool’s performance is measured using automatic evaluation metrics such as ROUGE and BERTScore to assess the accuracy and semantic quality of the generated summaries. Qualitatively, the artifact is assessed through expert feedback from CI/CD engineers, who interact with the tool in real operational settings. Their insights are critical for evaluating the tool’s practical utility in diagnosing microservices failures and handling unstructured log data.

This feedback loop ensures that the artifact remains responsive to real-world diagnostic needs and supports continuous, iterative improvement in both performance and usability.

Quantitative Evaluation

The quantitative evaluation focuses on assessing the quality of LLM-generated summaries using objective, automated metrics. Rather than evaluating the model directly, these metrics compare the generated summaries to human-written reference summaries—created by developers who manually inspect microservices logs and describe the detected errors and failures.

Two primary metrics are employed: BERTScore, which measures semantic similarity, and ROUGE, which captures lexical overlap. These metrics offer interpretable, numeric scores that help determine how well the generated summaries align with expected outputs. As such, they serve as a reliable means to track performance improvements across different prompting strategies and iterations.

Qualitative Evaluation

To complement the quantitative analysis, a qualitative evaluation is conducted through expert feedback provided by Ericsson’s CI/CD engineers—the primary users of the log analysis tool. These domain experts assess the generated summaries based on their clarity, relevance, and diagnostic usefulness. The evaluation uses a Likert scale (1 to 5) to capture subjective perceptions of summary quality.

This human-centered assessment ensures that the tool not only performs well technically but also delivers practical value in real operational settings. The feedback supports iterative refinement of the tool to better align with user expectations and real-world diagnostic workflows.

By combining both quantitative metrics and qualitative expert feedback, the study adopts a holistic evaluation strategy. This dual approach provides a balanced perspective—capturing both the technical performance of the model and its practical utility in supporting failure diagnosis in complex microservices environments. It thereby strengthens the overall validity and applicability of the research outcomes.

Rationale for Mixed-Methods Evaluation

Automated evaluation metrics offer a scalable and objective method for assessing the quality of summaries generated by large language models (LLMs). However, their limitations make them insufficient as standalone indicators of performance. Metrics such as ROUGE and

BERTScore primarily focus on surface-level or semantic similarity between generated and reference texts, often overlooking critical aspects such as clarity, conciseness, coherence, and practical usefulness in real-world applications.

As noted by (Davoodijam & Alambardar Meybodi, 2024), ROUGE's reliance on n-gram overlap can penalize summaries that use different yet semantically equivalent wording. Similarly, BERTScore—though based on contextual embeddings—may still fail to fully capture interpretability or domain-specific relevance. These shortcomings are particularly evident in technical summarization tasks where terminology, detail granularity, and structure vary widely across logs and use cases.

Given these constraints, relying solely on automated metrics may lead to inaccurate or incomplete evaluations of summary quality. To address this, the present study adopts a mixed-methods evaluation approach that combines quantitative metrics with qualitative assessments from domain experts. This hybrid strategy ensures that the model's output is not only statistically aligned with reference summaries but also meets the practical needs of CI/CD engineers who rely on these summaries for failure diagnosis and resolution.

Thus, the integration of human judgment alongside automated scores provides a more comprehensive and reliable assessment framework, aligning with recommendations in recent evaluation research.

4. Research Contributions

The fourth guideline in the Design Science Research (DSR) framework emphasizes that the developed artifact must provide a clear and meaningful contribution to the field in which it is applied. Contributions may take various forms, depending on the research objectives and context. These can include demonstrating the artifact's practical utility in addressing real-world challenges, extending existing theoretical frameworks, or introducing innovative methodological approaches.

In this study, contributions are made on both practical and theoretical fronts. Practically, the research presents a reproducible and adaptable framework for integrating large language models (LLMs) into log analysis processes, specifically for enhancing system observability and facilitating efficient error detection within microservices-based environments.

Theoretically and methodologically, the study contributes insights into the application of prompt engineering as a strategy to optimize LLM output quality. By evaluating the effectiveness of different prompting techniques through a combination of automated metrics and expert assessments, the research adds to the academic discourse on LLM performance optimization.

These contributions collectively support advancements in the field of log analysis and offer actionable strategies for industry practitioners—particularly within the emerging domain of Artificial Intelligence for IT Operations (AIOps).

5. Research Rigor

The fifth guideline of the Design Science Research (DSR) framework emphasizes the need for methodological rigor in both the development and evaluation of the designed artifact. Rigor is achieved by applying well-founded theories, systematic research methodologies, and practices drawn from the established knowledge base in information systems.

In this study, rigor is maintained through the structured design and comprehensive evaluation of the proposed log analysis tool. The artifact—an LLM-integrated log analyzer embedded within the JCAT system—was developed using proven techniques from machine learning and software engineering. These include NLP-based log vectorization, unsupervised learning strategies, and LLM embeddings, all grounded in recent literature on log analysis and anomaly detection.

To ensure a robust assessment, the study employs a mixed-methods evaluation framework, combining both quantitative metrics and qualitative user feedback. The artifact is tested within its intended operational context, providing real-world validation of its functionality and integration. Additionally, expert evaluations from CI/CD engineers offer insights into the tool's practical utility, especially regarding the clarity, accuracy, and relevance of the generated summaries.

Beyond technical development, this research incorporates a rigorous experimental approach to optimizing the summarization process via prompt engineering. Various prompting strategies were designed and evaluated to guide the large language model (LLM) more effectively in generating context-aware summaries. These strategies included structural modifications, refinement of task instructions, and the use of contextual cues, informed by best practices in the LLM literature.

This approach aligns with recent studies, such as (Gopinath, 2024), who demonstrated the efficacy of structured prompting—such as Chain-of-Thought reasoning and instruction tuning—in enhancing LLM performance for root cause extraction in log data. Similarly, this study iteratively tested multiple prompt formulations to empirically identify the most effective strategies, ensuring that summary outputs align with operational needs in microservices environments.

Together, these design and evaluation efforts establish the methodological rigor of the research, as required by Guideline five of the DSR framework, and strengthen the validity and reliability of the study's findings.

6. Design as a Search Process

The sixth guideline of the Design Science Research (DSR) framework emphasizes that artifact development should follow an iterative and exploratory process. Solving complex problems rarely involves a single, definitive solution; instead, it requires a systematic search for the most effective configuration, guided by domain-specific requirements, operational constraints, and established rules.

In this study, the enhancement of large language models (LLMs) for microservices log analysis is conceptualized as an iterative search process grounded in prompt engineering. Rather than modifying model architecture or retraining with new datasets, the research focuses on how

variations in prompt design affect the quality, clarity, and diagnostic relevance of generated summaries.

To conduct this exploration, a diverse set of prompting strategies was systematically tested, including role reframing, reordering instructions and input (input-first vs. instruction-first), minimal instruction prompting, few-shot prompting, and chain-of-thought prompting. Each strategy was applied with the goal of guiding the LLM more effectively in extracting meaningful information from complex and unstructured log data.

This iterative experimentation allowed for continuous refinement of the artifact based on both automated evaluation metrics and qualitative feedback from domain experts. By identifying prompt formulations that yielded clearer, more accurate, and contextually relevant summaries, the study improved the diagnostic utility of the tool and enhanced its practical effectiveness within Ericsson's CI/CD environment.

Through this search-driven approach, the research adheres to guideline six of the DSR framework systematically exploring alternative designs, validating their impact, and converging on the most effective prompting configurations for the target use case.

7. Communication of Research

In accordance with the seventh guideline of the Design Science Research (DSR) framework, this study ensures that its contributions are effectively communicated to both academic and practitioner audiences. The research is presented with academic rigor, including comprehensive documentation of the artifact's design, prompting strategies, model configuration, experimental setup, and evaluation methods. This level of detail supports scholarly reproducibility and contributes to the broader discourse in areas such as log analysis, anomaly detection, and LLM optimization.

To ensure practical relevance, the study also translates its findings into actionable insights for industry professionals, including engineering managers, DevOps teams, and AIOps practitioners. These insights encompass deployment recommendations, performance considerations within CI/CD environments, and guidance on resource allocation when integrating LLM-based tools into existing infrastructure.

The final thesis will be published through the DiVA portal (Digitala Vetenskapliga Arkivet), providing open access and long-term availability to academic and industry stakeholders. In addition, the findings will be shared with relevant industrial collaborators and professional communities to facilitate knowledge transfer, encourage implementation, and support future research and development initiatives.

4.3 Data Collection

This section describes the data collection procedures undertaken to support the evaluation and enhancement of the Microservices Log Analyzer tool, with a focus on assessing the performance of LLM-based summarization and exploring improvements through prompt engineering.

The initial phase of data collection aimed at evaluating the existing performance of the deployed LLM within Ericsson’s CI/CD environment. To this end, approximately 1,302 Jira tickets—representing the average number of issue reports generated monthly—were retrieved via the Jira REST API. These tickets were specifically related to system failures and included logs analyzed by the LLM, manually written summaries by CI/CD engineers, and diagnostic insights recorded during issue resolution. Many of the tickets also featured human-assigned ratings or qualitative comments regarding the usefulness of the LLM-generated summaries. This dataset formed the foundation for evaluating how well the current tool configuration supports failure diagnosis.

In the second phase, additional data was collected to examine the effect of prompt engineering on the quality of generated summaries. A controlled experiment was conducted using a carefully selected subset of four representative Jira tickets. These were chosen based on the variation in their prior summary quality scores—some highly rated, others less effective—ensuring diversity in evaluation scenarios.

For each selected ticket, associated log files were used to generate new summaries using five alternative prompting strategies, in addition to the original baseline prompt. To account for variation in LLM outputs, three generations were produced per prompt per ticket, resulting in a total of 72 generated summaries (4 tickets $\times$ 6 prompt types $\times$ 3 generations each).

All experiments were conducted under identical system conditions, using the same model version, infrastructure, and hyperparameter settings (e.g., temperature), ensuring consistency across results.

This comprehensive dataset provided the foundation for a comparative analysis of different prompting strategies and their influence on summary clarity, relevance, and diagnostic value. The experimental design and evaluation results are further detailed in the subsequent Results chapter.

4.4 Analysis Methods

This section presents the analytical methods used to evaluate the quality of summaries generated by the Microservices Log Analyzer. Given the objective of producing concise, accurate, and semantically meaningful summaries from complex log data, the study adopts a mixed-methods approach—combining both quantitative and qualitative evaluation techniques—to ensure technical rigor and practical relevance.

These methods are applied in two phases: first, to assess the baseline performance of the existing Microservices Log Analyzer; and second, to systematically compare it with summaries generated using various prompting techniques introduced during the experimental phase. This dual-purpose evaluation enables a robust examination of how prompt engineering influences output quality, as judged by both automated metrics and expert feedback from practitioners.

On the quantitative side, the study employs well-established automatic evaluation metrics from the field of natural language processing (NLP), particularly those commonly used for summarization tasks involving large language models (LLMs). Traditional surface-level metrics, such as ROUGE, are used to measure lexical overlap between generated and reference summaries. However, ROUGE may fall short in capturing deeper semantic similarity, especially when paraphrasing or reordering of content occurs. To address this limitation,

BERTScore is also employed. It leverages contextual embeddings to assess semantic equivalence between texts, offering a more nuanced evaluation. Combined with standard classification metrics such as precision, recall, and F1-score, these automated methods provide a comprehensive and balanced assessment of summary quality.

In addition to the automated evaluation, a qualitative assessment is conducted through practitioner feedback. Members of the CI/CD team participate in a pairwise comparison survey, evaluating the relative quality of summaries produced using different prompt formulations. The results are analyzed using the Bradley-Terry model, a statistical method for deriving ranked preferences based on pairwise comparisons. This qualitative component ensures that the summaries generated are not only linguistically sound but also practically useful for real-world diagnostic workflows.

Together, these quantitative and qualitative analyses offer a thorough assessment of the APD Log Analyzer's performance and contribute to a deeper understanding of the role prompt engineering plays in optimizing LLM-generated log summaries.

4.4.1 Large Language Models Automatic Evaluation Metrics

The automatic evaluation of text generation and summarization by Large Language Models (LLMs) has traditionally relied on surface-level similarity metrics such as BLEU and ROUGE. While these metrics are useful for measuring exact word and phrase overlap, they often fall short in capturing semantic equivalence, especially in cases involving paraphrasing or changes in word order. (Zhang et al., 2020)

To overcome these limitations, BERTScore has emerged as a more robust alternative. It leverages contextual embeddings from pre-trained transformer models like BERT to evaluate semantic similarity between generated and reference texts. According to Zhang et al. (2020), BERTScore compares the contextual embeddings of tokens in the generated summary with those in the reference summary, using cosine similarity to identify the most semantically aligned token pairs.

Unlike traditional n-gram-based metrics, BERTScore is particularly well-suited for abstractive summarization tasks, where semantic fidelity is more important than exact lexical overlap. As noted by (Davoodijam & Alambardar Meybodi, 2024), BERTScore applies a variant of Word Mover's Distance to provide a context-aware, position-sensitive evaluation that aligns more closely with human judgment than metrics such as ROUGE alone.

BERT Score

BERT Score, introduced by (Zhang et al., 2020), is an automatic metric for text generation that computes the similarity between a reference text and the AI generated one and it is based on semantic analysis.

BERT Score evaluates text similarity using contextual embeddings from a pretrained BERT model. Instead of relying on exact word matches like BLEU or ROUGE, it computes cosine similarity between token embeddings from the generated and reference texts. The best-matching token pairs are used to calculate precision, recall, and F1 score, with F1 serving as

the final metric. This approach captures semantic meaning, making it effective for evaluating tasks like machine translation and summarization. (Zhang et al., 2020)

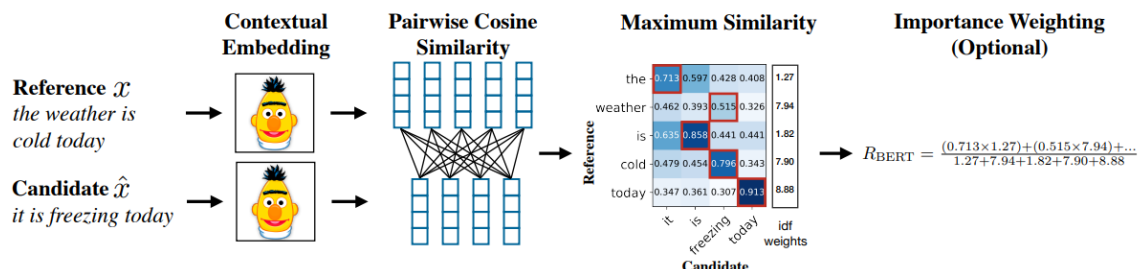


Figure 3: BERT SCORE Calculation

The above figure give a simple illustration on how BERT calculated The process involves computing contextual embeddings for the reference summary (x) and the candidate summary ($\hat{x}$), followed by calculating pairwise cosine similarities between their tokens. The illustration highlights the greedy token matching strategy in red, with optional weighting based on inverse document frequency (IDF) to emphasize the relative importance of tokens. (Zhang et al., 2020)

Recall-Oriented Understudy for Gisting Evaluation ROUGE

ROUGE is a set of metrics introduced by (Lin, 2004) for evaluating text summarization and generation quality by comparing generated text to reference summaries.

It primarily measures n-gram overlap, longest common subsequence (LCS), and skip-bigram matches between texts. The key variants include:

ROUGE-N: Measures overlap of n-grams (e.g., ROUGE-1 for unigrams, ROUGE-2 for bigrams).

ROUGE-L: Uses the Longest Common Subsequence (LCS) to capture sentence-level fluency and structure.

ROUGE-S (Skip-Bigram): Considers word pairs that appear in the same order, even if they are not consecutive.

Each variant is computed using Precision, Recall, and F1-score, where Recall is emphasized since summarization aims to retain key information. Higher ROUGE scores indicate better alignment with human-written references. (Lin, 2004)

Precision, Recall, and F1-score

Precision and recall are commonly used metrics in classification tasks. **Precision** reflects the *exactness* of a model's predictions—indicating the proportion of instances labeled as positive that are truly positive. In contrast, **recall** captures the *completeness* of a model's ability to identify positive instances—measuring the proportion of actual positive cases that were correctly labeled. Notably, recall is equivalent to **sensitivity** or the **true positive rate**. These metrics are typically calculated using the following formulas:

$$precision = \frac{TP}{TP + FP}$$

Figure 4 Precision equation

$$recall = \frac{TP}{TP + FN} = \frac{TP}{P}$$

Figure 5 Recall equation

TP: True Positive
 FP: False Positive
 FN: False Negative

The F1-score (also referred to as the F-measure or F-score) provides a unified evaluation metric by combining precision and recall into a single value. Specifically, it is the harmonic mean of precision and recall, offering a balanced measure when both metrics are considered equally important (Rezaei & Jabbari, 2022). The formula for the F1-score is:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Figure 6 F1 equation

4.4.2 Human-Centered Evaluation Using the Bradley-Terry Model and Elo rating

In addition to automated evaluation metrics, human judgment was included to assess the practical usefulness of the generated summaries. Since the Microservices Log Analyzer is intended to support CI/CD teams, participants were recruited directly from this user group.

The qualitative evaluation involved fifteen practitioners, all of whom were developers working with continuous integration and deployment pipelines. Their professional experience ranged from five to ten years (average approximately six years), and each reported prior involvement with log analysis as part of their regular responsibilities. This socio-demographic profile ensured that the feedback reflected the perspectives of practitioners who routinely interact with log data in their daily work.

Data was collected through a pairwise comparison survey. For each comparison, two summaries generated from the same log case were shown side by side. Participants were instructed to:

1. Carefully read both summaries.
2. Select the one they considered better, considering clarity, completeness, accuracy, and readability.
3. Progress automatically to the next comparison until all cases were completed.

Each participant evaluated four log cases, each containing six summaries produced by different prompt strategies.

The complete questionnaire, including survey instructions, demographic questions, and comparison format, is provided in Appendix A. This inclusion ensures transparency regarding what data was collected, how it was gathered, and how it was analyzed.

Responses were analyzed using the Bradley-Terry model, which estimates preference scores for each summary based on pairwise outcomes, and the Elo rating method, which tracks ranking dynamics across sequential comparisons.

Bradley-Terry model

To effectively capture CI/CD practitioners' preferences among summaries generated using different prompt engineering strategies, this study applies the Bradley-Terry model (Bradley & Terry, 1952)—a probabilistic framework widely used to model rankings based on pairwise comparison data. The core assumption of the model is that for any two alternatives, the probability that one is preferred over the other is a function of their associated latent “ability” parameters, which reflect their perceived quality or utility. This allows for the derivation of a global ranking across all items.

In this study, participants were shown pairs of summaries, each generated using a distinct prompting strategy. They were asked to select the version they found clearer, more relevant, and more helpful for diagnosing failures in microservices test logs.

To avoid survey fatigue and reduce cognitive load—especially since a full round-robin comparison of six summaries would require 15 pairwise evaluations per participant—a streamlined, sequential knockout format was adopted. In this format, the preferred summary in each comparison advances to face the next, limiting the number of evaluations required per user.

While this approach introduces potential bias—such as overexposing earlier candidates to more comparisons—it offers a pragmatic balance between analytical depth and participant engagement. The collected comparison data still fit within the Bradley-Terry model's structure and allow for reliable estimation of relative preferences.

Each summary is ultimately assigned an ability score (π_i), representing its likelihood of being preferred over others. The probability that summary i is preferred over summary j in a given comparison is defined as:

$$P(i \succ j) = \frac{\pi_i}{\pi_i + \pi_j}$$

Figure 7Bradely-Terry Model equation

where:

- $\pi_i, \pi_j > 0$ are the latent preference parameters for summaries i and j , respectively.

- The parameters are estimated by maximizing the likelihood of the observed pairwise outcomes.

This formulation enables estimation of a relative ranking across all summaries based on the aggregate results of binary comparisons, even when the comparisons are incomplete or unbalanced, as in the sequential design used in this study.

Elo Rating

To complement the Bradley-Terry model, this study also incorporates the Elo-rating method as an alternative approach for ranking summaries based on practitioner preferences. Originally developed for ranking chess players (Elo, 1961), the Elo-rating system updates the score of each item based on the outcome of sequential pairwise comparisons, adjusting scores proportionally to the difference between expected and actual results. This dynamic nature allows the model to track changes in perceived summary quality over time, providing a robust measure of comparative performance that accounts for the order and outcome of evaluations (Albers & De Vries, 2001).

Combining the Elo rating method with the Bradley-Terry model enables a more nuanced and robust evaluation of summary quality by capturing both static pairwise preferences and dynamic performance trends. Together, these models offer complementary strengths that enhance the reliability and interpretability of user-centered assessments in real-world CI/CD environments.

4.5 Research Object and Experimental Setup

This research was conducted on logs generated within a continuous integration and deployment (CI/CD) environment. The primary log source was the Java Common Auto Tester (JCAT), which executes microservices test cases and produces detailed logs capturing system behavior, warnings, and error traces. Jira served as the issue-tracking system, where human-written summaries were linked to the logs. For this study, four representative Jira tickets and their associated JCAT logs were selected as the dataset.

Log analysis was performed using an internally deployed log analysis tool powered by a large language model based on LLaMA-3.1-8B. The model was accessed via a custom Python interface that allowed log segments to be submitted for analysis and summarization. The workflow followed these steps:

- JCAT produced raw logs from automated microservices tests.
- A Python-based script extracted error and failure lines.
- These filtered logs, along with relevant container-level logs (e.g., deployment, pod, and traffic logs), were passed to the LLaMA-3.1-8B model via an internal API.
- The model generated intermediate summaries, which were then aggregated into a final summary.
- The summary generated was appended to the corresponding Jira ticket to support engineers in diagnosing failures.

Although other orchestration tools (e.g., Jenkins, Spinnaker) form part of the broader CI/CD pipeline, they were not directly included in this study's workflow. The research focused exclusively on JCAT (log generation), Jira (reference source), and the LLM (log analysis).

Because Ericsson's logs are confidential, this thesis refers to publicly available datasets, such as the LogHub benchmark (Zhu et al., 2019), to illustrate the general structure and complexity of microservices logs. These datasets highlight challenges similar to those in industrial settings: high volume, redundancy, a mix of structured (timestamps, identifiers) and unstructured (stack traces, free-text messages) content, and distributed dependencies. An illustrative snippet adapted from LogHub is included in Appendix B.

4.5.1 Software Environment

All experiments were implemented in Python 3.11 using Visual Studio Code as the development environment. The setup included the following libraries and dependencies:

Evaluation metrics:

- Rouge_score (ROUGE calculation)
- Bert_score (semantic similarity evaluation)
- Data handling and processing:
 - pandas (data manipulation, CSV handling)
 - NumPy (numerical operations)
 - csv, os, re, shutil, logging, traceback, time (system utilities and log parsing)
 - collections (default dictionaries for structured data)

Visualization:

- matplotlib (result plotting and visualization)
- Schema validation and testing modules:

schema, Schema_test1, Schema_test2, Schema_test3, Schema_test4, Schema_test5 (internal schema definitions used during evaluation)

Custom libraries:

- eli_log_analyser (log parsing and analysis routines, integrated with the LLM API)

Model Configuration

The internally deployed LLaMA-3.1-8B model was configured with the following generation parameters during experimentation:

```
model: "llama3.1-8b"  
max_new_tokens = 128  
temperature = 0.5  
top_p = 0.95  
top_k = 10
```

4.6 Ethical Considerations

By conducting this research on the use of large language models (LLMs) in log analysis for microservices logs, several ethical considerations are paramount to ensure the integrity and

responsibility of the study. The following outlines the key ethical concerns addressed in this project:

Data Security

Robust data security measures are implemented to protect the integrity and confidentiality of the log data. This includes secure storage solutions, encryption protocols, and stringent access controls to prevent unauthorized access, use, or disclosure of information during the research process.

Bias and Fairness

The research acknowledges the potential for bias within LLMs and datasets used in log analysis. Efforts are made to evaluate and mitigate any biases that may arise, ensuring that the models provide fair and unbiased insights. This involves critically assessing model outputs and making necessary adjustments to address identified discrepancies.

Transparency and Accountability

Transparency is maintained throughout the research process by clearly documenting methodologies, data sources, and assumptions. This ensures that the study is reproducible, and its findings are credible. Accountability is upheld by considering the potential impact of research outcomes on stakeholders, including developers, users, and Ericsson as an organization.

Intellectual Property and Proprietary Information

Respect for intellectual property rights and proprietary information is paramount, particularly given the involvement of proprietary tools and data from Ericsson. The research complies with legal and contractual obligations, ensuring that all use of such information is authorized and appropriately acknowledged.

Responsible Use of Technology

The broader implications of deploying LLMs in log analysis are carefully considered. The research evaluates whether the use of such technology might lead to unintended consequences or misuse and seeks to promote the responsible advancement of technological solutions in the field.

5. Results

This chapter presents the experimental framework and findings of the study, which investigates the effectiveness of various prompting techniques for improving the quality of large language model (LLM)-generated summaries of microservices logs in a CI/CD environment. It begins by outlining the experimental setup, including the dataset sampled from Jira-linked JCAT failures and the integration of the LLaMA 3.1–8B model used for summary generation. Multiple prompting strategies were designed and tested to evaluate their impact on the clarity, relevance, and diagnostic value of the outputs.

5.1 Experimental Datasets and Setup

The experimental setup was designed to evaluate prompt-based log summarization using a LLM within Ericsson’s CI/CD environment. A curated dataset of application-level logs, linked to Jira tickets, was sampled to represent a broad range of failure scenarios. The LLaMA 3.1–8B model was used without any additional fine-tuning, focusing instead on prompt engineering as the primary method of optimization.

Several distinct prompting strategies were applied to generate structured summaries for each case. These summaries were then evaluated using both automated similarity metrics—such as ROUGE and BERTScore—and qualitative feedback gathered from CI/CD engineers. This dual evaluation approach provides a comprehensive assessment of the impact of prompt design on the clarity, accuracy, and diagnostic usefulness of the generated summaries.

5.1.1 Dataset

The dataset for this study was collected using the Jira REST API from a microservices-based application under test within Ericsson’s software development workflow. Although this system operates within a CI/CD pipeline, the focus here is on pre-deployment testing stages, where failed automated tests generate Jira issue tickets linked to corresponding JCAT logs. These logs provide structured diagnostic information used for automated analysis and failure summarization.

To evaluate the performance of the LLM-generated summaries, four Jira issue tickets were selected. These were chosen to reflect a diverse set of failure scenarios and a range of perceived summary quality, as evaluated by domain experts. Each issue record includes:

- **Issue ID** – Unique identifier of the reported failure case.
- **JCAT Log Link** – Reference to the application-level test logs associated with the failure.
- **AI-Generated Summary** – Output of the existing LLM-based analysis tool applied to the log.
- **Human-Written Description** – Developer-authored issue description or manually created root cause summary.
- **User Ranking (1–5)** – CI/CD engineers’ rating of how useful the AI-generated

summary was for understanding and resolving the issue:

5: Very helpful; enabled resolution without manual log inspection.

1: Irrelevant or misleading; failed to reflect the actual issue.

This dataset supports both the automatic evaluation of LLM-generated summaries and the collection of user feedback for qualitative assessment.

To provide further clarity, a few representative examples from the dataset are included below.

a. Example 1: High Quality AI Summary (Rank 5)

The example below is derived from real-world operational data and illustrates the model's ability to analyze log outputs and generate a concise, accurate summary of the failure. The first figure presents a subset of the input log data—specifically focusing on pod logs, deployment logs, and traffic logs—used during the summarization process. The second figure displays the corresponding AI-generated summary. When compared with the human-authored diagnosis in the Jira ticket, the generated summary demonstrated strong semantic alignment and factual consistency.

This instance received the highest possible evaluation score (Rank 5), reflecting that the summary was clear, relevant, and sufficiently informative for issue resolution without additional log inspection. It exemplifies the potential of prompt-based LLM summarization tools in accelerating failure diagnosis in microservices environments.

Logs:

```
07T12:29:24.329+00:00", "version": "1.0.0"}
2025-02-07T13:29:24.641796934+01:00 {"extra_data":{"prometheus":{"caller":"main.go:645","mode":"server"},"message":"\\Starting Prom
(version=2.54.1, branch=heads/v2.54.1, revision=e6cfa720f6e6280153fab13090a483dbd40bece3)\\"}
2025-02-07T13:29:24.727006023+01:00 {"extra_data":{"prometheus":{"build_context":"\\(go=go1.23.4, platform=linux/amd64, user=adpgscd
platform=linux/amd64, user=adpgscd01@4343c8537155, date=20250123-06:50:51, tags=netgo,builtinassets,stringlabels)\\", "metadata":{"co
2025-02-07T13:29:24.727034714+01:00 {"extra_data":{"prometheus":{"caller":"main.go:651","host_details":"\\(Linux 5.14.21-150500.53-d
SMP PREEMPT_DYNAMIC Wed May 10 07:56:26 UTC 2023 (b630043) x86_64 eric-pm-server-0 )\\", "metadata":{"container_name":"eric-pm-server
2025-02-07T13:29:24.727611367+01:00 {"extra_data":{"prometheus":{"caller":"main.go:652","fd_limits":"\\(soft=1048576, hard=1048576)\\
0"},"service_id":"eric-pm-server","severity":"info","timestamp":"2025-02-07T12:29:24.727+00:00","version":"1.2.0"}
2025-02-07T13:29:24.730983933+01:00 {"extra_data":{"prometheus":{"caller":"main.go:653","vm_limits":"\\(soft=unlimited, hard=unlimit
0"},"service_id":"eric-pm-server","severity":"info","timestamp":"2025-02-07T12:29:24.730+00:00","version":"1.2.0"}
2025-02-07T13:29:24.732028071+01:00 {"extra_data":{"prometheus":{"address":"0.0.0.0:9090","caller":"web.go:571","component":"web"},"
07T12:29:24.731+00:00","version":"1.2.0"}
2025-02-07T13:29:24.733120964+01:00 {"extra_data":{"prometheus":{"caller":"main.go:1160"},"message":"\\Starting TSDB ...\\", "metada
2025-02-07T13:29:24.734066147+01:00 {"extra_data":{"prometheus":{"address":"[:]:9090","caller":"tls_config.go:313","component":"web
07T12:29:24.733+00:00","version":"1.2.0"}
2025-02-07T13:29:24.734820951+01:00 {"extra_data":{"prometheus":{"address":"[:]:9090","caller":"tls_config.go:352","component":"web
07T12:29:24.734+00:00","version":"1.2.0"}
2025-02-07T13:29:24.735442647+01:00 {"extra_data":{"prometheus":{"caller":"head.go:626","component":"tsdb"},"message":"\\Replaying i
07T12:29:24.735+00:00","version":"1.2.0"}
2025-02-07T13:29:24.736156660+01:00 {"extra_data":{"prometheus":{"caller":"head.go:713","component":"tsdb","duration":"7.406Âµs"},"i
server","severity":"info","timestamp":"2025-02-07T12:29:24.736+00:00","version":"1.2.0"}
2025-02-07T13:29:24.826013902+01:00 {"extra_data":{"prometheus":{"caller":"head.go:721","component":"tsdb"},"message":"\\Replaying i
07T12:29:24.825+00:00","version":"1.2.0"}
2025-02-07T13:29:24.827611336+01:00 {"extra_data":{"prometheus":{"caller":"head.go:793","component":"tsdb","maxSegment":"0","segment
07T12:29:24.827+00:00","version":"1.2.0"}}
```

Figure 8LOG Data EXAMPLE for Rank 5 ticket

Example of input log data used by the AI model for failure analysis and summarization.

AI Generated Summary Based on the log:

Initial analysis:

▼ Failure Summary (AI Analysis):

Jcat log: https://adpci.sero.wh.rnd.internal.ericsson.com/results/logs/fem30s10/cncs-lowfootprint-SingleAvailability-kaas-clean-upgrade/202502/7548_eric-pm-server_15.0.0-1/#/testcase/227
line=120

Jcat Log Summary

* Issue: HelmException during upgrade of SpiderApp application to a new version

* Error Message: Extra argument 'server.extraArgs.storage.tsdb.retention.size' not supported in server-statefulset.yaml file at line 135, column 45

* Service Name: Not explicitly mentioned, but likely related to the SpiderApp service (starts with 'eric-')

* Test Case: TC_MAINT_UPGRADE_001 failed due to the HelmException

Critical Issues Summary

* HelmException during upgrade: The test case TC_MAINT_UPGRADE_001 failed due to a HelmException during the upgrade of the SpiderApp application to a new version. The error message indicates that there is an extra argument 'server.extraArgs.storage.tsdb.retention.size' which is not supported in the server-statefulset.yaml file at line 135, column 45.

* Service Name: Not explicitly mentioned, but likely related to the SpiderApp service (starts with 'eric-').

* Test Case: TC_MAINT_UPGRADE_001 failed due to the HelmException.

Most Critical Service

The most critical issue is related to the upgrade of the SpiderApp application, which is a critical service. The error message indicates that there is an extra argument in the server-statefulset.yaml file that is not supported, causing the HelmException during the upgrade process.

Detailed Analysis: https://adpci.sero.wh.rnd.internal.ericsson.com/results/logs/fem30s10/cncs-lowfootprint-SingleAvailability-kaas-clean-upgrade/202502/7548_eric-pm-server_15.0.0-1/analysis/ai_analysis.html

Analysis Ranking: 5

▼ Issue Links

Caused by

ADPPRG-220370 PM-Server install an upgrade failure: Extra Argument 'server.extraArgs.storage.tsdb.retention.size' is not supported (missed deprecation) CLOSED

Figure 9 JIRA ticket details For Rank 5 Example

Screenshot from the issue tracking system (Jira) showing the AI-generated summary, the assigned ranking (Rank 5), and the developer's comment used for comparison and evaluation. This case illustrates how closely the AI summary aligns with the developer's original analysis.

b. Example 2 Low-Quality AI Summary (Rank 1)

The following example illustrates a real case where the AI model failed to generate a relevant or accurate summary from the log data input. The first figure presents an excerpt from the original logs analyzed—limited to pod, deployment, and traffic logs—while the second figure displays the corresponding AI-generated summary. In this case, a clear mismatch is observed between the generated summary and the actual root cause, as later identified by the developer. The summary misinterpreted the log content, lacked relevance to the true issue, and included misleading diagnostic information. As a result, it received the lowest evaluation score (Rank 1), indicating that it did not contribute meaningfully to issue resolution and required a full manual review of the logs. This example underscores the model's limitations in cases that demand precise contextual reasoning or deep domain-specific knowledge.

Logs:

```
2025-01-21T16:04:26.864578707+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.864Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Startup command: /app/Startup
2025-01-21T16:04:26.864611167+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.864Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Startup command timeout val
2025-01-21T16:04:26.869537932+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.868Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Readiness command: /app/Read
2025-01-21T16:04:26.869554904+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.868Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Readiness command timeout va
2025-01-21T16:04:26.869559335+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.868Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Liveness command: /app/Liven
2025-01-21T16:04:26.869562667+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.868Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Liveness command timeout va
2025-01-21T16:04:26.869566219+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:26.868Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"The http probe service 1.5.0
2025-01-21T16:04:26.908082523+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:26.906+00:00","service_id":"Meridio-ipam","message":"Configuration read","version":"1.0.0","extra_da
trench:7778","PrefixIPv4":"172.16.0.0/16","ConduitPrefixLengthIPv4":20,"NodePrefixLengthIPv4":24,"PrefixIPv6":"fd00::/48","ConduitPrefixLengthIPv6":56,"NodePrefixLengthIPv6":64,"IPfami
tionInterval":7200000000000,"GarbageCollectionThreshold":2400000000000}}}
2025-01-21T16:04:26.909214237+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:26.908+00:00","service_id":"Meridio-ipam","message":"NSM trace","version":"1.0.0","extra_data":{"en
2025-01-21T16:04:27.987165545+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:27.987+00:00","service_id":"Meridio-ipam","message":"GetX509Source","version":"1.0.0","extra_data":{"
2025-01-21T16:04:28.028488922+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:28.028+00:00","service_id":"Meridio-ipam","message":"Start Garbage Collector","version":"1.0.0","ex
2025-01-21T16:04:28.036115224+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:28.036+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"Name":"sj
2025-01-21T16:04:28.036170282+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:28.036+00:00","service_id":"Meridio-ipam","message":"Start the service","version":"1.0.0","extra_da
2025-01-21T16:04:28.766960747+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:28.766+00:00","service_id":"Meridio-ipam","message":"Probe service","version":"1.0.0","extra_data":{"
2025-01-21T16:04:28.769957739+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:28.769Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Readiness probe failed: serv
0","container_name":"ipam","namespace":"spider2"}}
2025-01-21T16:04:29.781947955+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:29.781+00:00","service_id":"Meridio-ipam","message":"Probe service","version":"1.0.0","extra_data":{"
2025-01-21T16:04:29.784250961+01:00 {"version":"1.2.0","timestamp":"2025-01-21T15:04:29.784Z","severity":"info","service_id":"eric-tm-senp-nvip","message":"Readiness probe failed: serv
0","container_name":"ipam","namespace":"spider2"}}
2025-01-21T16:04:30.633262825+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:30.633+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"Name":"sj
2025-01-21T16:04:30.643122237+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:30.643+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"Name":"sj
2025-01-21T16:04:37.381838091+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:37.381+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"class":"
2025-01-21T16:04:37.392548912+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:37.392+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"class":"
2025-01-21T16:04:37.418910390+01:00 {"severity":"info","timestamp":"2025-01-21T15:04:37.418+00:00","service_id":"Meridio-ipam","message":"Add","version":"1.0.0","extra_data":{"class":"
attr2c1vk-src","Cidr":"172.16.16.2/32"}}}
```

Figure 10 Data EXAMPLE for Rank 5 ticket

Example of input log data used by the AI model for failure analysis and summarization.

AI Generated Summary Based on the logs:

Initial analysis:

Failure Summary (AI Analysis):
[Jcat log: TC_MAINT_INSTALL_002[https://adpcli.sero.wh.rnd.internal.ericsson.com/results/logs/fem30s10/adp-staging-test-cnls-clean-install/202501/12779_eric-sec-ldap-
servr_15.3.0-20/#/testcase/19?line=753]
Jcat log summary:

* Assertion failed: Service Mesh sidecar not injected (expected [true] but found [false])
* Error message: Index 0 out of bounds for length 0 due to Unable to get container details
* Related context: TC_MAINT_INSTALL_002::AdpinstallTest - Install Spider-app application test case failed
* Service name: Not mentioned, but likely related to 'eric-' services

Deployment logs summary:

* No issues or errors found in deployment logs

Pod logs summary:

* No issues or errors found in pod logs

Traffic logs summary:

* Traffic stability not analyzed due to lack of traffic log data

Overall summary:

* Critical issue: Service Mesh sidecar not injected (Assertion failed)
* Most critical service: Not identified, but likely related to 'eric-' services
* Recommendation: Investigate and resolve the assertion failure in the Jcat logs

Analysis Ranking: 1

Issue Links

Caused by

ADPPRG-225959 Conflict resource booking in booking tool - n210-vpod2.spider2 resource

CLOSED

Figure 11JIRS TICKET Rank 1 EXAMPLE

Screenshot from the issue tracking system (Jira) showing the AI-generated summary, the assigned ranking (Rank 1), and the developer's comment used for comparison and evaluation. This case illustrates that the AI-generated summary does not align with the developer's written analysis.

In total, 1,302 Jira issues were retrieved, representing the average number of tickets generated over a typical month. Among these, 216 issues include an analysis ranking submitted by CI/CD engineers, while the remaining 1,086 tickets remain unrated.

This distribution provides an opportunity to explore correlations between the perceived usefulness of AI-generated summaries, their semantic similarity to human-written descriptions, and the summary quality. Furthermore, the substantial proportion of unrated tickets presents potential for semi-supervised learning or similarity-based inference techniques to estimate perceived quality, offering a foundation for future work on automated evaluation frameworks.

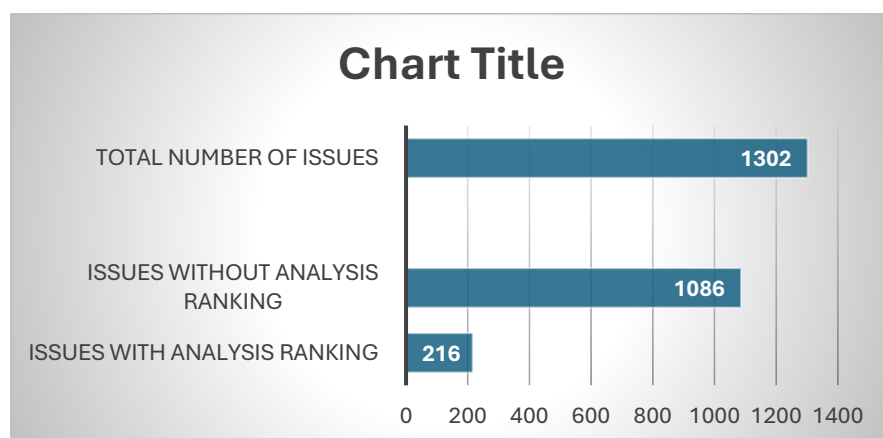


Figure 12 The monthly average Jira Tickets

5.1.2 Data Processing and Evaluation Metrics

To prepare the data for the experimental evaluation, a subset of Four issue tickets was selected from the full dataset, as described earlier. Each selected ticket included key metadata: the issue ID, a link to the corresponding JCAT logs, a developer-authored description, and, where available, a linked ticket summary that reflects a root cause or related context. For each case, the JCAT logs were extracted and formatted to serve as the input for the LLM-based summarization. In parallel, the engineer-written description and the linked summary were concatenated into a single unified text block, which served as the reference summary for automatic evaluation.

This preprocessing step ensured consistency between cases and enabled a standardized comparison of the model-generated summaries with the human-authored references using BERTScore and ROUGE similarity metrics. This approach allowed for the measurement of both semantic similarity and content overlap, providing a dual perspective on summary quality and alignment.

5.1.3 Prompt engineering configurations

To evaluate the impact of different prompting strategies on the quality of LLM-generated summaries, five distinct prompt configurations were designed and tested on the same set of log

inputs. All prompts were executed using the LLaMA 3.1–8B model under identical conditions to ensure consistency and fair comparison.

The baseline configuration was taken directly from the existing implementation currently used within Ericsson’s tool. It was applied as-is, without modification, and consists of the log data followed by a basic instruction prompting the model to generate a summary. This configuration served as the reference point against which all other prompt variants were evaluated.

The following alternative strategies were implemented:

Reordered Prompt: This version placed the instruction before the log data to assess whether changing the order of content and task specification influences the model’s output. Prior work has shown that prompt structure can significantly affect language model behavior (Y. Zhou et al., 2022).

Reframed Role Prompt: also referred to as Expert Prompting, is an advanced prompting technique aimed at enhancing the quality and relevance of responses generated by large language models (LLMs). This method instructs the model to respond as if it were a domain expert in a specific field. The underlying approach involves leveraging in-context learning to construct detailed, context-specific expert personas, which condition the model’s output. By adopting an expert role, the LLM is guided to produce more accurate, context-aware, and domain-relevant responses that reflect the depth of knowledge expected from a subject matter specialist. (Xu et al., 2025)

For instance, in the context of log analysis, the prompt may begin with a statement such as “You are an expert in microservices log analysis,” followed by task-specific instructions. This framing helps steer the model’s tone, reasoning process, and output quality toward greater precision and diagnostic value.

Minimal Instruction Prompt: In the context of large language models (LLMs), a minimal instruction prompt refers to a prompt design that includes only essential input with minimal task guidance. Typically limited to a brief phrase or single sentence, this approach reduces explicit instruction while aiming to generate coherent and meaningful responses from the model (Mann, 2020)

In this study, the minimal instruction configuration involved supplying the LLM with raw microservices log data and minimal textual guidance. This design was intended to evaluate the model’s baseline performance in a near zero-shot setting, relying primarily on its pre-trained knowledge and generalization capabilities. It aligns with the principles of zero-shot prompting, where the model is expected to perform tasks without prior exposure to specific labeled examples—drawing instead on its internalized understanding of language and context. (Yong et al., 2023)

Unlike more detailed prompts that incorporate examples, role declarations, or step-by-step instructions, minimal instruction prompting emphasizes the model’s ability to infer the task from sparse input. This approach is particularly valuable for assessing the LLM’s inherent reasoning capabilities, reducing the complexity of prompt engineering, and examining how effectively the model can operate with minimal human intervention (Mann, 2020)

Few-Shot Prompting:

In the few-shot prompting configuration, the model is provided with several example input-output pairs (i.e., logs and their corresponding high-quality summaries) before generating a new summary. These examples serve to guide the model's behavior through demonstration rather than explicit instruction.

In this case it is considered from the dataset ‘ the extracted tickets ‘ and has been rated as 5 which that their summary was clear and helpful and relevant to the actual identified error and failure, those ticket will be as This method provided one or more rank-5 examples of input logs and their high-quality summaries before the actual test case. This aligns with established few-shot prompting techniques, which improve task generalization by leveraging in-context demonstrations ((Mann, 2020); (H. Liu & Wang, 2023)).

Chain-of-Thought Prompting:

This technique was employed to simulate structured, step-by-step reasoning across multiple log sources. Rather than prompting the model to directly output a summary, this technique decomposes the task into sequential stages. Each stage guides the model to interpret one aspect of the logs—starting with high-level test context (from JCAT logs), followed by detailed error extraction, cross-verification with pod logs, and finally a synthesized failure diagnosis. This approach aligns with prior work in natural language reasoning (Wei et al., n.d.), which shows that multi-step decomposition can significantly enhance model interpretability and performance.

In the implementation, the chain-of-thought process was structured as follows:

1. JCAT Context Review – The model begins by summarizing the test case and noting initial error signs from structured test logs.
2. JCAT Deep Dive – Next, it provides a detailed explanation of the test outcome, failure cause, and affected services, returned in a structured JSON format.
3. Log Correlation – The model is asked to verify if runtime pod logs align with earlier JCAT findings and identify additional patterns.
4. Final Summary – A final prompt synthesizes all prior observations into a comprehensive failure diagnosis and executive summary.

Each of these techniques was applied to a fixed set of Four issue tickets, and their resulting summaries were assessed using automatic similarity metrics (BERTScore and ROUGE) and taking the human written summary as the reference for calculating the similarity, alongside feedback from CI/CD engineers. The objective was to determine which prompting strategies yielded the most accurate and actionable summaries in the context of microservices log analysis.

To clarify the methodological distinctions between the various prompting strategies employed in this study, the below table summarizes the structure of each prompt variant used in the experiments. While all techniques utilize the same underlying LLM configuration, they differ in how instructions are framed, ordered, or supplemented with additional context.

The table below presents concise prompt snippets that reflect the functional intent of each technique, whether emphasizing instruction placement, role assignment, brevity, or exemplar-based prompting.

Prompting Technique	Prompt Snippet (structure only)
Baseline	I have provided some JSON data, and I need you to summarize the information strictly in JSON format using the specified schema. Log: {...JCAT log in JSON format...} Schema: { "Error_discription": "...", "testcase_description": "...", "PODS": [...], "DEPLOYMENTS": [...] } Instructions: 1. Identify and summarize issues, failures, or assertions with any related error messages. 2. Describe the test case involved. 3. List the pods and deployments.
Reordered	Please summarize this structured JSON test log based on the instructions and schema below: 1. Identify and summarize issues, failures, or assertions with any related error messages. 2. Describe the test case involved. 3. List the pods and deployments. Schema: { "Error_discription": "...", "testcase_description": "...", "PODS": [...], "DEPLOYMENTS": [...] } Log: {...JCAT log in JSON format...}
Reframed Role	You are a specialized AI assistant trained in debugging Kubernetes test pipelines. Your job is to extract failure patterns and generate accurate JSON summaries. Summarize the following JCAT test log using the schema below: <pre>{ "Error_discription": "Describe in detail about the issue/failure/assertion", "testcase_description": "Describe in detail about the testcase", "PODS": ["only list of pod names"],</pre>

	"DEPLOYMENTS": ["only list of deployment names"] } Input JCAT Log: {...log content...}
Minimal Instructions	Summarize the following JCAT log content using this schema for response: <pre>{ "Error_discription": "...", "testcse_description": "...", "PODS": [], "DEPLOYMENTS": [] }</pre> Log: {...log content...}
Few Shot	Example 1: Input Log: <Log content from ticket with Rank 5> Output Summary: <pre>{ "Error_discription": "Service A failed to deploy due to missing volume configuration.", "testcse_description": "Upgrade test for Service A during staging.", "PODS": ["service-a-pod"], "DEPLOYMENTS": ["service-a-deploy"] }</pre> Example 2: Input Log: <Another high-quality ticket log> Output Summary: <pre>{ "Error_discription": "...", "testcse_description": "...", "PODS": [...], "DEPLOYMENTS": [...] }</pre> Now, summarize the following log using the same format: Input Log: <New log to be summarized>
Chain of Thoughts	Step 1: Review the following JCAT JSON log and provide a high-level summary: - What is the test case about? - Are there any visible errors or service mentions? Input: {JCAT_JSON_Log} ↓ Step 2: Based on the summary from Step 1, give a detailed diagnosis: - Describe the specific failure. - What is being tested? - Which pods or deployments are affected?

	↓ Step 3: Analyze the following pod logs and compare with Step 2: {Pod_Logs} ↓ Step 4: Integrate findings from JCAT, pod logs, and deployment logs into a final summary.
--	--

5.2 Performance Evaluation Results

The experiment involved four Jira tickets and their corresponding JCAT logs. To ensure consistency, all summaries were generated using the same model version, infrastructure, and hyperparameter settings (e.g., temperature) to focus on prompting Engineering. For each prompting technique, three summaries were produced per ticket to account for variability in the model’s outputs, resulting in 72 generated summaries in total (4 tickets × 6 prompt types × 3 generations). The evaluation combined automated metrics with user feedback. In the automated assessment, a human-written summary served as the reference for calculating similarity scores using BERTScore and ROUGE.

Then using the human written summary for each ticket as a reference to calculate BERT score and rouge scores for each group of summaries and take the average score,

5.2.1 Expert user Assessment Results

To evaluate the practical value of the generated summaries, expert feedback was collected from members of the CICD team—engineers who routinely work with microservices logs. These participants assessed the summaries produced by different prompting techniques based on criteria including clarity, usefulness, and accuracy.

The survey involved four Jira tickets previously used in the experiments, each associated with six different summaries generated through distinct prompt engineering strategies. Participants were shown pairs of summaries and asked to choose the one they found more effective in identifying failures within microservices test pipelines. This pairwise comparison approach was analyzed using the Bradley-Terry model to determine the relative ranking of the prompting techniques.

As illustrated in the figure below, the Minimum Instruction prompting strategy received the highest overall preference, indicating it was most favored by the CICD team. This was followed closely by the Role Reframing technique, which also showed strong performance in terms of clarity and relevance.

In contrast, the Few-Shot and Chain-of-Thought prompting methods received the lowest preferences. In the case of Few-Shot prompting, this may be attributed to the limited generalizability of the examples used, which might not fully capture the variety and complexity of failure scenarios encountered in microservices environments. Similarly, the Chain-of-

Thought approach may have introduced excessive verbosity, reducing clarity and hindering practical usefulness in time-sensitive operational contexts.

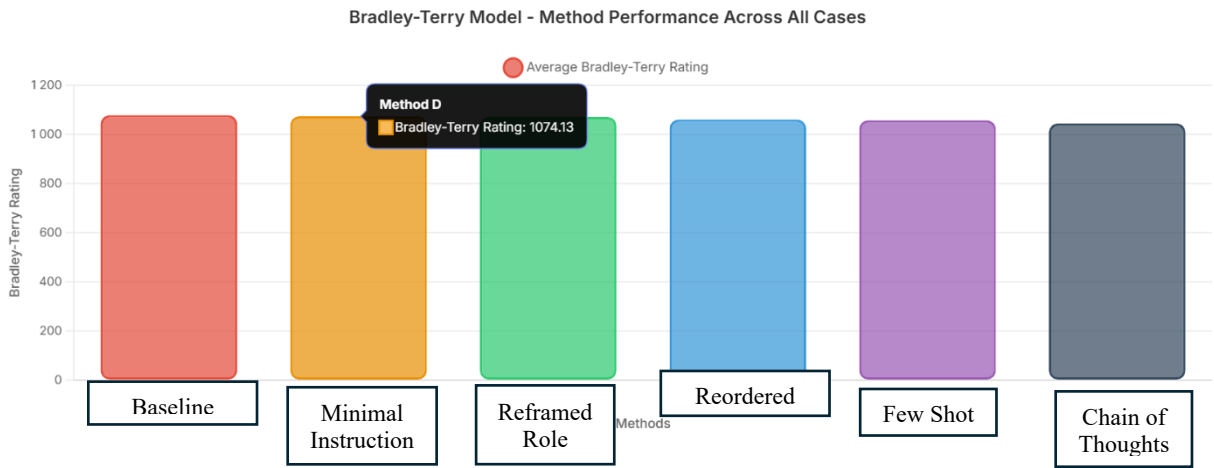


Figure 13Bradley Model survey Results

The figure above presents the survey results analyzed using the Bradley-Terry model, indicating that the baseline prompt achieved the highest score, followed by the minimal instruction prompt and the reframed role prompt.

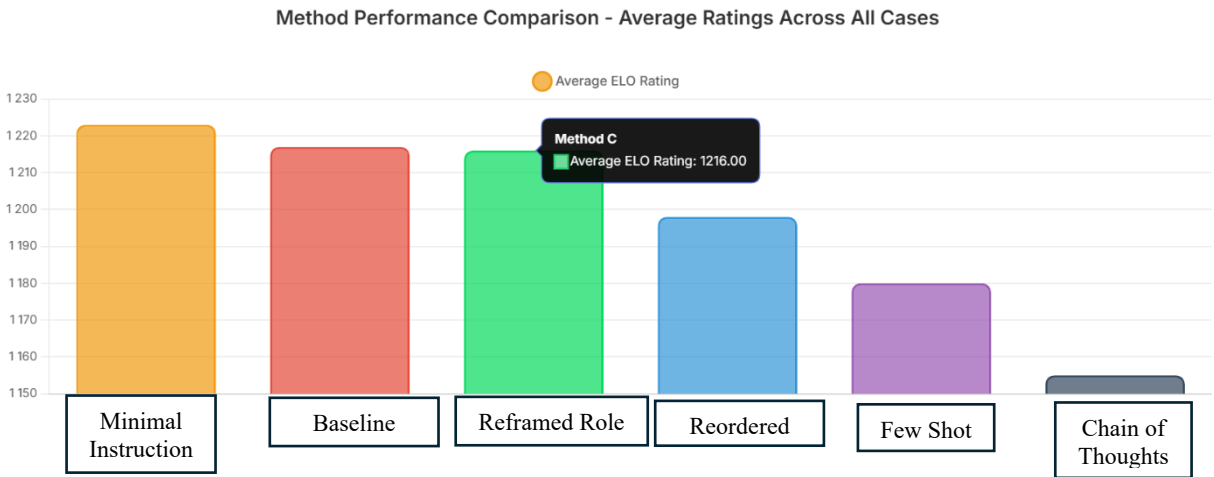


Figure 14 ELO Rating Survey Results

The figure above presents the survey results analyzed using the Elo rating, indicating that the minimal instruction prompt achieved the highest score, followed by the baseline prompt and the reframed role prompt.

5.2.2 Automatic Evaluation Results

Before deep dive in the results of the automatic metrics it is important to mention that the automatic evaluation metrics which are Bert score and rouge their models were used as open-

source implementation and no additional training or fine-tuning was performed on the models prior to testing.

ROUGE Metric Results

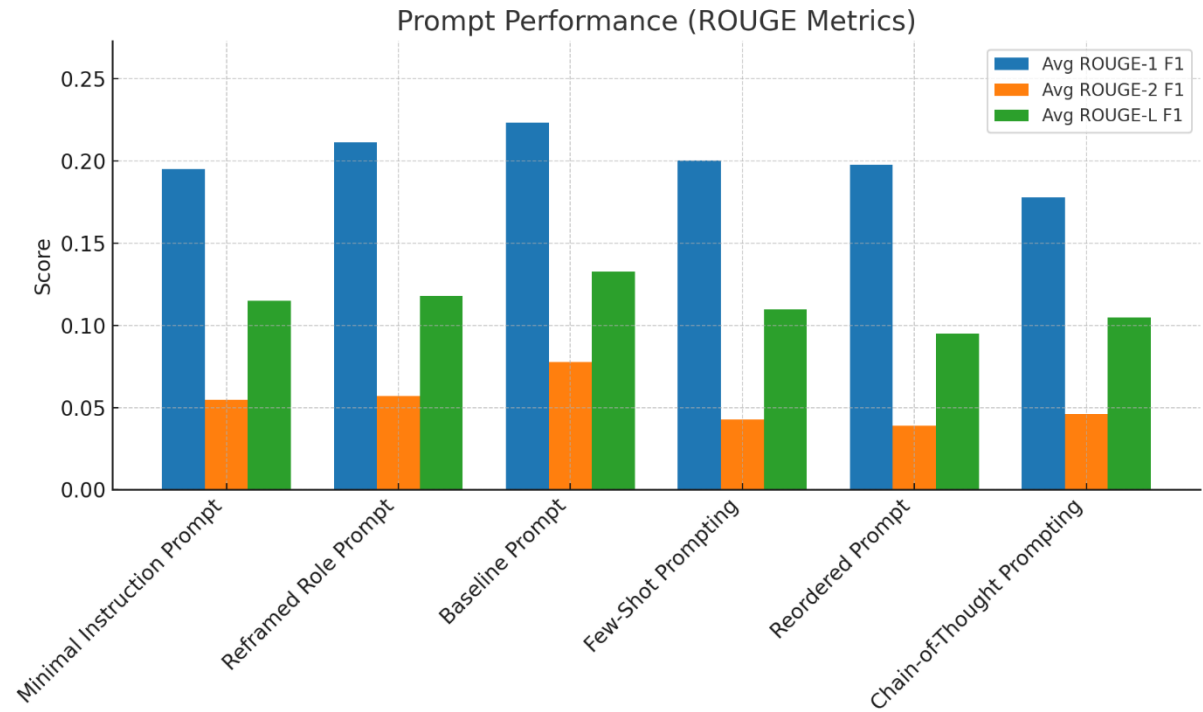


Figure 15Rouge Metric Results

Across all three ROUGE metrics, baseline Prompt achieved the highest scores, with ROUGE-1 F1 at 0.2233, ROUGE-2 F1 at 0.0777, and ROUGE-L F1 at 0.1326. These results indicate that baseline Prompt produced summaries with the greatest lexical and structural overlap with the human-written references. The Reframed Role Prompt and Minimal Instruction Prompt followed closely, showing competitive performance across all ROUGE variants, particularly in ROUGE-1 and ROUGE-L, which capture unigram overlap and sequence similarity respectively. In contrast, Few-Shot Prompting and Chain-of-Thought Prompting recorded lower ROUGE scores, suggesting a greater divergence in word choice and phrasing from the reference summaries. This pattern highlights a trade-off: prompts that adhere closely to the reference wording score higher in ROUGE, while those that prioritize reasoning or rephrasing may sacrifice exact lexical matches.

It was initially expected that Few-Shot Prompting and Chain-of-Thought Prompting would achieve the highest scores, given their potential to produce more context-aware and coherent summaries. However, the ROUGE metrics evaluate performance primarily based on lexical overlap and sequence similarity with the reference text.

The table below presents the ROUGE evaluation scores (ROUGE-1 F1, ROUGE-2 F1, and ROUGE-L F1) for each prompting strategy. The results indicate that the baseline prompt achieved the highest overall scores across all three ROUGE metrics, followed by the reframed role prompt and the minimal instructions prompt.

Table 1 ROUGE Metric Results

Prompt Version	Avg ROUGE-1 F1	Avg ROUGE-2 F1	Avg ROUGE-L F1
Baseline	0.2233	0.0777	0.1325
Reframed Role Prompt	0.2113	0.0571	0.1181
Minimal Instructions Prompt	0.1952	0.0548	0.1151
Few-Shots Prompt	0.2003	0.0428	0.1097
Reordered Prompt	0.1978	0.0389	0.0950
Chain Of Thoughts Prompting	0.1780	0.0459	0.1048

BERT-SCORE Results

Upon analyzing the results for BERT-scores, a notable pattern emerged: as shown in the table below, the BERTScores for the six different summaries were remarkably close, with no substantial differences that could clearly indicate one as superior. This is expected, given that BERTScore primarily focuses on measuring semantic similarity, and all summaries were derived from the same source—the JCAT logs—albeit generated using different prompting strategies.

This outcome does not diminish the metric’s usefulness; rather, it reflects its underlying design. BERTScore assesses semantic proximity, not task relevance or factual correctness. Consequently, a summary that is syntactically and semantically similar to the reference, yet factually inaccurate or uninformative, may still receive a high score. This finding aligns with limitations discussed in prior research, where automatic evaluation metrics, including BERTScore, may diverge from human judgment in task-specific contexts, particularly when factual grounding or domain-specific accuracy is critical (Davoodijam & Meybodi, 2024, pp. 7721–7724).

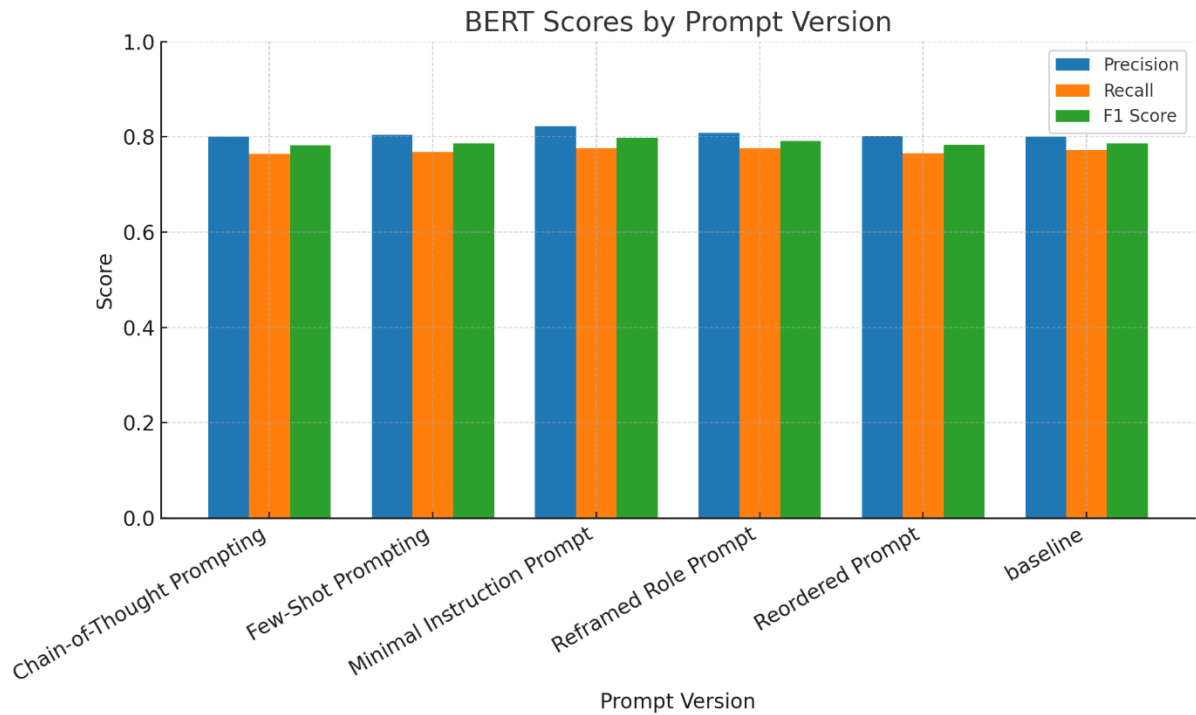


Figure 16 BERTSCORE Metric Results

The table below shows that across all three BERTScore metrics, the *Minimal Instruction Prompt* achieved the highest scores, with Precision at 0.8096, Recall at 0.7722, and F1 at 0.7902. The *Reframed Role Prompt* followed closely, recording Precision of 0.8049, Recall of 0.7716, and F1 of 0.7875. These results indicate that both prompting strategies produced summaries with the strongest semantic similarity to the human-written references. The *Few-Shot Prompting* and *Reordered Prompt* approaches yielded moderately lower scores, reflecting a slight decrease in semantic alignment while still maintaining substantial overlap with the reference summaries. The *Chain-of-Thought Prompting* method recorded the lowest BERTScores across all metrics, suggesting that while it may encourage more elaborate reasoning, it can also introduce deviations in meaning or structure that reduce measured similarity.

Table 2 BERT SCORE Results

Prompt Version	Avg BERT Precision	Avg BERT Recall	Avg BERT F1	Average BERT
Minimal Instruction Prompt	0.8220	0.7762	0.7984	0.7989
Reframed Role Prompt	0.8082	0.7755	0.7915	0.7917
Few-Shot Prompting	0.8042	0.7678	0.7855	0.7858
baseline	0.8005	0.7713	0.7856	0.7858
Reordered Prompt	0.8021	0.7655	0.7832	0.7836
Chain-of-Thought Prompting	0.8009	0.7643	0.7821	0.7824

6. Discussion

This chapter discusses the findings of the study in relation to the research questions and objectives. The analysis is based on a small but representative set of four Jira tickets, making the study exploratory and inductive in nature. The results therefore provide indicative insights into how large language models (LLMs) can be optimized for microservices log summarization, while recognizing that broader generalizations require further validation. The discussion is organized around the two research questions, followed by reflections on the study's contributions, practical implications, and limitations.

6.1 Research questions

RQ1: How effective are LLM-generated summaries in conveying microservices log information to support accurate and efficient failure diagnosis within CI/CD workflows?

The baseline evaluation of Ericsson's existing LLM-based log analyzer showed mixed effectiveness. Out of 216 ranked Jira tickets, engineers rated some summaries as highly useful (Rank 5), while others were deemed irrelevant or misleading (Rank 1). Automated metrics (ROUGE and BERTScore) confirmed this inconsistency, with scores varying significantly across cases. These results indicate that while LLMs can generate concise and relevant summaries, their effectiveness is not uniform and depends heavily on the way tasks are framed in prompts. In other words, the tool demonstrates potential but requires systematic refinement to consistently support accurate and efficient failure diagnosis.

RQ2: What is the impact of different prompting strategies on the clarity, relevance, and diagnostic utility of LLM-generated log summaries?

The comparison of six prompting strategies revealed clear differences in their effectiveness. Minimal Instruction and Role Reframing produced the highest-quality outputs, balancing clarity, relevance, and diagnostic value. In contrast, Few-Shot prompting suffered from limited coverage of examples, while Chain-of-Thought prompting often introduced unnecessary detail that reduced clarity. These findings show that concise, targeted prompts are more effective than verbose or example-heavy approaches in this domain. The results highlight the importance of domain-aware prompt engineering as a lightweight yet powerful method for optimizing LLM outputs.

6.3 Knowledge contribution

This study contributes to the understanding of prompt engineering in industrial applications. It demonstrates that carefully designed prompts can significantly improve LLM performance in microservices log summarization. The integration of automated metrics with human feedback also provides a dual-perspective evaluation framework that balances technical quality with practical usability.

6.4 Practical Implications

For Ericsson and similar industrial environments, the findings offer actionable insights. First, prompt engineering can substantially improve the quality of log summaries without requiring model fine-tuning or retraining—making it a cost-effective strategy. Second, the results suggest that CI/CD teams can achieve better diagnostic support by adopting concise and role-specific prompts, which reduce cognitive load and improve interpretability. Finally, this research demonstrates that even with limited datasets, structured experimentation can yield insights that translate directly into operational improvements.

6.4 Limitations

This study is exploratory and inductive in nature. The experimental evaluation was based on a small dataset of four representative Jira tickets. While this design allowed for detailed, controlled analysis, it limits the generalizability of the results. Furthermore, the evaluation focused exclusively on pre-deployment JCAT logs within Ericsson’s CI/CD pipeline; findings may not extend to other log types, architectures, or industrial contexts. Another limitation is that practitioner feedback, while valuable, involved a limited number of participants, which may introduce bias.

By contrast, Few-Shot and Chain-of-Thought prompting strategies proved less effective in this domain. The limited generalization capability of Few-Shot prompts may be due to the narrow coverage of failure scenarios in the examples used. Meanwhile, Chain-of-Thought prompting—although beneficial for reasoning tasks—often introduced verbose or tangential content that reduced the relevance and clarity of the generated summaries.

7. Conclusion and Future Research

This final chapter summarizes the key outcomes of the thesis, including the applied methodology, experimental implementation, and observed results. It also presents the primary conclusions drawn from the study and outlines potential directions for future research.

The main objective of this research was to investigate and optimize prompt engineering techniques to enhance the performance of large language models (LLMs) in summarizing microservices logs, with a particular focus on identifying errors and failures. The study was motivated by a real-world need to support Ericsson's CI/CD team in efficiently diagnosing issues within complex and distributed systems. To address this challenge, a set of experiments was conducted using Jira tickets and associated microservices log data. Six prompting strategies were evaluated: Reordered Input, Minimal Instruction, Role Reframing, Few-Shot Prompting, Chain-of-Thought Prompting, and a baseline representing the tool's original configuration.

Each of these strategies was applied to generate summaries, which were then evaluated both quantitatively and qualitatively. Automated metrics such as BLEU, ROUGE, and BERTScore were used to assess the linguistic and semantic similarity of the generated summaries against expert-written references. In addition, CI/CD engineers participated in a pairwise comparison survey to provide practitioner feedback, which was analyzed using the Bradley-Terry model to rank the different prompting strategies.

The findings revealed that the Minimal Instruction prompt consistently yielded the most preferred results, closely followed by the Role Reframing prompt. These strategies produced summaries that were clearer, more relevant, and better aligned with the diagnostic needs of the engineers. In contrast, Few-Shot and Chain-of-Thought prompts performed less effectively, likely due to the limited generalizability of few-shot examples and the verbosity introduced by thought-chain reasoning. These outcomes highlight that concise and well-focused prompts enable LLMs to better extract critical information from log data without unnecessary elaboration.

Overall, the study confirms that prompt engineering plays a decisive role in enhancing LLM-generated outputs for domain-specific applications such as log analysis. It demonstrates that significant performance improvements can be achieved without requiring model fine-tuning or architectural changes. This has important implications for operational environments like CI/CD pipelines, where clarity, speed, and scalability are essential.

Looking ahead, future research may build upon this foundation by exploring adaptive prompting strategies that respond dynamically to the content and context of logs. Expanding the study to multilingual log environments or integrating LLMs more deeply into root cause analysis workflows could also broaden the applicability of this approach. Additionally, improving the few-shot method through better-curated examples and incorporating real-time feedback from practitioners could further refine prompt engineering practices. By addressing these areas, future work can continue to bridge the gap between general-purpose LLMs and domain-specific operational tasks, enhancing the practical value of AI-driven tools in industry.

7.1 Future work

While this study concentrated on optimizing LLM-generated log summaries through prompt engineering, there remains considerable scope for further improving model performance in the domain of microservices log analysis. One promising direction for future research involves applying model-level optimization techniques, such as domain-specific fine-tuning or continued pretraining using logs from the target operational environment. These methods could enhance the model's ability to interpret system-specific terminology, log patterns, and contextual dependencies more effectively.

Although prompt engineering was prioritized in this research due to its practicality and low computational cost, future studies may investigate hybrid prompting strategies. For instance, combining minimal instruction with selective role-based framing, or tailoring the prompt structure dynamically based on the characteristics of the log data, could further improve summary relevance and clarity. Additionally, the development of adaptive prompt selection mechanisms—where the choice of prompting strategy is informed by the type of failure, service component, or log structure—could lead to more intelligent and context-aware summarization.

Such explorations would deepen the current understanding of how large language models can be customized and deployed in CI/CD environments, advancing both theoretical and practical contributions to the field of intelligent log analysis.

References:

- Albers, P. C. H., & De Vries, H. (2001). Elo-rating as a tool in the sequential estimation of dominance strengths. In *Animal Behaviour* (Vol. 61, Issue 2, pp. 489–495). Academic Press. <https://doi.org/10.1006/anbe.2000.1571>
- Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In *IEEE Software* (Vol. 33, Issue 3). <https://doi.org/10.1109/MS.2016.64>
- Bradley, R. A., & Terry, M. E. (1952). Biometrika Trust Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. In *Source: Biometrika* (Vol. 39, Issue 3). <https://about.jstor.org/terms>
- C. Waite. (2008). *JUnit Common Auto Tester Framework User Guide*.
- Chen, M., Zhang, H., Wan, C., Wei, Z., Xu, Y., Wang, J., & Gu, X. (2023). *On the Effectiveness of Large Language Models in Domain-Specific Code Generation*. <https://doi.org/10.1109/QRS62785.2024.00030>
- Cinque, M., Della Corte, R., & Pecchia, A. (2019). Advancing monitoring in microservices systems. *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*. <https://doi.org/10.1109/ISSREW.2019.00060>
- Davoodijam, E., & Alambardar Meybodi, M. (2024). Evaluation metrics on text summarization: comprehensive survey. *Knowledge and Information Systems*. <https://doi.org/10.1007/s10115-024-02217-0>
- Enrique, O., & Casado, L. (n.d.). *Improving Industrial Maintenance: Leveraging ML and NLP for better Breakdown Log Analysis and Automated Workflows*. www.hv.se
- Fisher, J., Koning, D., & Ludwigsen, A. P. (2013). *Utilizing Atlassian Jira For Large-Scale Software Development Management**.
- Gopinath, A. M. (2024). *Root Cause Prediction from Log Data using Large Language Models*. www.liu.se
- Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. In *IEEE Software* (Vol. 35, Issue 3). <https://doi.org/10.1109/MS.2018.2141039>
- Karlsen, E., Luo, X., Zincir-Heywood, N., & Heywood, M. (2024). Large language models and unsupervised feature learning: implications for log analysis. *Annales Des Telecommunications/Annals of Telecommunications*. <https://doi.org/10.1007/s12243-024-01028-2>
- Lin, C.-Y. (2004). *ROUGE: A Package for Automatic Evaluation of Summaries*.
- Liu, H., & Wang, Y. (2023). *Towards Informative Few-Shot Prompt with Maximum Information Gain for In-Context Learning*. <http://arxiv.org/abs/2310.08923>
- Liu, Y., Tao, S., Meng, W., Wang, J., Ma, W., Chen, Y., Zhao, Y., Yang, H., & Jiang, Y. (2024). Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies. *IEEE International Conference on Program Comprehension*, 35–46. <https://doi.org/10.1145/3643916.3644408>
- Mann, B. (n.d.). *Language Models are Few-Shot Learners*.
- Mauro, T. (2015). Adopting Microservices at Netflix: Lessons for Architectural Design. In *Nginx Blog*.
- Nenkova, A., & McKeown, K. (2011). Automatic summarization. In *Foundations and Trends in Information Retrieval* (Vol. 5, Issues 2–3, pp. 103–233). <https://doi.org/10.1561/15000000015>
- Pang, Y., Zhang, M., Liu, Y., Li, X., Wang, Y., Huan, Y., Liu, Z., Li, J., & Wang, D. (2024). Large language model-based optical network log analysis using LLaMA2 with instruction

- tuning. *Journal of Optical Communications and Networking*, 16(11), 1116–1132. <https://doi.org/10.1364/JOCN.527874>
- Park, D., An, G. T., Kamyod, C., & Kim, C. G. (2023). A Study on Performance Improvement of Prompt Engineering for Generative AI with a Large Language Model. *Journal of Web Engineering*, 22(8), 1187–1206. <https://doi.org/10.13052/jwe1540-9589.2285>
- Rezaei, N., & Jabbari, P. (2022). Naïve Bayes’ classifiers in R. In *Immunoinformatics of Cancers* (pp. 71–85). Elsevier. <https://doi.org/10.1016/b978-0-12-822400-7.00010-5>
- Rodriguez, A. D., Dearstyne, K. R., & Cleland-Huang, J. (2023). Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability. *Proceedings - 31st IEEE International Requirements Engineering Conference Workshops, REW 2023*, 455–464. <https://doi.org/10.1109/REW57809.2023.00087>
- Sarika, P. K., Badampudi, D., Josyula, S. P., & Usman, M. (2023). Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3593434.3593472>
- Shan, S., Huo, Y., Su, Y., Li, Y., Li, D., & Zheng, Z. (2024). *Face It Yourself: An LLM-Based Two-Stage Strategy to Localize Configuration Errors via Logs*. <https://doi.org/10.1145/3650212.3652106>
- Usman, M., Badampudi, D., Smith, C., & Nayak, H. (2022). An Ecosystem for the Large-Scale Reuse of Microservices in a Cloud-Native Context. *IEEE Software*, 39(5). <https://doi.org/10.1109/MS.2022.3167447>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi Quoc, E. H., Le, V., & Zhou, D. (n.d.). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting*.
- Xia, Q., Zhao, H., & Liu, M. (2024). Prompt Engineering Approach Study for Supervised Fine-Tuned (SFT) Large Language Models (LLMs) in Spacecraft Fault Diagnosis. *Proceedings of the 3rd Conference on Fully Actuated System Theory and Applications, FASTA 2024*, 819–824. <https://doi.org/10.1109/FASTA61401.2024.10595331>
- Xu, B., Yang, A., Lin, J., Wang, Q., Zhou, C., Zhang, Y., & Mao, Z. (2025). *ExpertPrompting: Instructing Large Language Models to be Distinguished Experts*. <http://arxiv.org/abs/2305.14688>
- Xuhui Zhao. (2019). *Automatic analysis for*.
- Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, & J. Ba. (2022). *Large language models are human-level prompt engineers*.
- Yong, G., Jeon, K., Gil, D., & Lee, G. (2023). Prompt engineering for zero-shot and few-shot defect detection and classification using a visual-language pretrained model. *Computer-Aided Civil and Infrastructure Engineering*, 38(11), 1536–1554. <https://doi.org/10.1111/mice.12954>
- Yu, X., Nong, S., He, D., Zheng, W., Ma, T., Liu, N., Li, J., & Xie, G. (2024). LogGenius: An Unsupervised Log Parsing Framework with Zero-shot Prompt Engineering. *Proceedings of the IEEE International Conference on Web Services, ICWS*, 1321–1328. <https://doi.org/10.1109/ICWS62655.2024.00159>
- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2020). BERTSCORE: EVALUATING TEXT GENERATION WITH BERT. *8th International Conference on Learning Representations, ICLR 2020*.
- Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and Benchmarks for Automated Log Parsing. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, 121–130. <https://doi.org/10.1109/ICSE-SEIP.2019.00021>

- Albers, P. C. H., & De Vries, H. (2001). Elo-rating as a tool in the sequential estimation of dominance strengths. In *Animal Behaviour* (Vol. 61, Issue 2, pp. 489–495). Academic Press. <https://doi.org/10.1006/anbe.2000.1571>
- Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In *IEEE Software* (Vol. 33, Issue 3). <https://doi.org/10.1109/MS.2016.64>
- Bradley, R. A., & Terry, M. E. (1952). Biometrika Trust Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. In *Source: Biometrika* (Vol. 39, Issue 3). <https://about.jstor.org/terms>
- C. Waite. (2008). *JUnit Common Auto Tester Framework User Guide*.
- Chen, M., Zhang, H., Wan, C., Wei, Z., Xu, Y., Wang, J., & Gu, X. (2023). *On the Effectiveness of Large Language Models in Domain-Specific Code Generation*. <https://doi.org/10.1109/QRS62785.2024.00030>
- Cinque, M., Della Corte, R., & Pecchia, A. (2019). Advancing monitoring in microservices systems. *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*. <https://doi.org/10.1109/ISSREW.2019.00060>
- Davoodijam, E., & Alambardar Meybodi, M. (2024). Evaluation metrics on text summarization: comprehensive survey. *Knowledge and Information Systems*. <https://doi.org/10.1007/s10115-024-02217-0>
- Enrique, O., & Casado, L. (n.d.). *Improving Industrial Maintenance: Leveraging ML and NLP for better Breakdown Log Analysis and Automated Workflows*. www.hv.se
- Fisher, J., Koning, D., & Ludwigsen, A. P. (2013). *Utilizing Atlassian Jira For Large-Scale Software Development Management**.
- Gopinath, A. M. (2024). *Root Cause Prediction from Log Data using Large Language Models*. www.liu.se
- Jamshidi, P., Pahl, C., Mendonca, N. C., Lewis, J., & Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. In *IEEE Software* (Vol. 35, Issue 3). <https://doi.org/10.1109/MS.2018.2141039>
- Karlsen, E., Luo, X., Zincir-Heywood, N., & Heywood, M. (2024). Large language models and unsupervised feature learning: implications for log analysis. *Annales Des Telecommunications/Annals of Telecommunications*. <https://doi.org/10.1007/s12243-024-01028-2>
- Lin, C.-Y. (2004). *ROUGE: A Package for Automatic Evaluation of Summaries*.
- Liu, H., & Wang, Y. (2023). *Towards Informative Few-Shot Prompt with Maximum Information Gain for In-Context Learning*. <http://arxiv.org/abs/2310.08923>
- Liu, Y., Tao, S., Meng, W., Wang, J., Ma, W., Chen, Y., Zhao, Y., Yang, H., & Jiang, Y. (2024). Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies. *IEEE International Conference on Program Comprehension*, 35–46. <https://doi.org/10.1145/3643916.3644408>
- Mann, B. (n.d.). *Language Models are Few-Shot Learners*.
- Mauro, T. (2015). Adopting Microservices at Netflix: Lessons for Architectural Design. In *Nginx Blog*.
- Nenkova, A., & McKeown, K. (2011). Automatic summarization. In *Foundations and Trends in Information Retrieval* (Vol. 5, Issues 2–3, pp. 103–233). <https://doi.org/10.1561/15000000015>
- Pang, Y., Zhang, M., Liu, Y., Li, X., Wang, Y., Huan, Y., Liu, Z., Li, J., & Wang, D. (2024). Large language model-based optical network log analysis using LLaMA2 with instruction tuning. *Journal of Optical Communications and Networking*, 16(11), 1116–1132. <https://doi.org/10.1364/JOCN.527874>

- Park, D., An, G. T., Kamyod, C., & Kim, C. G. (2023). A Study on Performance Improvement of Prompt Engineering for Generative AI with a Large Language Model. *Journal of Web Engineering*, 22(8), 1187–1206. <https://doi.org/10.13052/jwe1540-9589.2285>
- Rezaei, N., & Jabbari, P. (2022). Naïve Bayes’ classifiers in R. In *Immunoinformatics of Cancers* (pp. 71–85). Elsevier. <https://doi.org/10.1016/b978-0-12-822400-7.00010-5>
- Rodriguez, A. D., Dearstyne, K. R., & Cleland-Huang, J. (2023). Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability. *Proceedings - 31st IEEE International Requirements Engineering Conference Workshops, REW 2023*, 455–464. <https://doi.org/10.1109/REW57809.2023.00087>
- Sarika, P. K., Badampudi, D., Josyula, S. P., & Usman, M. (2023). Automating Microservices Test Failure Analysis using Kubernetes Cluster Logs. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3593434.3593472>
- Shan, S., Huo, Y., Su, Y., Li, Y., Li, D., & Zheng, Z. (2024). *Face It Yourself: An LLM-Based Two-Stage Strategy to Localize Configuration Errors via Logs*. <https://doi.org/10.1145/3650212.3652106>
- Usman, M., Badampudi, D., Smith, C., & Nayak, H. (2022). An Ecosystem for the Large-Scale Reuse of Microservices in a Cloud-Native Context. *IEEE Software*, 39(5). <https://doi.org/10.1109/MS.2022.3167447>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi Quoc, E. H., Le, V., & Zhou, D. (n.d.). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models Chain-of-Thought Prompting*.
- Xia, Q., Zhao, H., & Liu, M. (2024). Prompt Engineering Approach Study for Supervised Fine-Tuned (SFT) Large Language Models (LLMs) in Spacecraft Fault Diagnosis. *Proceedings of the 3rd Conference on Fully Actuated System Theory and Applications, FASTA 2024*, 819–824. <https://doi.org/10.1109/FASTA61401.2024.10595331>
- Xu, B., Yang, A., Lin, J., Wang, Q., Zhou, C., Zhang, Y., & Mao, Z. (2025). *ExpertPrompting: Instructing Large Language Models to be Distinguished Experts*. <http://arxiv.org/abs/2305.14688>
- Xuhui Zhao. (2019). *Automatic analysis for*.
- Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, & J. Ba. (2022). *Large language models are human-level prompt engineers*.
- Yong, G., Jeon, K., Gil, D., & Lee, G. (2023). Prompt engineering for zero-shot and few-shot defect detection and classification using a visual-language pretrained model. *Computer-Aided Civil and Infrastructure Engineering*, 38(11), 1536–1554. <https://doi.org/10.1111/mice.12954>
- Yu, X., Nong, S., He, D., Zheng, W., Ma, T., Liu, N., Li, J., & Xie, G. (2024). LogGenius: An Unsupervised Log Parsing Framework with Zero-shot Prompt Engineering. *Proceedings of the IEEE International Conference on Web Services, ICWS*, 1321–1328. <https://doi.org/10.1109/ICWS62655.2024.00159>
- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2020). BERTSCORE: EVALUATING TEXT GENERATION WITH BERT. *8th International Conference on Learning Representations, ICLR 2020*.
- Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and Benchmarks for Automated Log Parsing. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, 121–130. <https://doi.org/10.1109/ICSE-SEIP.2019.00021>

Appendix A - Practitioner Survey Protocol (Qualitative Evaluation)

Purpose

To elicit practitioner preferences among LLM-generated log summaries and assess clarity, completeness, accuracy, and readability for CI/CD diagnostic use.

Participants

Fifteen CI/CD practitioners (CICD developers; 2–12 years of experience, mean ≈ 6) with regular exposure to microservices logs and automated testing pipelines.

Materials (

- 4 log cases drawn from real CI/CD scenarios.
- For each case, 6 summaries produced by distinct prompt strategies (same underlying logs).

Procedure (what participants did)

1. Read both summaries

For each comparison, two summaries from the same case were shown side-by-side. Participants were instructed to read both carefully.

2. Select the better summary

Participants chose the option they considered better overall, considering clarity, completeness, accuracy, and readability.

3. Automatic progression

After each selection, the interface advanced to the next comparison and displayed brief visual confirmation of the choice.

4. Progress through all cases

Participants proceeded through 4 cases. For each case, the 6 summaries were compared using a sequential pairwise tournament (knockout) to minimize fatigue.

- Start summary was randomized per participant.
- Each case required 5 pairwise choices (total ≈ 20 comparisons per participant).

Measures (what was collected)

For each pairwise comparison:

- Primary outcome: which summary was preferred (A vs B).
- Secondary tags (implicit in instructions): perceived clarity, completeness, accuracy, readability (operationalized via the single overall choice).

Randomization & Fatigue Control

- Randomized starting summary per case and randomized pairing order.
- Sequential tournament design (5 comparisons per case) instead of full round-robin (15) to reduce cognitive load.

Analysis Plan

- Bradley–Terry model to estimate latent preference (“ability”) scores for each prompt strategy from pairwise outcomes.
- Elo rating to complement Bradley–Terry with order-sensitive, incremental scoring across the sequence of comparisons.

Data Handling & Ethics

- No personal identifiers collected beyond role and years of experience.
- Responses stored in aggregate form; individual choices not attributable to specific participants.
- Participation was voluntary; instructions and consent were presented on the first screen.

Survey Instrument (as shown to participants)

A. Instructions

You will see pairs of summaries for the same log case.

Please read both and select the better summary considering clarity, completeness, accuracy, readability.

After each choice, the next pair will appear automatically. A progress bar shows your status across 4 cases.

B. Pairwise Comparison Screen (per trial)

- Summary A (left panel)
- Summary B (right panel)
- Question: “Which summary is better for diagnosing this issue in your workflow?”
☐ Summary A ☐ Summary B
- [Continue] (auto-advance after selection, brief visual confirmation)

(Internally, pair order and initial summary are randomized; comparisons follow a sequential tournament until a case winner is determined.)

The table below shows an example of the different prompt techniques used for specific case and generate different summaries.

Prompting Technique	Prompt Snippet (structure only)
Baseline	Final Unified Summary Consolidated Failure Diagnosis: The system is experiencing multiple failures due to a combination of scaling issues, resource misconfiguration, and potential rollout blockers. The assertion failed error in the (eric-data-search-engine) service is a symptom of a larger problem, which

	includes frequent restarts, chunk flush retries, and connection timeouts across multiple containers. Most Affected Service or Component: The (eric-data-search-engine) service is the primary focus of the test case and appears to be the most affected component. This service is experiencing assertion failed errors, frequent restarts, and potential rollout blockers due to resource misconfiguration. Critical Point of Failure: The critical point of failure is likely related to resource misconfiguration, including unbound PersistentVolumeClaims (PVCs), insufficient resources, and incorrect node selectors or missing selectors. These issues are causing scheduling failures, container-level failures, and potential rollout blockers. Executive-Level Summary (3-5 sentences): The system is experiencing critical failures due to scaling issues and resource misconfiguration in the (eric-data-search-engine) service. This has resulted in assertion failed errors, frequent restarts, and potential rollout blockers. Immediate attention is required to investigate and address these issues, including reviewing PVC configuration, adjusting resource requests or limits, and implementing load balancing or scaling strategies. A comprehensive analysis of system performance and configurations is necessary to prevent similar failures in the future. Recommendations: 1. Investigate potential causes of connection timeouts or resource exhaustion (e.g., network congestion, insufficient resources). 2. Review PVC configuration and adjust resource requests or limits as needed. 3. Consider implementing load balancing, scaling, or optimizing resource allocation to mitigate these issues. 4. Monitor system performance and adjust configurations as needed to prevent similar failures in the future. Escalation Plan: The findings will be escalated to the relevant stakeholders for immediate attention and action. A comprehensive plan will be developed to address the identified issues, including reviewing PVC configuration, adjusting resource requests or limits, and implementing load balancing or scaling strategies. Regular progress updates will be provided to ensure timely resolution of these critical failures.
Reordered	Here are some key findings: 1. Certificate issues: Multiple logs indicate certificate-related problems, such as ""Certificate not available"" or ""Loading new PM CA certificate"". This suggests that there might be an issue with certificate management or availability. 2. Resource exhaustion: Some logs mention resource constraints (e.g., memory, disk space) that prevent successful data ingestion or processing. 3. Network connectivity issues: Problems with network connectivity between components of the system could lead to failed flush operations and retries. 4. Data corruption or inconsistencies: Errors in the data being ingested or processed might cause the application to retry or fail to complete tasks. To address these issues, I would recommend: 1. Monitoring system resources: Keep an eye on resource usage (e.g., memory, CPU) and network performance to identify potential bottlenecks. 2. Log analysis: Use log analysis tools to extract relevant information from the logs, such as error frequencies, retry counts, and timestamped events. 3. Error tracking: Implement a dedicated error tracking system to capture detailed error messages and stack traces for each failure. 4. System checks: Perform regular health checks on the application, its dependencies, and underlying infrastructure to identify potential issues.

	Additionally, it's essential to investigate the root causes of these problems and take corrective actions to improve the stability and performance of your application. Here are some specific recommendations: • For certificate-related issues: ○ Verify that certificates are properly configured and available. ○ Check for any certificate expiration dates or revocation status. ○ Consider implementing a certificate rotation policy. • For resource exhaustion: ○ Monitor system resources (e.g., memory, CPU) to identify potential bottlenecks. ○ Optimize application performance by reducing memory usage or leveraging parallel processing capabilities. ○ Implement a load balancing strategy to distribute workload across multiple nodes. • For network connectivity issues: ○ Verify that network configurations are correct and up-to-date. ○ Check for any firewall rules or access control lists (ACLs) that might be blocking communication between components. ○ Consider implementing a network monitoring tool to detect potential issues. By following these recommendations, you should be able to better understand the root causes of these errors and take corrective actions to improve the stability and performance of your application." "This is a large log file with multiple entries from different containers and services. I'll try to summarize the key points: Container 1: eric-data-search-engine-ingest-tls-0 • The container is experiencing issues with chunk flushing or processing large amounts of data. • There are many retries (e.g., retry 2, retry 3) indicating performance issues or resource constraints. • Possible causes: ○ Resource exhaustion (memory, disk space) ○ Network connectivity issues ○ Data corruption or inconsistencies Container 2: eric-data-search-engine-feoperator-8d5d45fb4-qtxql • The container is an OpenSearch cluster being managed by a controller. • The log shows various resource creation events (e.g., creating Secrets, ConfigMaps). • There are two instances of ""Start Reconcile"" messages indicating that the OpenSearch cluster is undergoing reconciliation. Container 3: eric-data-search-engine-ingest-tls-0 • The container is experiencing certificate-related issues. • A critical error message indicates that a certificate is not available at (/run/secrets/rhsm). • Possible causes: ○ Certificate configuration issues ○ Network connectivity problems Common Issues • Timezone configuration warnings (e.g., ""The configured timezone does not exist in the host, using UTC instead."" • Certificate-related issues (e.g., certificate not available, loading new PM CA certificate)
Reframed Role	Here are some key points that can be extracted from this output: 1. Container Logs: The output contains logs from multiple containers, including (eric-data-search-engine-ingest-tls-0), (feoperator), and others. 2. Error Messages: There are several error messages scattered throughout the log entries, indicating issues with certificate management, resource constraints, network connectivity, and application configuration.

	3. Recurring Errors: Some logs show recurring errors related to chunk flushing failures, which might be due to resource bottlenecks or temporary connectivity issues. 4. Retry Mechanism: Many log entries indicate retry attempts with increasing attempt numbers, suggesting that the system is designed to retry flushing chunks when they fail initially. To troubleshoot these issues, I would recommend: 1. Monitoring System Resources: Check if there are any resource bottlenecks (CPU, memory, disk space, network bandwidth) that could be causing the chunk flushing failures. 2. Network Connectivity Checks: Verify that the application has stable network connectivity to ensure that data is being transmitted correctly. 3. Application Configuration Review: Review the application's configuration settings to ensure that they are set up correctly for chunk processing and flushing. 4. Error Analysis: Investigate the root cause of recurring errors, such as temporary issues with disk space or network congestion. Some specific recommendations based on the log entries include: • Verifying secret configuration and certificate path in (eric-data-search-engine-ingest-tls-0) container • Checking resource utilization (CPU, memory, disk space) in (feoperator) container • Reviewing application configuration settings for chunk processing and flushing
Minimal Instructions	Findings: Assertion failed: Failed to kick-in scale operation for operand eric-data-search-engine, Root Cause: Failed to scale in ReplicaSet using kubectrl commands, overall_summary: Test case TC_SCALE_MIN_002 failed due to scaling issue with eric-data-search-engine operand, Action Plan: Investigate and resolve the scaling issue with the eric-data-search-engine operand
Few Shot	Critical Issue: • The MinIO server process in the POD 'eric-data-object-storage-mn-0' was killed due to an out-of-memory (OOM) situation. Other Issues: • The POD 'eric-data-object-storage-mn-0' is not ready, with Condition [Ready] set to False. • STATEFULSET 'eric-data-object-storage-mn' has DESIRED [1] and READY [0] replicas. • Upgrade process for the SpiderApp application did not complete successfully. Deployment Logs Summary: • No logs or data were found in the provided JSON, resulting in a (null) value. • Potential issue with log collection or deployment failure. Pod Logs Summary: • Service Name: eric-data-object-storage-mn • Issue Identified: The MinIO server process was killed due to an out-of-memory (OOM) situation. • Error Message: ""Killed"" in the final log line indicates memory exhaustion. • Context: MinIO memory limit is set to 90% of available memory, which may be insufficient for the workload. Recommendations: 1. Immediate attention is required to address the OOM issue and ensure all resources are up and running. 2. Further investigation into log collection and memory allocation is recommended.

	- Review the MinIO documentation to understand the recommended configuration and settings for the KES component. - Monitor system resources (e.g., CPU, memory) to identify any bottlenecks or issues. Overall, it appears that there are some critical issues with the MinIO server process being killed due to OOM, which is impacting the overall performance of the application. It's essential to address this issue promptly and investigate further to prevent similar incidents in the future.
Chain of Thoughts	Final Unified Summary Consolidated Failure Diagnosis: The system is experiencing multiple failures due to a combination of scaling issues, resource misconfiguration, and potential rollout blockers. The assertion failed error in the (eric-data-search-engine) service is a symptom of a larger problem, which includes frequent restarts, chunk flush retries, and connection timeouts across multiple containers. Most Affected Service or Component: The (eric-data-search-engine) service is the primary focus of the test case and appears to be the most affected component. This service is experiencing assertion failed errors, frequent restarts, and potential rollout blockers due to resource misconfiguration. Critical Point of Failure: The critical point of failure is likely related to resource misconfiguration, including unbound PersistentVolumeClaims (PVCs), insufficient resources, and incorrect node selectors or missing selectors. These issues are causing scheduling failures, container-level failures, and potential rollout blockers. Executive-Level Summary (3-5 sentences): The system is experiencing critical failures due to scaling issues and resource misconfiguration in the (eric-data-search-engine) service. This has resulted in assertion failed errors, frequent restarts, and potential rollout blockers. Immediate attention is required to investigate and address these issues, including reviewing PVC configuration, adjusting resource requests or limits, and implementing load balancing or scaling strategies. A comprehensive analysis of system performance and configurations is necessary to prevent similar failures in the future. Recommendations: - Investigate potential causes of connection timeouts or resource exhaustion (e.g., network congestion, insufficient resources). - Review PVC configuration and adjust resource requests or limits as needed. - Consider implementing load balancing, scaling, or optimizing resource allocation to mitigate these issues. - Monitor system performance and adjust configurations as needed to prevent similar failures in the future. Escalation Plan: The findings will be escalated to the relevant stakeholders for immediate attention and action. A comprehensive plan will be developed to address the identified issues, including reviewing PVC configuration, adjusting resource requests or limits, and implementing load balancing or scaling strategies. Regular progress updates will be provided to ensure timely resolution of these critical failures.

Appendix B: Illustrative Example of Microservices Log Data

Because the logs used in this study originate from Ericsson’s internal systems and are therefore confidential, this appendix provides an illustrative example of microservices log data adapted from the LogHub benchmark dataset (Zhu et al., 2019). LogHub contains publicly available logs from large-scale systems such as Hadoop, Spark, and Kubernetes. These datasets demonstrate similar complexity to industrial logs, including high volume, redundancy, and a mixture of structured and unstructured information.

B.1 Example Log Snippet (adapted from LogHub)

```
2024-01-18 12:45:32,456 INFO [ServiceA] Starting container for deployment task ID=depl_9342
2024-01-18 12:45:33,102 WARN [ServiceB] Connection timeout detected for pod=traffic-mgr-21
2024-01-18 12:45:35,781 ERROR [ServiceC] NullPointerException at ModuleX.java:87 - Stack trace follows
2024-01-18 12:45:35,782 INFO [ServiceC] Attempting automatic restart of failed module
2024-01-18 12:45:36,110 ERROR [ServiceD] Resource allocation failed: insufficient memory on node=node-14
```

B.2 Characteristics of Microservices Logs

This example highlights several typical features of microservices log data:

- High concurrency: multiple services (ServiceA, ServiceB, ServiceC, ServiceD) record messages simultaneously.
- Mixed log levels: informational (INFO), warnings (WARN), and errors (ERROR) appear within short time intervals.
- Structured elements: timestamps, service identifiers, and error codes.
- Unstructured elements: stack traces, descriptive error messages, and recovery attempts.
- Cross-service dependencies: one service’s failure (e.g., memory allocation) can affect others, complicating root cause analysis.