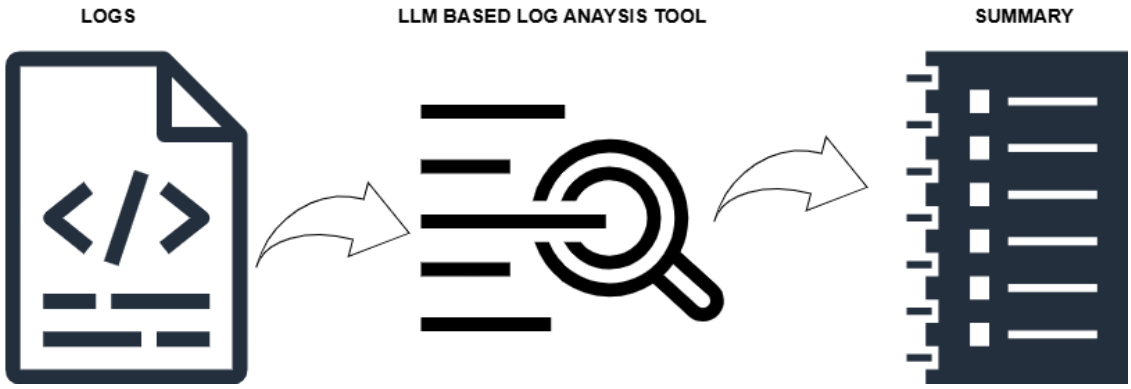
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



LLM-based Log Analysis

Evaluating LLM-Based Log Analysis in Industrial CI Environments: Technical Metrics, Practitioner Perspectives and Perceived Workflow Impact

Master's Thesis in Computer Science and Engineering

Tesfu Tekleab Hailu
Pallavi Gupta

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

LLM-based Log Analysis

Evaluating LLM-Based Log Analysis in Industrial CI Environments: Technical Metrics, Practitioner Perspectives and Perceived Workflow Impact

Tesfu Tekleab Hailu
Pallavi Gupta

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Evaluating LLM-Based Log Analysis in Industrial CI Environments: Technical Metrics, Practitioner Perspectives and Perceived Workflow Impact

Tesfu Tekleab Hailu, Pallavi Gupta

Supervisor: Philipp Leitner, Department of Computer Science and Engineering
Industrial Supervisor 1: Fredrik Johnsson, Robert Bosch AB
Industrial Supervisor 2: Isidro Calvo, Robert Bosch AB
Examiner: Farnaz Fotrousi, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Tesfu Tekleab Hailu, Pallavi Gupta
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Continuous Integration (CI) pipelines generate large volumes of complex logs that developers must analyze to diagnose failures and identify root causes. As software systems become increasingly distributed, manual log analysis has become time-consuming, cognitively demanding, and highly dependent on practitioners' expertise. Recent advances in Large Language Models (LLMs) have created new opportunities for automated log analysis and debugging support. However, there remains a limited understanding of how LLM-based log analysis tools should be evaluated and how practitioners perceive their usefulness and influence on industrial CI environments.

This thesis addresses that gap through an industrial case study conducted in collaboration with Bosch. The study follows a mixed-methods research design that combines qualitative and quantitative approaches, including a systematic literature review, technical evaluation, descriptive statistical analysis, and correlation analyses using Spearman's rank correlation and Kendall's tau.

The findings show that conventional text-overlap metrics, such as BLEU and ROUGE, are not suitable for evaluating log analysis tool outputs because they require a reference and cannot handle the reasoning-intensive nature of CI log analysis. LLM-as-a-judge metrics, specifically G-Eval and GPTScore, were identified as more appropriate because they support reference-free and reasoning-aware evaluation. However, the comparison between automated and practitioner evaluations revealed only weak alignment and limited discriminative power in automated scoring, indicating that automated evaluation cannot reliably replace human judgment. Practitioners were found to evaluate outputs based on precise root-cause identification and whether the outputs support debugging. The study further shows that practitioners perceive the LLM-based tool as useful for supporting CI troubleshooting by accelerating failure investigation, reducing manual log inspection, and providing interactive support, particularly for complex and unfamiliar failures. It also highlights risks related to hallucinations, over-reliance, and reduced critical reasoning.

Finally, practitioners perceived the integration of an MCP-based agent into the development environment as useful for supporting workflow continuity by reducing context switching and enabling iterative conversational debugging. However, they viewed it as complementary to, rather than a replacement for, the standalone web-based tool. Together, the findings contribute to evaluation methodologies and empirical insights into how LLM-based log analysis tools can be assessed and considered for adoption in industrial CI workflows.

Keywords: Large Language Models, LLM-based Log Analysis, Human-Centered Metrics, LLM-as-a-Judge, Root Cause Analysis, MCP-based Agent, CI Workflows

Acknowledgements

First and foremost, we would like to express our sincere gratitude to our academic supervisor, **Philipp Leitner**, for his continuous guidance, thoughtful feedback, and unwavering support throughout this thesis. His insights and encouragement have been invaluable in shaping both the direction and quality of this work.

We would also like to thank our industrial supervisors, **Fredrik Johnsson** and **Isidro Calvo**, for their collaboration, support, and practical guidance during this project. Their domain expertise and industrial perspectives greatly contributed to our understanding of the challenges of log analysis within real-world CI environments.

In addition, we would like to extend our sincere appreciation to our examiner, **Farnaz Fotrousi**, for the constructive feedback and valuable comments provided during the proposal approval and thesis midterm presentation. Her suggestions helped us refine and strengthen this research throughout its development.

We are especially grateful to the fourteen practitioners who participated in this study. Their willingness to dedicate their time, share their experiences, participate in interviews, and help evaluate log analysis results formed the foundation of this research. Their contributions were essential in making this study possible.

Finally, we would like to thank our family and friends for their constant encouragement, patience, and support throughout this journey. Their motivation helped us remain focused and complete this work successfully.

Tesfu Tekleab Hailu, Pallavi Gupta, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

Modern software systems have become increasingly complex, distributed, and continuously evolving. To maintain reliability and rapid development cycles, many organizations rely on Continuous Integration (CI) pipelines that automatically build, test, and validate software after each code change. These pipelines generate extensive execution logs that record system performance, compilation results, test outcomes, and runtime errors. Such logs are essential for diagnosing failures and understanding system anomalies, making log analysis a critical activity in software maintenance and debugging workflows [1, 2]. However, as software systems scale and CI pipelines execute thousands of builds per day, the volume and diversity of generated logs has grown substantially, making manual analysis increasingly difficult for developers.

Traditionally, automated log analysis has relied on rule-based techniques, pattern matching, and statistical models. Early approaches focused on extracting structured templates from logs and applying heuristic rules to identify abnormal patterns [1]. More recent methods introduced machine learning techniques that model log sequences to detect anomalies or predict failures, such as DeepLog [2] and LogAnomaly [3]. While these methods have improved the ability to detect deviations from normal system performance, they typically depend on predefined log structures, require training data, and often struggle to adapt to evolving log formats and system architectures. Furthermore, many existing approaches focus primarily on anomaly detection rather than providing developers with interpretable explanations or actionable debugging insights.

Recent advances in Large Language Models (LLMs) have created new opportunities for addressing these limitations. LLMs have demonstrated strong capabilities in understanding natural language, synthesizing information, and generating structured explanations across a wide range of domains, including engineering [4], medicine [5], and the social sciences [6]. These capabilities have encouraged researchers and practitioners to explore LLMs not only as text generation tools but also as systems capable of supporting analytical and reasoning-intensive tasks. For example, LLMs are increasingly used in code generation [7], software debugging, and documentation analysis [8], where they assist developers in understanding complex artifacts and reasoning about system behavior.

Within the context of software engineering, the ability of LLMs to interpret textual information makes them particularly promising for log analysis tasks. Logs are largely composed of semi-structured textual messages that describe system events, errors, and state transitions. By leveraging contextual understanding, LLMs can potentially summarize large log files, identify relevant error messages, and generate explanations that support root cause analysis. Recent research has begun exploring these possibilities. Studies have shown that LLMs can assist in tasks such as log summarization, anomaly detection, and automated debugging support [9]. Applied research prototypes, such as LLMLogAnalyzer [10], further demonstrate the feasibility of integrating LLMs into interactive log analysis workflows in industrial settings.

Despite these promising developments, the deployment of LLM-based log analysis systems introduces several challenges. One key concern is the reliability of LLM outputs, as models may produce hallucinated explanations or incorrect reasoning when interpreting complex system logs [11]. In addition, the computational cost and latency associated with large models may limit their scalability in high-throughput CI environments. Furthermore, there are currently no widely accepted methodologies for evaluating the usefulness and reliability of LLM-generated log analysis outputs. Traditional automated metrics, such as BLEU [12] and ROUGE [13], rely on n-gram comparisons and are not designed to evaluate the quality of reasoning, clarity of explanations, or practical debugging value. On the other hand, qualitative evaluations provided by experts, while richer, are difficult to scale across large volumes of logs. This creates a significant evaluation gap when attempting to assess LLM-supported analytical systems.

Industrial environments further highlight the need for practical and scalable log analysis solutions. This thesis is conducted in collaboration with the Bosch company, where software teams rely heavily on CI pipelines to build and validate complex embedded and software systems. Failures occurring during CI builds often produce extensive logs that developers must inspect to identify the root cause of errors. To support this process, Bosch has developed the *Log Analyzer*, a hybrid log analysis tool designed specifically for CI environments.

The Log Analyzer combines multiple complementary analysis pipelines. The first applies a Binary Keywords method, which performs fast and transparent keyword-based matching to detect known patterns in log data. The second applies Regular Expression filtering to capture predefined textual patterns. The third uses Neural Search, which relies on vector embeddings and similarity search within a Qdrant database to capture semantic relationships between log messages and previously observed patterns. The neural component models known “good” logs and identifies deviations without requiring retraining, enabling the system to adapt to evolving log patterns.

Outputs from these analysis pipelines are provided as structured inputs to Large Language Models, which generate summaries, root-cause hypotheses, and debugging guidance for developers. By combining traditional pattern matching, embedding-

based retrieval, and LLM-based reasoning, the system aims to balance efficiency, interpretability, and semantic understanding. Importantly, this architecture limits the LLM's direct exposure to raw log data while still leveraging its ability to synthesize contextual explanations.

As Bosch plans to integrate the Log Analyzer into CI platforms such as Jenkins, GitLab, Gerrit, and Azure DevOps, it becomes necessary to systematically characterize how the system behaves in real industrial CI environments. In particular, several questions arise: how developers evaluate the usefulness of LLM-generated log analysis outputs, how configuration settings influence the outputs and practical performance of the Log Analyzer, and how the system architecture could be extended to support more advanced debugging workflows.

The central problem addressed in this thesis is the lack of a systematic and empirically grounded understanding of how LLM-based log analysis tools should be evaluated and how practitioners perceive their usefulness in industrial CI environments. This includes four related challenges: identifying meaningful technical and human-centered criteria for evaluating tool output, understanding how automated LLM-based evaluations compare with practitioner judgments, examining how practitioners perceive the usefulness, risks, and workflow fit of an LLM-based log analysis tool, and understanding how MCP-based IDE integration is perceived in comparison with a standalone web-based log analysis tool. Addressing these challenges is important for enabling more reliable evaluation, improving tool design, and supporting the trustworthy adoption of LLM-based assistance in software engineering practice.

1.1 Research Questions

RQ1: What technical and human-centered evaluation criteria or metrics are used to evaluate an LLM-based log analysis tool output in an industrial CI context?

- **RQ1.1:** What technical criteria or metrics are relevant for evaluating LLM-based log analysis tool output on industrial build logs?
- **RQ1.2:** What human-centered criteria or metrics are used by CI practitioners to evaluate LLM-based log analysis outputs?

RQ2: How is the output of an LLM-based log analysis tool evaluated using technical metrics in practice, and to what extent do automated evaluation metrics align with human judgments when assessing the quality of outputs?

RQ3: How do CI practitioners perceive the usefulness and influence of an LLM-based log analysis tool on CI workflows?

RQ4: How do CI practitioners perceive the usefulness and influence of integrating an MCP-based agent into the development environment on CI workflows, compared to a standalone web-based log analysis tool?

The first research question (**RQ1**) investigates which evaluation criteria are appro-

priate for assessing the outputs of LLM-based log analysis tools in an industrial CI context. It focuses on both technical dimensions and human-centered aspects, reflecting how practitioners judge the usefulness of generated analyses. Specifically, **RQ1.1** addresses which technical evaluation criteria or metrics are available to assess the LLM-based log analysis tool's output, and **RQ1.2** examines the human-centered criteria practitioners use to evaluate the tool's output.

To further validate the applicability of these criteria, **RQ2** examines how the tool's output is evaluated in practice, and the extent to which automated evaluation metrics align with practitioner-based human judgments. To address this, the identified technical metrics are operationalized into an automated evaluation framework implemented in Python and applied to assess the outputs of the LLM-based log analysis tool. The resulting metric scores are analyzed alongside practitioner evaluations using descriptive statistics, followed by Spearman and Kendall rank correlation analyses to determine the level of agreement between automated and human assessments.

The third research question (**RQ3**) examines how CI practitioners perceive the usefulness and influence of an LLM-based log analysis tool on CI workflows. It aims to understand how practitioners perceive the influence of these tools on the efficiency of debugging processes and decision-making when diagnosing and resolving issues in logs. Additionally, the study further explores potential unintended side effects that practitioners perceive the use of such tools may introduce into CI workflows.

The fourth research question (**RQ4**) investigates how CI practitioners perceive the usefulness and influence of integrating an MCP-based agent into the development environment, compared to a standalone web-based log analysis tool, for CI troubleshooting. Specifically, it examines practitioners' perceptions of workflow continuity, interaction patterns, tool usage, and the transition between log analysis and code modification within industrial CI practices.

1.2 Thesis Contributions

This thesis contributes to the emerging area of LLM-based log analysis in industrial CI environments in six ways. First, it identifies technical evaluation metrics to evaluate the output quality of LLM-based log analysis tools applied to industrial CI logs. Second, it identifies the human-centered criteria that CI practitioners use to judge tool output. Third, it designs evaluation prompts and develops a practical evaluation framework for the structured assessment of LLM-based log analysis outputs against both technical and practitioner-oriented criteria. Fourth, it provides empirical evidence on the degree of alignment between automated technical metrics and practitioner-based evaluations. Fifth, it identifies how CI practitioners perceive the usefulness and influence of an LLM-based log analysis tool on CI workflows, capturing both perceived efficiency gains and associated risks. Sixth, it identifies how CI practitioners perceive the usefulness and influence of integrating an MCP-based agent into the development environment compared to a standalone web-based tool, indicating that the two approaches serve complementary roles.

2

Background

This chapter outlines the key concepts relevant to the study. It begins by introducing log analysis, root cause analysis and log summarization. It then proceeds to explain large language models, prompts, and token limits. Finally, it presents the LLM-based log analysis tool used by Bosch, along with its analysis methods and internal workflow.

2.1 Log Analysis

The analysis of logs is a critical step in locating, detecting, and investigating faults in software systems [1, 2, 14]. To enable automated log mining, semi-structured logs are typically first converted to structured formats via log parsing [15, 14]. In many cases, pre-processing is performed alongside parsing, serving to remove irrelevant information, normalize variables, or otherwise transform the log contents. Once logs are structured, a variety of approaches can be applied to mine for insights, including statistical methods [1], clustering techniques [10], and machine learning methods [14, 16, 17, 18]. Building upon these foundations, more recent advances in log analysis have incorporated LLMs, leveraging prompt-based strategies [19], vector databases [20], and hybrid pipelines that combine pre-processing, retrieval, and reasoning [9, 21].

2.1.1 Root Cause Analysis

Root cause analysis (RCA) in software systems involves systematically examining logs to identify the underlying causes of failures or anomalies. Traditionally, RCA has relied on statistical analysis, rule-based methods, or machine learning techniques applied to structured log data [1, 2, 14]. Recently, however, the emergence of LLMs has enabled RCA to leverage natural language understanding to interpret complex log sequences, summarize relevant events, and suggest potential causes [9, 18]. Building on these advances, hybrid approaches combining preprocessing, retrieval, and LLM reasoning have shown promise, enabling more interactive and iterative debugging workflows. These methods provide developers with interpretable explanations and help reduce the time needed to pinpoint failures [19, 17].

2.1.2 Log Summarization

Log summarization condenses log data to highlight key information. This makes it easier for engineers and analysts to understand and act on system behavior. Logs can be extensive and cognitively demanding; summarization extracts insights without manually parsing every log message [22]. This is especially useful in large, complex systems, as summarization streamlines the identification and localization of faults.

2.2 Large Language Models

LLMs are generative Artificial Intelligence (AI) models that produce natural-language text by predicting the probability of the next token in a sequence of words. A token is a basic unit of text that can be as small as a single character or as large as a word or subword, depending on the tokenization method used. Tokens are the building blocks that the model processes to understand and generate text. LLMs are trained on large datasets of books, news articles, forums, blogs, and other textual content, often sourced from the internet. A user can ask an LLM a question and converse with it using different prompts. Due to the large amount of source code available on websites such as GitHub[1], LLMs can also assist with software engineering tasks, such as program repair [23] or fault localization [24].

2.2.1 Token Limit

The context window of an LLM is the number of tokens that the LLM can hold in memory at any given moment [22]. In a conversational model, the entire conversation must fit within the LLM's context window for it to remember the conversation. The same principle applies to log summarization: the entire log file must fit within the LLM's token limit. Otherwise, the model may miss important information in the file. Additionally, the LLM's ability to retain information tends to degrade near the token limit, leading to worse performance and more hallucinations [25]. Since the emergence of transformer-based models [26], LLMs have become increasingly capable of handling large volumes of data. Models such as Gemini 2.5 Flash[2] have a context window of 1,048,576 ($\sim$ 1M) tokens and are capable of generating solutions to software development problems with the ability to reason about complex topics [27].

2.2.2 Prompt Design

Prompts are often classified as either zero-shot or few-shot prompts [28, 19]. Prompts that are classified as zero-shot only contain instructions on how to produce the expected output, but do not provide any examples. Few-shot prompts, however, include examples of the input data and the appropriate output. Chain-of-thought (CoT) refers to prompting the model with intermediate reasoning steps. Few-shot

[1] `https://github.com`

[2] `https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash`

CoT involves providing the model with examples of step-by-step reasoning. Zero-shot CoT may simply be adding "Let us think step by step" to the prompt, which has been shown to improve the results on several benchmarks [28]. In-context learning refers to providing the LLM with information in the prompt to help the model produce better outputs.

2.3 LLM-based Log Analysis at Bosch

Bosch recently implemented an LLM-based log analysis tool that accepts log files as input and generates error names, detailed problem descriptions, suggested fix solutions, and the main root cause line numbers. The tool utilizes GPT-4o-mini (version 2024-08-01-preview) by OpenAI and Gemini-2.0-Flash-Lite to summarize output data. As illustrated in Figure 2.2, the workflow begins when a user uploads a log file to summarize through the web application. The user can select analysis methods (keywords, regular expressions, or neural search) and optionally apply filters (adding or removing keywords, false positive keywords, regular expressions, false positive regular expressions, or training the neural search tool). In addition, users can add context prompts (customized instructions) for the LLM, such as a brief description of the error or keywords to filter messages. Upon clicking "Analyze," the user waits for the tool to finish processing and analyzing the log file. The tool then displays a structured summary that comprises error names, detailed problem descriptions, suggested solutions, conclusions, and root cause lines. Furthermore, the tool generates an HTML visualization of the log file with hover functionality for root causes, errors, and information lines, enabling developers to easily locate errors. An example of the summary output is presented in Figure 2.1.

Error Names: ETIMEDOUT, Module build failed, OutOfMemory, Frontend build failed, Packaging failed, Build failed.
Description: The log indicates multiple critical errors. A network timeout (ETIMEDOUT) occurred during an npm package installation, likely preventing the build from proceeding correctly. Several "Module build failed" errors, likely related to PostCSS, suggest issues with the build process. An "OutOfMemory" error further indicates resource exhaustion during the build. The frontend build subsequently failed, and the packaging process also failed, ultimately leading to an overall build failure. The presence of a warning about a deprecated package and a missing frontend build artifact adds to the issues. The test suite reported one failure, which was later re-run and passed, indicating a potential flakiness.
Solutions: Address the network timeout by verifying network connectivity and the availability of the npm registry. Investigate the "Module build failed" errors by examining the PostCSS configuration and dependencies, and ensuring they are compatible. Optimize the build process to prevent "OutOfMemory" errors, potentially by increasing memory allocation, optimizing build steps, or splitting the build into smaller tasks. Correct the frontend build issues by examining the build configuration and dependencies. Ensure the frontend build artifact is generated before packaging. Review and update the deprecated package.

Notes: The log contains several warnings, including a deprecated package and a skipped packaging step due to a missing artifact. These warnings, while not directly causing the failure, should be addressed to improve the project's overall stability and maintainability. The test suite flakiness should be investigated to ensure reliable test results. The build process appears to have multiple failure points, indicating a need for a thorough review of the build configuration and dependencies.
Conclusion: The log indicates a significant build failure due to network issues, resource exhaustion, and build process errors. The combination of these errors prevents the build process from completing successfully.
Root Cause Lines: 35, 93, 108

Figure 2.1: Example of a summary generated by the LLM-based log analysis tool.

2.3.1 Internal Workflow

To begin with, the log analysis tool first accepts a log file and pre-processes it using one of the analysis methods discussed in Section 2.3.2. After pre-processing, the LLM receives the filtered log data along with a prompt. The LLM accepts only preprocessed log data, not the entire log, analyzes it, and provides a summary to the user according to the structure defined in the prompt. The data flow of the tool is illustrated in Figure 2.2, which shows the sequence of steps it follows.

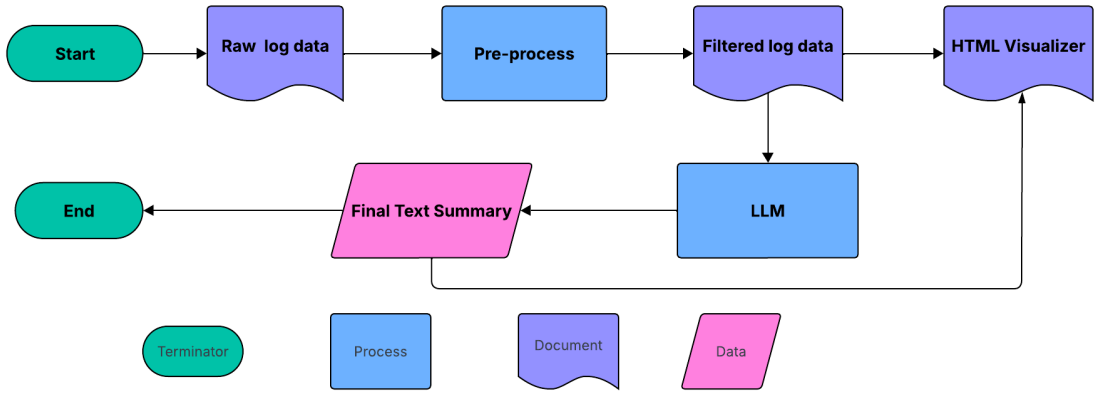


Figure 2.2: Overview of the internal workflow of the LLM-based log analysis tool.

2.3.2 Analysis Methods

1. Keywords Search: This method employs a straightforward binary keyword matching algorithm. It identifies relevant entries within log files by comparing each line against a predefined vocabulary of keywords. This approach, while simple, proves highly effective and valuable in specific use-case scenarios. The detailed process flow is presented in Figure 2.3.
2. Regular Expression Search: This method incorporates a regular expression matching capability, wherein individual lines from log files are evaluated against

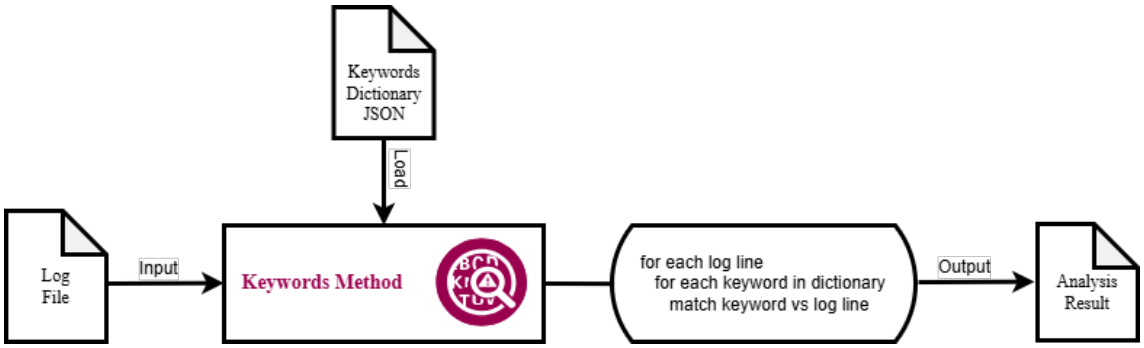


Figure 2.3: Binary Keyword Matching for log entry identification using a predefined keyword dictionary.

a comprehensive and user-defined list of regular expressions to achieve granular pattern detection. The detailed process flow is presented in Figure 2.4.

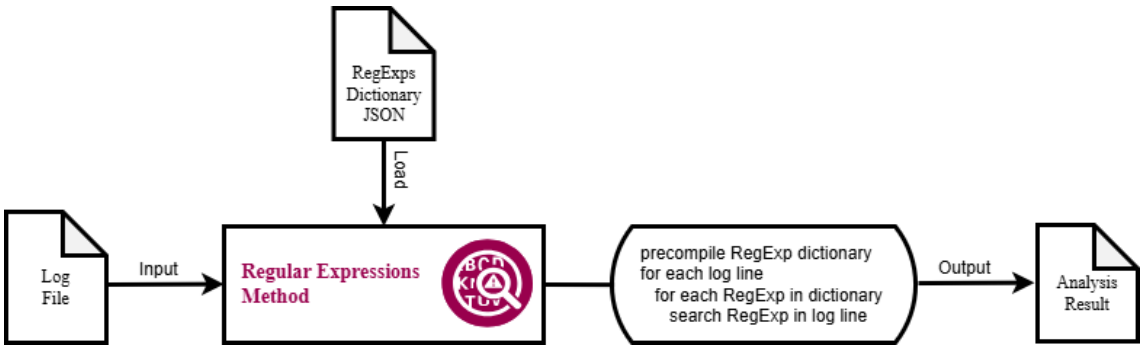


Figure 2.4: Regular Expression Matching for log entry identification using predefined pattern rules.

3. Neural Search: This advanced technique leverages a multi-dimensional vector space to represent collections of known good log lines, learned from extensive training data. It builds upon the principles outlined by Lashkari et al. [29], which demonstrate how neural embeddings can encode semantic relationships and construct unified, efficient search indices for large document collections. Similarly, in log analysis, the model uses neural search and similarity scores to identify anomalies in incoming log lines by comparing their embeddings to a vector representation of previously observed "good" logs. A key advantage of this approach is its integration with a dynamic vector database (e.g., Qdrant), allowing on-the-fly adjustments and adaptive pattern updates informed by user input, which increases flexibility and reduces maintenance costs compared to traditional trained models. As with semantic search indices, proper training on high-quality data is crucial: the model must be trained on clean log samples to ensure accurate identification of anomalies, and contamination with "bad" logs may degrade performance, requiring restoration of the vector database. The more thoroughly the model is trained on normal logs, the more proficient it becomes at detecting subtle errors and anomalous patterns, mirroring the benefits reported for neural embedding-based indices in document retrieval [29]. The detailed process flow is presented in Figure 2.5.

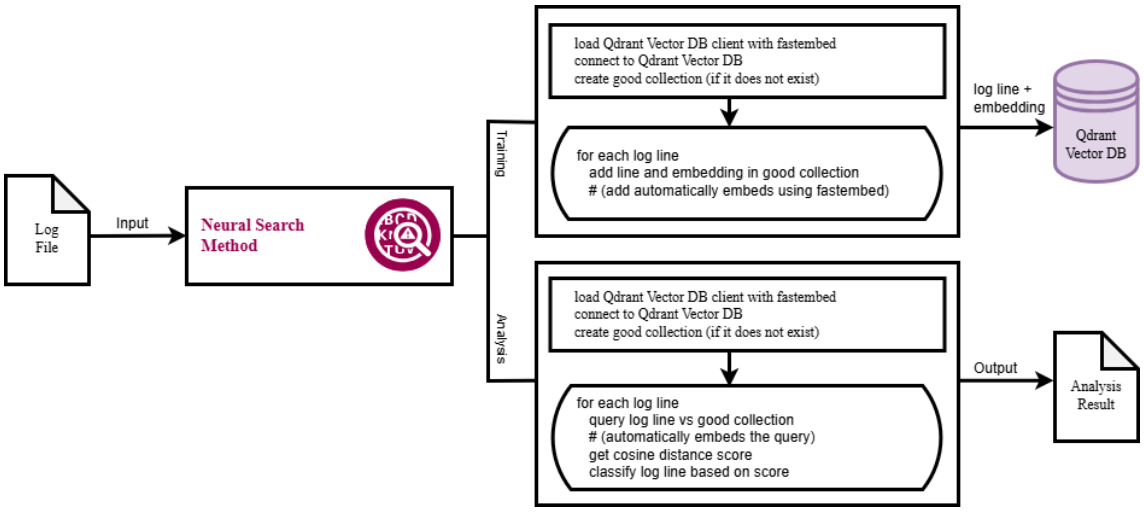


Figure 2.5: Neural Search using a vector database for semantic log entry identification based on contextual similarity.

2.3.3 Prompts

The prompt follows a zero-shot prompting approach, since it provides an example of the formatting and some instructions, but does not provide examples of appropriate responses to example inputs. The prompt template begins with general instructions and context, which is followed by the format for the output response. Finally, there is a list of precise instructions before giving the input data at the end.

INSTRUCTIONS:
Role:
- You are an expert log analyzer automation tool.
- Analyze the given log file extracts (never mention or comment on "extracts") as if it were a 'full log file' and generate plain text without any tags, formatting, or lists.
- Treat the log file extracts as plain text with line numbers as keys and log strings as values.
- A line containing 'Finished:' cannot be a root cause.
- DO NOT mention that the analysis is based on 'log extracts' or anything about 'extracts'.
- When unsure, suggest a thorough system and code examination.
- DO NOT comment on JSON structure or keys.
- DO NOT mention "success_lines" or '"success_lines":' in the output.
- 'Root cause' means the most critical errors that directly cause the system to fail or stop functioning.
- If "success_lines" is empty (e.g., [] or), the log file is most likely a failure.
- If "success_lines" is not empty (e.g., ...] or [...]] or [...]) and contains line numbers, and the text log file could be successful, but not necessarily.
- Analyze and decide if the log file is a failure or a success.

- In case of failures, generate PLAIN TEXT only 4-5 one-liner paragraphs delimited by two new lines as follows, in EXACTLY this order:
‘**Error Names:** ..., ...‘ (name of the most important error(s), a string with a title-like length ..)
‘**Description:** ..., ...‘ (brief description of the errors and causes, min-maximum 1200 characters..)
‘**Solutions:** ..., ...‘ (proposed solutions to fix the errors, min-maximum 1200 characters..)
‘**Notes:** ..., ...‘ (optional paragraph, additional notes, min-maximum 1200 characters..)
‘**Conclusion:** ..., ...‘ (optional paragraph, conclusion, min-maximum 1200 characters..)
- After all paragraphs, add an additional PLAIN TEXT one-liner paragraph:
’**Root Cause Lines**: ..., ...’ (ideally 1, maximum of 3 root cause number lines, only numeric line numbers, listing unique line number(s) of the main root cause ..)
- Do not add any comments or text outside 4-5 labeled paragraphs.
- Use only plain text, DO NOT use numbered lists or ordered lists or any kind of formatting.

- In the case of an overall successful outcome:
- Briefly comment on any errors or potential fixes as warnings.
- Write one single paragraph stating that the logs are most likely successful:
’..’ (details, descriptions, notes, etc.)
LOG FILE EXTRACTS: <FILTERED LOG FILES CHUNK>

Figure 2.6: Zero-shot prompt template defining instructions, roles, and output structure for the LLM-based log analysis tool.

This chapter introduced the foundational concepts underpinning this study, including log analysis, root cause analysis, LLMs, prompt design, token limitations, and the industrial LLM-based log analysis tool developed at Bosch. The tool’s internal workflow reveals that output quality is shaped by the interplay between preprocessing methods, prompt structure, and the LLM’s reasoning capability, none of which are captured by conventional Natural Language Processing (NLP) evaluation metrics. This gap directly motivates the first research question: identifying which technical and human-centered criteria are appropriate for evaluating such outputs in an industrial CI context. Similarly, the tool’s design as a standalone web-based tool, operating without agentic components, establishes the baseline against which the MCP-based integration examined in RQ4 is compared, in terms of how practitioners perceive it. Together, these concepts position the evaluation and perceived workflow influence of LLM-based log analysis as an open research problem, which the following chapters address through technical evaluation, thematic analysis, and correlation analysis.

3

Related Work

Given the challenges and opportunities discussed in the previous chapter, this chapter reviews prior research on log-based anomaly detection and root cause analysis. It first outlines traditional statistical and deep learning approaches, then discusses transformer-based methods, and finally presents recent LLM-driven frameworks for log analysis. It also examines LLM-as-a-judge evaluation frameworks and highlights the research gap this thesis addresses.

3.1 Deep Learning-Based Log Anomaly Detection

Before the introduction of transformers and LLMs, research primarily focused on recurrent and template-based deep learning techniques. Log analysis can uncover system anomalies by leveraging frequency and sequence statistics in large operational datasets [1]. Their experience study highlighted common challenges in industrial environments, such as large volumes of unlabeled logs and dynamic changes in system behavior that render static rules ineffective.

A significant milestone was the introduction of DeepLog [2], which used Long Short-Term Memory (LSTM) to learn normal log sequences and detect deviations as potential anomalies. DeepLog represented a transition from manual feature engineering to automated sequence learning, enabling anomaly detection without domain-specific templates. However, DeepLog and similar approaches require extensive preprocessing, stable log structures, and reliable sequence-consistency conditions that are rarely guaranteed in real-world systems.

Deep learning for log anomaly detection was extended by applying BERT to model bidirectional contextual dependencies between log events [14]. LogBERT was less dependent on log templates and demonstrated improved accuracy compared to LSTM-based systems. Yet, it still relied on training data, struggled with unseen log patterns, and did not solve fundamental scalability limitations related to high-dimensional vector representations.

3.2 Transformer-Based Log Anomaly Detection

The emergence of transformer architectures brought increased contextual awareness to log analysis. LogLLaMA was proposed by [16], using the Large Language Model Meta AI (LLaMA) architecture to detect anomalies by capturing semantic relationships between log lines. This model improved over single-directional sequence learning, but required fine-tuning, filtering, and domain-specific preprocessing to remain effective, especially in dynamic production environments.

LogLLM was introduced by [17], a framework that incorporates LLM reasoning into anomaly detection tasks. Although LogLLM demonstrated that pretrained models can interpret log content with minimal domain knowledge, it was computationally expensive, had limited context capacity, and struggled to process very large log corpora efficiently. These issues illustrate a fundamental limitation of pure transformer architectures: while context-aware, they become resource-intensive when dealing with millions of log entries.

Prompt-based strategies for log analysis have been examined by [19]. Rather than training custom models, the study focused on controlling the outputs of pretrained LLMs using structured prompts. This approach improved interpretability and enabled online analysis, but the authors acknowledged sensitivity to prompt engineering and variability in performance depending on dataset complexity and log format consistency.

3.3 LLM-based Log Analysis

The latest research trend involves using pretrained LLMs for log understanding, clustering, and anomaly explanation. A survey on LLM-based event log analysis methods, providing a structured overview of recent developments, was presented by [9]. They observed that while LLMs offer greater semantic understanding than previous approaches, most existing work is preliminary and focuses on controlled experiments rather than on industrial-scale deployment.

The application of ChatGPT to anomaly detection tasks was explored in LogGPT [18]. Their work highlighted LLMs' strengths in generating contextual descriptions and interpretive explanations of anomalies. However, they reported inconsistent performance when the log messages contained highly technical domain-specific vocabulary not represented in the model's pretraining dataset.

LLMLogAnalyzer, a chatbot-style system that uses clustering to group similar log entries before applying LLM reasoning, was developed by [10]. This hybrid approach reduces the number of LLM queries, improving scalability while preserving semantic insight. Nevertheless, the authors acknowledged limitations in clustering quality, hallucination risk, and the absence of standardized evaluation metrics between datasets.

An unsupervised LLM-based log parser that eliminates the need for labeled examples was proposed by [15]. Their study demonstrated promising performance, especially on heterogeneous logs, yet their system still required careful tuning of prompting strategies and domain adaptation to avoid misinterpretation of rare or infrequent log patterns.

In addition to purely LLM-centric approaches, hybrid systems that integrate keyword-based analysis and vector database-driven search have gained attention for scalable and context-aware log processing. Tools such as Bosch's log analyzer combine simple deterministic keyword matching and a semantic search over a vector database (e.g., Qdrant) that stores high-dimensional embeddings of known "good" log lines, enabling efficient retrieval of similar patterns and deviations before applying LLM reasoning. This strategy aligns with broader trends in the integration of LLMs and vector databases, where vector stores are used to manage and retrieve high-dimensional representations, mitigating issues such as limited context windows and enhancing semantic retrieval efficiency [20]. By leveraging vector search to pre-filter and contextualize log entries for subsequent LLM analysis, such hybrid pipelines aim to improve both scalability and semantic relevance in industrial log analysis settings.

As LLMs are increasingly used to generate explanations, summaries, and recommendations, evaluating the quality of these outputs has become a critical challenge. Human evaluation, while often considered the gold standard, is difficult to scale at large volumes of content, making the process both time-consuming and resource-intensive. This limitation has led to growing interest in using LLMs not only as content-generation tools but also as evaluators of generated outputs, in an emerging paradigm often referred to as *LLM-as-a-Judge* [30].

3.4 LLM-as-a-Judge for Evaluating Generated Outputs

Traditional automatic evaluation metrics such as BLEU [12] and ROUGE [13] rely primarily on lexical similarity between generated text and reference outputs. While useful for tasks with clearly defined ground truth, they are less suitable for open-ended tasks where multiple valid responses may exist. This limitation is particularly relevant in domains such as log analysis, where the value of an output depends less on exact wording and more on its correctness, usefulness, and actionability [30].

In the *LLM-as-a-Judge* paradigm, an LLM assesses generated outputs based on criteria such as relevance, coherence, factual consistency, and helpfulness, offering a more scalable and semantically aware alternative to manual evaluation [30]. However, prior work also highlights important limitations: LLM judges may favor fluent but incorrect responses, show bias toward particular response styles, and produce inconsistent judgments depending on prompt design and evaluation setup.

These challenges are especially important in log analysis, where outputs must not

only be coherent but also support effective debugging and failure diagnosis. An explanation may sound convincing while failing to identify the actual root cause, or a summary may omit details critical for troubleshooting. Therefore, evaluating LLM-generated log analysis outputs requires going beyond semantic quality alone and also considering practical usefulness, including trustworthiness, actionability, and relevance to real debugging workflows.

Next, we present the research gaps that motivate this thesis, particularly regarding the evaluation of LLM-generated log analysis outputs and their use in industrial CI workflows.

3.5 Research Gap and Positioning of This Thesis

Despite rapid advances in applying LLMs to software log analysis and anomaly detection, several key gaps remain, hindering the reliable use of LLMs in industrial CI environments. First, much of the existing work focuses on improving technical performance on curated datasets or benchmarking tasks (e.g., anomaly detection or template extraction) but offers limited insight into how these tools behave, are interpreted, or are evaluated in realistic, heterogeneous CI environments where logs are noisy, context-dependent, and task ambiguity is high. This creates a disconnect between controlled technical evaluation and practical use cases in industry [31].

Second, there is no clear, holistic way to evaluate LLM-based log-analysis systems. Current research focuses on accuracy, precision, and recall, but these metrics overlook key human-centered factors such as trust, perceived usefulness, interpretability, and actionability in debugging workflows [31].

Third, while hybrid approaches that combine pre-processing, retrieval, and LLM reasoning are increasingly proposed to handle complex log structures, practitioner perceptions of their usefulness and workflow support remain underexplored. It is unclear how developers perceive these tools in relation to their own practices, workflow efficiency, and decision-making during debugging. Understanding these perceived influences is critical to assess the perceived usefulness of LLM-based log analysis tools beyond technical performance metrics [31].

Most prior work focuses on single-shot tasks such as summarization and anomaly detection. By contrast, approaches that use agents or multi-step reasoning, such as Model Context Protocol (MCP)-based agentic systems, have not been systematically compared with simpler summarization techniques in terms of CI practitioners' perceived usability and workflow influence. Comparing these approaches helps determine whether agent-based systems improve CI troubleshooting compared to simpler methods from practitioners' perspectives.

This thesis addresses these gaps through an exploratory case study of LLM-supported log analysis in realistic CI environments. Through this multi-faceted approach, the thesis aims to bridge the divide between technical advances in LLM-driven log analysis and their perceived use and evaluation in software engineering.

4

Methods

This chapter presents the methodological approach used to address the research questions of this thesis. The study is conducted as an industrial case study in collaboration with Bosch and combines qualitative and quantitative methods to evaluate an LLM-based log analysis tool in CI environments. The chapter begins with an overview of the research design, followed by a detailed description of the methods used to address each research question, including the identification of evaluation criteria, practitioner-based inquiry, and the tool output evaluation. It then outlines the methodological approaches used to examine practitioners' perceptions of the tool's influence on CI workflows, as well as practitioners' perceptions of the influence of MCP-based agent integration on CI workflows.

4.1 Research Design and Methods Overview

This thesis investigates the evaluation and perceived application of an LLM-based log analysis tool within an industrial CI environment. Given the exploratory and practice-oriented nature of the research, the study adopts an industrial case study design using a mixed-methods approach. The methodology integrates qualitative and quantitative methods and draws on multiple data sources, including literature-derived evaluation metrics, industrial log data, practitioner interviews, and expert-based assessments of tool outputs.

The research design is structured around four research questions. **RQ1** focuses on identifying evaluation metrics for assessing LLM-based log analysis tool outputs, covering both technical and human-centered perspectives. **RQ2** examines how evaluation metrics are used in practice and the alignment between automated evaluation metrics and practitioner judgments. **RQ3** investigates practitioner perceptions of the usefulness and workflow influence of the LLM-based log analysis tool in CI troubleshooting. **RQ4** examines practitioners' perceptions of the influence of MCP-based IDE integration into the development environment compared with the one-shot summary standalone web-based tool. An overview of the methodological workflow and its alignment with the research questions is presented in **Figure 4.1**.

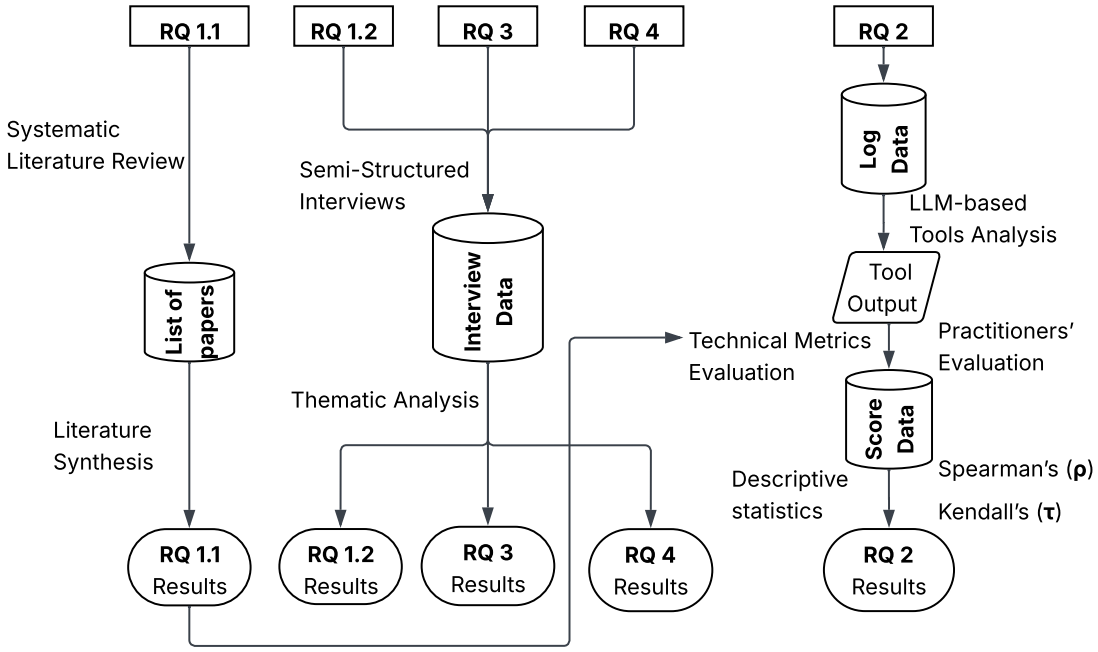


Figure 4.1: Overview of the methodological design across the four research questions: evaluation criteria (RQ1), automated-human evaluation alignment (RQ2), practitioners' perceptions of tool influence on CI workflows (RQ3), and practitioners' perceptions of MCP-based IDE integration (RQ4).

The first phase addresses **RQ1** by establishing an evaluation framework. Technical evaluation metrics (**RQ1.1**) are identified through a **structured literature review** and **literature synthesis**, then applied to assess the quality of tool outputs and compared with human evaluations to examine alignment. In parallel, human-centered evaluation criteria (**RQ1.2**) are derived from **semi-structured practitioner interviews** and analyzed using **thematic analysis**.

To address **RQ2**, a structured testing framework is developed to evaluate tool outputs using the technical metrics identified in RQ1.1, and to assess the alignment between automated and human evaluations for validation. The LLM-based tool is applied to industrial log data, and its outputs are assessed through two complementary approaches: (i) automated evaluation implemented in Python, and (ii) human evaluation conducted by practitioners. The resulting scores are analyzed using **descriptive statistics** and **correlation analysis**, specifically **Spearman (ρ)** and **Kendall's Tau (τ)**, to determine the degree of agreement between automated and human judgments.

The second qualitative phase addresses **RQ3** by examining CI practitioners' perceptions of the usefulness and workflow influence of the LLM-based log analysis tool. This is conducted through semi-structured practitioner interviews and thematic analysis, focusing on perceived support for debugging efficiency, failure investigation, decision-making, and perceived unintended risks in CI environments.

The extended second phase addresses **RQ4** by examining CI practitioners' perceptions of an MCP-based agentic version of the tool compared with a single-shot web-based tool. This comparison focuses on perceived differences in workflow integration, interaction patterns, and usability, particularly in relation to CI troubleshooting. The findings are analyzed qualitatively using **thematic analysis** to identify how practitioners perceive the influence of agent-based integration on their workflows.

The overall research process follows a semi-sequential design, where findings from earlier phases inform subsequent stages. In particular, the technical evaluation metrics identified in RQ1.1 provide the basis for addressing **RQ2**. While the evaluation framework developed in **RQ1** informs the broader analysis, the investigations of practitioners' perceptions regarding workflow influence (**RQ3**) and MCP-based agent integration (**RQ4**) are conducted independently.

4.2 Methods Used to Answer RQ1.1

This section describes the methods used to identify and validate the technical evaluation criteria. It outlines both the data collection and analysis methods and procedures employed to address the research question.

4.2.1 Data Collection

This section describes the data collection process used to answer RQ1.1. Since no established evaluation framework exists specifically for LLM-generated log analysis outputs, relevant secondary data were collected from academic literature and related research domains. The collected sources were used to identify commonly applied technical evaluation criteria and approaches suitable for this study.

4.2.1.1 Systematic Literature Review

We conducted a Systematic Literature Review (SLR) to identify technical evaluation metrics for LLM-based log analysis tools. We used a structured, reproducible process based on established software engineering review guidelines, such as those from Barbara Kitchenham [32]. We aimed to gather and analyze relevant evaluation metrics from related domains. Since no specific set exists for LLM-based log analysis, we included studies on LLM evaluation, natural language generation, and AI-assisted software engineering tools.

We searched major academic databases: IEEE Xplore, ACM Digital Library, SpringerLink, Elsevier ScienceDirect, and Google Scholar. We combined keywords such as "LLM evaluation," "natural language generation evaluation metrics," "LLM-generated output evaluation," and "Software log analysis tool evaluation." We limited our search to peer-reviewed English-language publications, but also included arXiv papers highly relevant to our topic. We prioritized recent studies to reflect current practices in LLM-based system evaluation.

To ensure relevance and quality, we applied inclusion and exclusion criteria during screening. Papers were included if they proposed, used, or discussed technical evaluation metrics applicable to LLM outputs or similar AI-generated artifacts. We excluded studies that focused exclusively on domain-specific applications without transferable evaluation criteria, came from non-peer-reviewed sources, or lacked methodological clarity.

The study selection process was conducted in multiple stages. First, an initial screening was performed based on titles and abstracts. This was followed by a full-text review of the remaining studies to determine final eligibility. The selection process and the number of studies included and excluded at each stage are reported in the results Section 5.1.1.1.

4.2.2 Data Analysis

This section describes the data analysis process used to answer RQ1.1. The collected literature was reviewed and synthesized to identify recurring technical evaluation criteria, commonly used metrics, and relevant assessment approaches. The analysis was used to determine which methods were most suitable for evaluating LLM-generated log analysis outputs in this study.

4.2.2.1 Literature synthesis

A literature synthesis [33] was conducted to systematically identify and categorize technical evaluation metrics for LLMs. This synthesis [34] employed a structured literature review and qualitative synthesis methodology, collecting and analyzing relevant academic papers, evaluation frameworks, and benchmarking studies on natural language generation and LLM evaluation. The literature was reviewed iteratively to extract commonly used evaluation metrics, their underlying principles, limitations, and appropriate application contexts. The identified metrics were then grouped into higher-level categories based on their evaluation methodologies, yielding three main categories: statistical scorers, model-based scorers, and LLM-as-a-judge approaches. This categorization emerged from a comparative analysis of evaluation mechanisms described across the literature, rather than from a single source. The primary aim of the synthesis was to identify suitable technical evaluation metrics for the LLM-based log analysis tool within the context of this study. The outcome of this literature synthesis served as the basis for selecting and defining the technical evaluation metrics used in the subsequent testing phase.

4.3 Methods Used to Answer RQ1.2

This section describes the methodology used to identify human-centered evaluation criteria for LLM-generated log analysis outputs. The focus is on understanding how CI practitioners assess the usefulness, clarity, trustworthiness, and actionability of such outputs in real-world CI workflows.

4.3.1 Data Collection

To address **RQ1.2**, qualitative data were collected through semi-structured interviews with CI practitioners. The aim was to understand how practitioners evaluate LLM-generated log analysis outputs in real CI workflows. The following subsections describe the semi-structured interview, participant sampling process, pilot study and interview procedure.

4.3.1.1 Semi-Structured Interview Design

Semi-structured interviews were used as the primary data collection method. This approach provided a balance between consistency across interviews and flexibility to explore participant-specific experiences in greater depth.

An interview guide was prepared in advance in the form of a questionnaire aligned with RQ1.2. The guide was developed based on the research question and the industrial CI context. Scenario-based questions were included to encourage participants to reflect on realistic troubleshooting situations and evaluate tool generated outputs in context.

The final guide consisted of eight main questions. Two introductory questions focused on participant background, including CI experience and familiarity with LLM-based tools. The remaining six questions aimed to understand the explicit criteria (metrics) practitioners use (e.g., how they judge whether an explanation is good enough and what makes them trust or distrust it). Although the same core questions were asked in each session, follow-up probing questions were used where relevant to clarify responses or encourage elaboration. The complete interview guide is provided in Appendix A.

4.3.1.2 Participants and Sampling

A total of 9 CI practitioners from the industrial partner organization, Bosch, participated in the study. All participants had practical experience with CI-related activities such as build monitoring, troubleshooting failed pipelines, log inspection, and debugging tasks. We selected 9 participants as this was sufficient to reach data saturation, where no substantially new insights emerged from additional interviews.

A purposive sampling strategy was used to recruit participants with relevant domain expertise. Participants were selected through internal coordination with the industrial partner based on their involvement in CI workflows and experience with failure investigation.

The sample represented a range of CI responsibilities, experience levels, and familiarity with the Log Analyzer tool. In focused qualitative studies, smaller expert samples are appropriate when participants possess domain-specific knowledge and when recurring patterns begin to emerge across interviews. The participant group included both junior and senior engineers, allowing perspectives from different levels of experience to be captured. **Table 4.1** provides an anonymized overview of the

participating practitioners, including their experience, scope of CI work, familiarity with the tool, and role in the study. For RQ1.2, 9 practitioners (P1–P9) participated in the interview and evaluation phase.

Table 4.1: Demographic and professional overview of practitioners participating in the interview-based study

ID	CI Exp. (Years)	LLM Exp. (Years)	CI Scope	Tool Familiarity	Role in the study
P1	13	3	All levels	Yes	Interview,Evaluation
P2	18	2	All levels	No	Interview,Evaluation
P3	5	3	Component	No	Interview,Evaluation
P4	11	2	Component	No	Interview,Evaluation
P5	2	1	Component	Yes	Interview,Evaluation
P6	11	3 months	Component	No	Interview,Evaluation
P7	20	3	Complete	Yes	Interview,Evaluation
P8	5	6 months	Complete	Yes	Interview,Evaluation
P9	26	2	Complete	Yes	Interview,Evaluation

4.3.1.3 Pilot Study

Before the formal interviews, the interview guide was pilot tested in two separate sessions with industrial CI Practitioners. The purpose of the pilot sessions was to evaluate question clarity, interview flow, and expected time duration.

Based on feedback, minor refinements were made to the interview procedure. A short introduction to the Log Analyzer tool was added before the interview questions, the order of some questions was adjusted to improve flow, and greater use of follow-up prompts was planned when participants provided brief responses.

4.3.1.4 Interview Procedure

The interviews were conducted remotely through Microsoft Teams and each session lasted approximately one hour. At the start of each interview, participants received a brief introduction to the Log Analyzer tool and its functionality to ensure a consistent understanding of the system under evaluation. Participants were informed about the purpose of the study, the use of audio recordings for research analysis, and the anonymization of collected data. Verbal consent was obtained prior to recording.

All sessions were audio recorded using Screenpresso, and notes were taken during the interviews. Recordings were later transcribed using a combination of automated transcription tools and manual correction. To preserve confidentiality, participants were anonymized in the transcripts using labels such as Speaker 1, Speaker 2, and Speaker 3. Participation was voluntary, and participants could withdraw at any time.

4.3.2 Data Analysis

This section describes the data analysis process used to answer RQ1.2. The interview data collected from practitioners were analyzed using qualitative methods to identify recurring patterns, perspectives, and evaluation criteria regarding LLM-generated log analysis outputs. The analysis focused on understanding how practitioners assess the usefulness and quality of the tool outputs in practical CI troubleshooting contexts.

4.3.2.1 Thematic Analysis

Thematic analysis [35] was conducted on the data gathered from the semi-structured interviews with practitioners in the partner company, Bosch. The main purpose of the thematic analysis was to identify human-centered evaluation metrics for assessing the outputs of an LLM-based log analysis tool, particularly whether they were considered useful for debugging or perceived as nonsensical.

All interviews were recorded and transcribed using MS Teams[1] to facilitate systematic analysis. The analysis followed an **inductive coding approach**, in which codes were derived directly from the data rather than predefined categories. An inductive approach was chosen due to the exploratory nature of the study and the lack of well-established human-centered evaluation frameworks for LLM-based log analysis tools.

We performed **open coding**[2] on each interview transcript using Taguette[3]. During this process, meaningful segments of text were identified and assigned initial codes that captured key ideas relevant to the research question. Examples of such codes include **spotting the correct error**, **noise in logs**, **too generic explanations**, **suggested fixes**, and **confidence in the output**. For instance, statements in which participants emphasized identifying the correct error among irrelevant log messages were coded as spotting the correct error and as noise filtering. These codes were later grouped into the sub-theme **Exact Error Identification and Relevance**. Similarly, comments describing **vague or non-specific responses** were coded as too generic explanations and grouped under the sub-theme **Specific and Meaningful Explanations**. These codes emerged iteratively as we examined the transcripts.

To ensure rigor and reduce bias, the transcripts were divided between the researchers for initial coding. Subsequently, the coded transcripts were exchanged for cross-checking, where discrepancies in code interpretation were discussed and resolved. This iterative process led to the refinement, merging, and clarification of codes.

Following code extraction, related codes were grouped into sub-themes, representing recurring evaluation aspects mentioned by practitioners. These sub-themes were then further aggregated into broader themes that capture the main human-centered

[1] `https://www.microsoft.com/en-us/microsoft-teams/group-chat-software`
[2] `https://delvetool.com/blog/openaxialselective`
[3] `https://app.taguette.org/`

evaluation criteria used in practice. The themes were reviewed and validated against the original interview data and notes taken during the interviews to ensure consistency and representativeness.

The final thematic structure was used to answer the research question and to inform the design of evaluation metrics.

4.4 Methods Used to Answer RQ2

After identifying technical criteria through the literature review and human-centered criteria through practitioner interviews, the selected technical criteria were operationalized in a structured evaluation activity. This part of the study was designed to examine how the outputs of the LLM-supported log analysis tool could be assessed in practice using the selected criteria compared with human judgments.

4.4.1 Data Collection

This section describes the data collection process used to answer RQ2. To evaluate the selected criteria in practice, secondary data from industrial CI log environments and practitioner assessments were collected. Real log files were used as input for the LLM-based log analysis tool, while expert evaluations were gathered to assess the quality of the generated outputs across multiple criteria.

4.4.1.1 Secondary data obtained from company log storage systems

The secondary data used in this study consisted of log files obtained from Bosch's log storage systems and associated project environments. In total, the dataset contained 428 log files, including 29 failed logs and 399 successful logs. All available failed logs included in the dataset were used in the study. The failed logs were used as the primary failure-related input, as they contain the information relevant for log analysis and fault investigation. The successful logs were used during the preparation phase to generate vector embeddings and populate the vector database used by the neural search component. Before use, the logs were processed and filtered to generate summaries suitable for LLM-based analysis.

Using log data from an actual industrial setting enabled the study to work with realistic CI artifacts produced in practice, rather than relying on synthetic or publicly available examples.

4.4.1.2 Expert-Based Evaluation of LLM Log Analysis Outputs Using Multi-Criteria Assessment

An expert-based evaluation was conducted to assess the quality of LLM-generated log analysis outputs using a multi-criteria assessment approach. This method was chosen because the usefulness of log analysis outputs in industrial CI contexts depends not only on whether they appear reasonable, but also on how well they support fault understanding and troubleshooting in practice. The study focused on

evaluating the log analysis output generated by the tool using the technical metrics identified in RQ1.1 and comparing them with human judgments for metric validation. To perform the evaluation, a Microsoft Forms[4] questionnaire was designed and shared with the practitioners, including a brief description of the study and instructions to complete the evaluation (see Appendix B).

Participants were asked to evaluate the outputs based on four criteria: correctness, completeness, clarity, and actionability, using a 5-point Likert scale, and to provide reasoning for their scores. These criteria were selected to capture both the factual quality of the output and its practical value from a user perspective. The participants were the same 9 CI practitioners involved in RQ1.2 (see Table 4.1), ensuring consistency in experience and context. Each participant received four analysis output files in HTML visualizer format generated by the LLM-based log analysis tool. These files were intentionally mixed to include outputs from different analysis methods: keywords, regular expression, and neural search. This setup provided a structured way to collect expert judgments across multiple evaluation criteria while allowing comparison of practitioner perceptions across different analysis methods.

4.4.2 Data Analysis

The quantitative evaluation data were analyzed to compare how the generated outputs were assessed by human practitioners and by the LLM-based evaluator across the selected technical criteria. Descriptive statistics were first used to summarize the distribution of scores for each evaluation criterion, including measures such as mean, median, standard deviation, minimum, and maximum. This provided an overview of how the outputs were rated across both evaluation approaches.

To further examine the relationship between human and LLM-based scores, Spearman's rank correlation and Kendall's tau were used. These non-parametric methods were appropriate because the evaluation scores were ordinal in nature and the purpose of the analysis was to examine the degree to which the two evaluation approaches produced similar ranking patterns across outputs.

The thematic analysis of interview data used to identify human-centered evaluation criteria is described separately in Section 4.3.2.1. In contrast, the analysis presented here focuses specifically on the technical evaluation scores and their comparison across human and LLM-based assessment.

4.4.2.1 Descriptive statistics

To summarize and describe results from both the LLM-based evaluator and practitioners, we applied descriptive statistics[5] to the score data. These statistics organize and present quantitative data without making claims beyond the observed sample. For each criterion, correctness, completeness, clarity, and actionability, scores from 29 log file outputs were aggregated and analyzed. We calculated the mean and

[4] `https://forms.office.com/Pages/DesignPagev2.aspx`
[5] `https://en.wikipedia.org/wiki/Descriptive_statistics`

median to show typical scores, and the standard deviation and range to assess variability and consistency. This enabled a clear comparison of how each group rated the tool's outputs across quality dimensions.

We analyzed LLM evaluators and practitioners separately to compare scoring patterns. This highlights differences between automated and human assessments. The structured evaluation format enabled systematic extraction and numerical processing of each criterion's scores. Descriptive summaries show the distribution and tendencies within each group's evaluations and serve as the basis for further statistical comparisons. This approach follows standard quantitative analysis, starting with descriptive statistics.

4.4.2.2 Spearman's and Kendall's Correlation Between Human and LLM Evaluations

To validate the technical evaluation metrics identified through literature synthesis, we conducted a structured testing procedure on the LLM-based log analysis tool. The tool's outputs were evaluated using these metrics, yielding scores for correctness, completeness, clarity, and actionability. To assess alignment between automated tool evaluations and human judgments, we computed **Spearman's rank correlation**[6] and **Kendall's tau correlation**[7] between practitioner-provided and tool-generated scores.

These methods were selected because the evaluation data consisted of 5-point Likert-scale scores, which are ordinal rather than continuous interval data. The analysis, therefore, focused on whether outputs that received higher scores from automated evaluators also tended to receive higher scores from practitioners, rather than assuming equal numerical distances between scale points. Furthermore, the distributional assumptions required by parametric correlation methods, such as Pearson's correlation, could not be guaranteed due to the relatively small sample size and the subjective nature of the evaluation scores. Both Spearman's rank correlation and Kendall's tau are nonparametric measures that assess the degree to which two variables preserve the same ordering of observations, without assuming linear relationships or normally distributed data.

Spearman's rank correlation (ρ) measures the strength and direction of a monotonic relationship between two ranked variables by comparing their rank orderings. It is particularly useful for assessing whether higher scores assigned by one evaluator tend to correspond to higher scores assigned by another evaluator. Kendall's tau (τ) measures agreement based on the number of concordant and discordant pairs of observations and is generally considered more robust to ties and small sample sizes. While both metrics assess rank agreement, Kendall's tau provides a more conservative estimate of association, whereas Spearman's correlation is often more sensitive to broader ranking trends.

[6] https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient
[7] https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient

The coefficients should be interpreted as indicators of ranking alignment rather than exact agreement in absolute score values. A positive coefficient means that outputs ranked higher by the automated evaluator are also ranked higher by practitioners, while values close to zero indicate little consistent ranking relationship. Negative values indicate that the two evaluation approaches rank outputs in opposite directions for that criterion.

Using both measures provides complementary perspectives on agreement between human and automated evaluations. Consistent results across the two correlation coefficients increase confidence that the observed relationships are not artifacts of a single statistical measure. This procedure, therefore, provides an empirical assessment of the identified technical metrics and their alignment with practitioner judgments.

4.5 Methods Used to Answer RQ3 and RQ4

This section describes the methodology used to investigate practitioner perceptions of how the LLM-based log analysis tool supports CI workflows in industrial practice and how MCP-based integration affects interaction with the tool during troubleshooting activities. While **RQ3** focuses on perceived usefulness, limitations, and potential workflow support during debugging and failure investigation, **RQ4** extends this perspective by examining practitioner perceptions of integrating an MCP-based agent within the development environment in comparison to the standalone web-based tool.

4.5.1 Data Collection

To address RQ3 and RQ4, qualitative data were collected through semi-structured interviews with CI practitioners. The overall data collection approach follows the same methodology as described in Section 4.3; however, the interview guide was extended to focus on workflow-related aspects and comparative experiences between the standalone web-based tool and the MCP-based integration.

4.5.1.1 Semi-Structured Interview Design

Semi-structured interviews were used as the primary data collection method to explore how practitioners perceive the LLM-based log analysis tool and its MCP-based integration influence CI workflows in practice.

A dedicated interview guide was developed to capture practitioner experiences with CI failure investigation, debugging workflows, and interaction with both the web-based tool and the MCP-based integration. The guide consisted of open-ended questions designed to explore how practitioners investigate CI failures, the challenges of manual log analysis, and practitioners' perceptions of how the tool affects troubleshooting activities.

The interview questions covered aspects such as debugging strategies, workflow ef-

ficiency, changes in investigation steps, confidence in root cause identification, integration into existing workflows, and situations where the tool is useful or less effective. In addition, participants were asked to compare their experiences using the standalone web-based tool and the MCP-based integration within the development environment. Questions related to workflow continuity, context switching, interaction style, iterative exploration, and usability were also included.

Participants were also encouraged to compare their usual troubleshooting approach with and without the tool. Follow-up questions were used to explore responses in greater depth, particularly when discussing workflow changes and decision-making processes. The complete interview guide is provided in Appendices A.3 and A.4.

4.5.1.2 Participants and Sampling

A total of 14 CI practitioners from the industrial partner organization participated in this phase of the study. This includes the 9 participants from RQ1.2 (see Section 4.3.1.2), along with five additional practitioners, enabling a broader range of perspectives on workflow-related aspects and direct comparison between experiences with the web-based tool and the MCP-based integration. Similar to RQ1.2, a purposive sampling strategy was used to recruit participants with relevant experience in CI workflows, log analysis, and failure investigation. The sample includes practitioners with varying levels of experience and responsibilities, allowing the study to capture diverse perspectives on how the tool is perceived and used in practice. Tables 4.1 and 4.2 present an overview of all practitioners involved in the study.

Table 4.2: Demographic and professional overview of practitioners participating in the interview-based study (RQ3 and RQ4 only)

Participants P1–P9 are reported in Table 4.1					
ID	**CI Exp. (Years)**	**LLM Exp. (Years)**	**CI Scope**	**Tool Familiarity**	**Role in the study**
P10	5	2	All Levels	Yes	Interview
P11	10	1	Component	No	Interview
P12	6	1	Component	Yes	Interview
P13	4	1	Component	Yes	Interview
P14	11	3	Complete	Yes	Interview

4.5.1.3 Pilot Study

The interview guide was pilot tested with two industrial CI practitioners to assess question clarity, interview flow, and relevance to real CI troubleshooting workflows. Based on feedback, minor refinements were made to question phrasing and ordering. Additional emphasis was placed on encouraging participants to elaborate on

workflow changes, practical debugging experiences, and comparisons between the standalone and MCP-based interaction approaches.

4.5.1.4 Interview Procedure

The interviews were conducted remotely through Microsoft Teams and lasted approximately one and a half hours. At the beginning of each session, participants were informed about the purpose of the study, recording procedures, and anonymization of collected data. Verbal consent was obtained before recording began.

Participants were first introduced to the LLM-based log analysis tool and its MCP-based integration through a brief walkthrough demonstrating how the tool is used during troubleshooting activities. This included demonstrating interaction with the MCP-based agent within the development environment to ensure that participants had a consistent understanding of both approaches before responding to the interview questions.

All sessions were audio recorded with participant consent and later transcribed using a combination of automated transcription tools and manual correction. To preserve confidentiality, anonymization was applied in the same manner as described in Section 4.3.1.4.

4.5.2 Data Analysis

This section describes the method used to analyze data collected through semi-structured interviews, with the aim of examining practitioners' perceptions of how the LLM-based log analysis tool influences CI workflows and how integrating an MCP-based agent into the development environment is perceived relative to a standalone web-based tool.

4.5.2.1 Thematic Analysis

Thematic analysis [35] was conducted on the semi-structured interview data to examine practitioners' perceptions of how the LLM-based log analysis tool influences CI troubleshooting workflows (RQ3) and how MCP-based integration affects workflow continuity and interaction patterns compared to the standalone web-based tool (RQ4). The overall transcription and analysis procedure follows the process described in Section 4.3.2.1.

Given the exploratory nature of the study, the analysis followed a primarily **inductive approach**. While the interview questions for RQ3 and RQ4 provided general direction by encouraging participants to reflect on troubleshooting practices, workflow changes, integration experiences, and tool usage patterns, the themes emerged from recurring patterns in the data rather than predefined theoretical categories.

Open coding was used to identify meaningful data segments and assign initial codes reflecting participants' experiences and practices.

For RQ3, examples of open codes included **manual log scanning**, **root cause identification difficulty**, **faster identification of failure points**, **reduced manual log inspection**, and **human verification still required**.

For RQ4, examples included **reduced context switching**, **staying inside VS Code**, **iterative debugging process**, **conversational interaction with logs**, and **tool complementarity**.

Related codes were iteratively grouped into sub-themes and broader themes. For example, RQ3 codes concerning manual troubleshooting effort and AI-assisted debugging contributed to sub-themes such as **Manual and Heuristic-Based Debugging**, **Reduction in Manual Effort**, and **Iterative Human-AI Collaboration Loop**.

Similarly, RQ4 codes related to IDE integration, conversational interaction, and workflow continuity contributed to sub-themes including **Reduced context switching improves workflow continuity**, **Conversational interaction supports progressive investigation**, and **Existing practices are complemented rather than replaced**.

Following coding, related sub-themes were further aggregated into broader themes through an iterative process analogous to axial coding. The resulting themes capture key dimensions of workflow influence, including troubleshooting efficiency, interaction patterns, integration preferences, workflow adaptations, and perceived risks associated with AI-assisted CI troubleshooting.

5

Results

This chapter presents our findings from the literature on technical evaluation metrics and the thematic analysis of CI practitioners' evaluation perspectives and metrics. Following the results for technical evaluation metrics, we present the themes derived from the semi-structured interviews.

5.1 Evaluation Metrics

This section presents the findings on evaluation metrics. It includes technical evaluation metrics (**RQ1.1**) identified through literature synthesis and validated through testing, as well as human evaluation perspectives (**RQ1.2**), which were analyzed using thematic analysis.

5.1.1 Technical Evaluation Metrics (RQ1.1)

This section presents our findings on the technical evaluation criteria for LLM-based log analysis tool outputs, based on a literature review and validation testing. Since there was no literature specifically addressing LLM-based log analysis, our literature review began with an examination of LLM-based application evaluation metrics. So, to answer our research question, we first conducted an extensive literature review [32] of LLM-based applications and a literature synthesis [34] to identify relevant metrics. The identified metrics, their categories, and their detailed literature synthesis are presented in **Table 5.1**.

5.1.1.1 Literature Selection Process

The literature selection process proceeded in several stages. First, we collected 39 papers evaluating LLM-based applications. We then conducted a title and abstract screening, removing studies that focused on model-level metrics, such as precision and recall, rather than on the evaluation of overall application outputs. This resulted in 26 remaining studies.

Next, a full-text review of these 26 studies was performed. Papers focusing on RAG-specific or agent-based evaluation metrics were excluded, as the scope of this study is limited to a single LLM-based log analysis tool without RAG or agent components. This step reduced the set to 18 studies.

Finally, during the in-depth reading phase, studies focused on highly specific metrics, such as the QAG Score (Question-Answer Generation) and the DAG (Directed Acyclic Graph), were excluded because they did not directly address the research questions. This resulted in a final set of 12 studies included in the analysis.

From the selected studies, several technical evaluation metrics were identified, including BLEU, ROUGE, BERTScore, and LLM-based evaluation methods. These metrics were grouped into three categories: statistical scorers, model-based scorers, and LLM-as-a-judge approaches. The findings indicate that these categories are commonly used in evaluating LLM-generated outputs across text generation and software engineering contexts. This categorization informed the selection of metrics used in the metrics validation evaluation framework.

5.1.1.2 Literature Synthesis

This section presents the results of the literature synthesis, summarizing the evaluation metrics identified in previous studies and their relevance to this research.

1. **Statistical Scorers:** Metrics such as BLEU [12], ROUGE [13], METEOR [36], and Edit Distance (Levenshtein)[2] are computationally efficient and objective. However, these metrics do not capture semantic meaning or address reasoning-intensive tasks. Their reference-based nature limits their applicability, as they are primarily designed to evaluate outputs against reference texts. This approach is unsuitable for LLM-based analysis outputs, which lack reference outputs due to the large size of log files. Furthermore, existing literature consistently demonstrates a low correlation between these metrics and human judgment [40]. The intended use cases for these metrics are language translation, text summarization, paraphrasing, and text comparison. Therefore, statistical scorers are inadequate for evaluating LLM-based log analysis outputs.

2. **Model-Based Scorers:** Model-based scorers, such as BERTScore [37], MoverScore [38], and BLEURT [39], demonstrate superiority over statistical metrics by capturing semantic similarity and exhibiting stronger correlation with human judgment [44]. Nevertheless, these metrics remain imperfect and often function as black-box evaluators. Most of these metrics are reference-dependent and are best suited for text summarization, paraphrasing, and natural language generation. Their capabilities are therefore limited and do not fully address scenarios requiring extensive reasoning, such as LLM-based log analysis. Consequently, these metrics serve as an intermediate solution but are insufficient as standalone evaluators for this context.

3. **LLM-as-a-Judge (Generative Evaluation Metrics):** LLM-as-a-judge approaches are recent evaluation methods that use large language models to directly assess generated outputs [30]. Unlike statistical and model-based scorers, they do not rely on reference texts or embedding similarity.

[2] `https://en.wikipedia.org/wiki/Levenshtein_distance`

Table 5.1: Literature synthesis of LLM evaluation metrics categorized into statistical scorers, model-based scorers, and LLM-as-a-judge approaches.

	Metric	**How it works**	**Drawbacks**	**Ideal Use Case**
Statistical Scorers	BLEU[12]	Measures n-gram precision overlap between generated and reference text	Ignores semantics; penalizes valid paraphrases; weak correlation with human judgment	Machine translation, structured outputs
	ROUGE[13]	Measures recall-based overlap with reference text	Rewards verbosity; ignores meaning	Text summarization
	METEOR [36]	Combines precision and recall with synonym matching	Limited semantic understanding	Translation, paraphrasing tasks
	Levenshtein (Edit Distance)[1]	Computes minimum edits between two texts	Ignores semantics completely	Structured output comparison
Model-based Scorers	BERTScore [37]	Uses contextual embeddings to measure semantic similarity	Computationally expensive; sensitive to phrasing	Summarization, paraphrased outputs
	MoverScore [38]	Measures semantic similarity using embedding distance	Complex and slow	Long text comparison
	BLEURT [39]	Learned metric trained on human judgments	Requires training data; black-box behavior	General NLG evaluation
LLM-as-a-Judge	G-Eval [40]	Uses GPT-4 with structured prompts and CoT reasoning to evaluate outputs	Bias toward LLM-generated text and reproducibility issues	Complex reasoning tasks
	GPTScore [41]	Uses LLM likelihood to score outputs	Model-dependent; costly	General-purpose evaluation
	MT-Bench [42]	Assigns scores based on evaluation criteria	Inconsistent and non-deterministic	Chatbots, QA systems
	Pairwise Comparison [43]	Compares two outputs and selects the better one	Relative scoring only	Model benchmarking

Instead, they assess outputs using structured prompts and frameworks to evaluate reasoning, correctness, clarity, and other qualitative criteria.

The literature identifies several LLM-based evaluation strategies: G-Eval [40], GPTScore [41], MT-Bench [42], and Pairwise Comparison [43]. G-Eval uses structured prompts and chain-of-thought reasoning to generate scores against predefined criteria, enabling detailed, explainable assessments. GPTScore uses the language model's likelihood scores to provide a probabilistic quality measure. MT-Bench scores performance on multi-turn tasks using set criteria. Pairwise Comparison contrasts two outputs and selects the better one, rather than using absolute scores.

LLM-as-a-judge methods offer advantages but also pose challenges: potential bias toward LLM-generated text, reproducibility concerns from non-deterministic outputs, and higher computational costs than traditional metrics. Still, these methods suit complex reasoning tasks, where semantic understanding and reasoning quality matter more than lexical similarity.

After reviewing the literature and the requirements for evaluating an LLM-based log analysis tool, **G-Eval** and **GPTScore** were selected as the main evaluation metrics for the LLM-based log analysis tool in this study. These methods offer absolute scoring, support reasoning-based evaluation, and assess outputs without reference texts. As such, they suit log analysis outputs where there are no ground-truth answers and evaluation must focus on reasoning quality, such as correctness, completeness, clarity, actionability, and other qualitative criteria.

5.1.2 Thematic Findings: Human-Centered Evaluation Metrics (RQ1.2)

This section presents the thematic analysis results from the semi-structured interviews we conducted on the main perspectives and metrics practitioners use to evaluate LLM-based log analysis outputs. The semi-structured interview was conducted to identify the human oracles, perspectives, and metrics they use to judge the LLM-based log analysis outputs. Using thematic analysis, we identified **6** themes and **13** sub-themes categorized under each theme as shown in **Figures 5.1 and 5.2**. The identified themes, with explanatory descriptions, are presented in **Table 5.2**, and their corresponding sub-themes are also described in **Appendix D.1**. The findings and participants' key points are discussed below, and each theme discussion includes sub-themes and illustrative quotes from participants.

Table 5.2: The six main themes identified in the thematic analysis. Each theme is complemented with a short description.

Theme	Description
Spotting Exact Error and Location	Ability to accurately identify the root error within noisy logs and pinpoint its location.
Trustworthiness of Explanations	Degree to which explanations are clear, complete, and reliable.
Handling Complexity and Ambiguity	Effectiveness in dealing with complex, ambiguous, or uncertain log data. Mainly, how it helps with cascading failures, especially those that emerge from different sources.
Output Design and User Interaction	Usability and clarity of the tool's output and interaction design.
Reliability and Long-Term Trust	Consistency, and the tool's dependability over time.
Relevance of the tool	How well the tool supports real-world CI debugging workflows and troubleshooting needs.

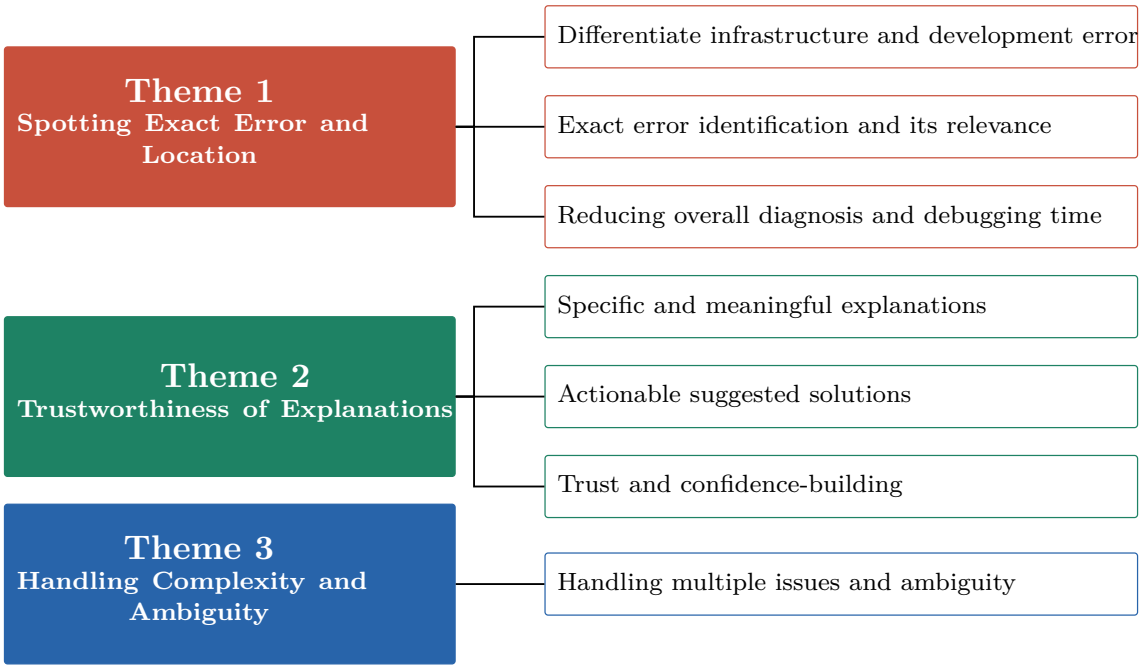


Figure 5.1: Thematic map of human-centered evaluation criteria (metrics) used by CI practitioners to assess LLM-based log analysis outputs (Part 1/2).

[2]https://en.wikipedia.org/wiki/Levenshtein_distance

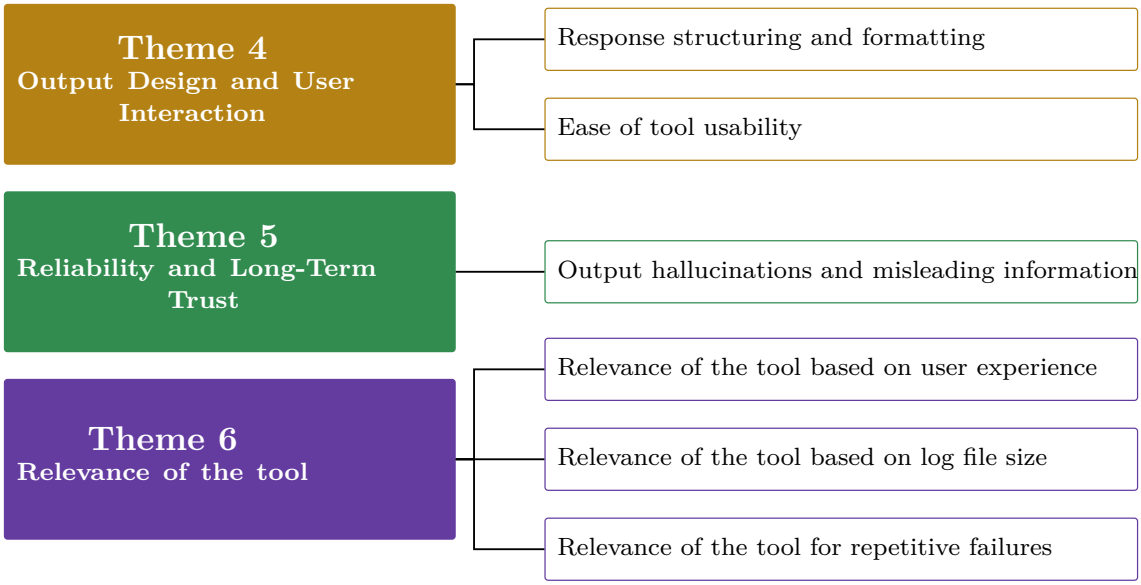


Figure 5.2: Continuation of the thematic map illustrating human-centered evaluation criteria (metrics) and corresponding sub-themes (Part 2/2).

5.1.2.1 Spotting exact error and location

This theme shows that all participants strongly emphasized the importance of pinpointing the exact error and its location, particularly in large CI logs where multiple cascading errors occur. A key expectation is the **ability to differentiate between infrastructure and development errors**, since mixing both leads to confusion during debugging. One practitioner noted that infrastructure issues (e.g., memory, network) and code-related failures should not be presented together without separation, as it obscures the real cause.

> " It is important to distinguish between infrastructure-related failures and code or test failures. These represent different scenarios and should be identified separately to accurately understand the cause of a CI pipeline failure."
>
> **Participant 3**

> " The first thing I want to know is whether it is a build error in the code itself or something failing in the pipeline, such as a specific glitch. I want to know where the most probable cause is."
>
> **Participant 8**

Another participant's recurring requirement is **exact error identification and its relevance**, in which the tool must highlight the initial root cause rather than list all errors. Participants explicitly mentioned that many errors are secondary and caused by an earlier failure, and that the tool should identify the root cause amid the noise.

> “ It is important to correctly identify the actual error, since CI logs contain a lot of noise, including non-critical warnings, debug prints, and irrelevant system messages. The tool should clearly pinpoint the real error and indicate where it occurs, which is the main requirement.”
>
> **Participant 1**

> “ It is always useful if the LLM can create a simple list of findings, clearly showing what the error is and what to try next. It should also clearly separate the different errors in the log, so that it is easier to evaluate whether the proposed solution actually makes sense.”
>
> **Participant 9**

> “ There is usually one initial error that triggers a chain of subsequent errors. However, in the logs, what we mostly see is the final error message. The actual cause of the problem typically appears earlier in the log, before the reported error, and that earlier issue is what leads to the later failures.”
>
> **Participant 4**

One more important point from the participant responses is the need to **reduce overall diagnosis and debugging time** required to diagnose and fix pipeline failures. Participants noted that CI logs are often very large and difficult to navigate manually, making error identification time-consuming.

One participant emphasized that the main purpose of using LLM-based log analysis in pipelines is to reduce the effort required to locate and fix errors. Another participant pointed out the difficulty of working with large log files, noting that they are very large and that it is hard to pinpoint the exact point at which the build failed.

> “For me, the main purpose of using an LLM-based log analysis tool is to receive a notification about where the issue occurred, along with possible fixes, so that I do not have to manually go through logs or search online for solutions.”
>
> **Participant 1**

> “ The logs are often very large, making it difficult to identify the exact point at which the build has failed. The tool is useful if it can pinpoint the precise error and provide guidance, as this helps in understanding both the nature of the error and its potential root cause.”
>
> **Participant 4**

5.1.2.2 Trustworthiness of Explanations

This theme explores participants’ thoughts on the explanations of LLM-based log analysis output and their trustworthiness. Most participants emphasized that beautiful but general explanations are pointless and that explanations should be more specific and to the point. They also pointed out that the explanations should give actionable points, not general suggestions.

Participants highlighted that they evaluate the trustworthiness of explanations by whether the outputs are **specific and meaningful**, rather than generic summaries. They expect explanations to align with their own understanding and debugging intuition. Explanations that are too general, vague, or overly descriptive without referring to the actual log content reduce confidence in the tool. Instead, participants expect explanations that connect the error to specific log lines, possible causes, and relevant context.

> "I would first check if the explanation is too general. If it only gives generic answers without log-specific information, then it is not very useful. I would expect it to point to possible connections and provide meaningful, actionable hints."
>
> **Participant 1**

> " Having the evidence would make me confident in it."
>
> **Participant 2**

> " When you have worked on a project for a long time, you have seen many errors before, so based on experience, you can often judge whether the explanation seems reasonable."
>
> **Participant 7**

Participants emphasized that trust is not immediate; rather, it builds over repeated use and validation of the tool. They explained that **trust and confidence-building** develop when the tool consistently delivers accurate results and helps reduce debugging time. Over time, this repeated successful use increases confidence in the system and its explanations.

> "The error and the solution should match. If the tool provides a precise solution to the exact error, I can trust it. But if different errors and solutions are mixed together, it becomes confusing, and I end up cross-verifying with other sources."
>
> **Participant 3**

> " By using it consistently, I can build trust over time. If we align on our goals, that trust can grow."
>
> **Participant 6**

> "If the explanation aligns with how I would analyze the issue myself, then I feel there is some truth behind it. But if it clearly contradicts my expectations, I start distrusting it and ask more specific follow-up questions"
>
> **Participant 9**

Another practitioner highlighted the need for validation through experimentation and cross-checking.

> "I usually try out the solutions it provides and verify them using Google, Copilot, ChatGPT, or other tools. I also test the solutions directly in my source code or on my local machine to see if they actually fix the issue."
>
> **Participant 4**

Additionally, all participants emphasized that the presence of **actionable suggested solutions** is critical for building trust in the log analysis tool. They highlighted their expectation that concrete fixes and validation steps should be included in the tool's output. Specifically, participants wanted explanations to include specific fixes, validation steps, or suggested workflows, such as running tests, creating commits, or verifying specific log lines. The participants highlighted that these actionable steps are essential, as they help them determine whether they have sufficient information to start fixing and take appropriate action.

> " I want a concrete solution linked to a specific line. Even if it is not the exact error, matching the line with a reasonable fix gives me initial confidence and saves me from having to scan the entire log. The fix can be general, but it should fit the selected line, such as a missing library or compile error, rather than something overly generic."
>
> **Participant 1**

> " The tool could suggest immediate or first-level fixes, such as updating the Python version or resolving environment issues. It could also provide validation steps or commands to check whether the issue is fixed. For complex issues caused by code changes, especially in C or C++ files, it may be difficult to suggest exact fixes. However, for simpler problems like file-not-found errors, I expect at least a basic solution."
>
> **Participant 6**

Participants underscored that incorrect, misleading, or poorly matched explanations significantly reduce trust in the system. They also noted that if the explanation points to the wrong error line or provides mismatched solutions, they would lose confidence and revert to manual log scanning. Therefore, accuracy, specificity, and actionable guidance were identified as the main factors influencing the trustworthiness of LLM-based explanations of log analysis. A participant highlighted their experience regarding the LLM-based outputs as helpful and utter nonsense, depending on their actionability.

> " Sometimes the explanation is good enough for me, and sometimes it feels like it is utter nonsense. What really matters is whether the explanation is actionable and helps me move towards fixing the issue. "
>
> **Participant 4**

5.1.2.3 Handling Complexity and Ambiguity

This theme explores participants' perceptions of how the LLM-based log analysis tool handles **complexities and ambiguities**. Participants highlighted that the

tool should be able to handle ambiguous situations, particularly when multiple errors or root causes are present. In such scenarios, structured explanations such as ranking the influence of errors, indicating probabilities, prioritising issues, or showing possible error trajectories are considered important in CI environments, where failures often involve multiple interdependent issues. Practitioners, therefore, expect tools to explicitly address **handling multiple issues and ambiguity** rather than presenting flat summaries or single-cause explanations.

A participant emphasized the potential confusion caused by multiple error lines in CI logs and highlighted the need for prioritization to identify the most critical failure:

> " If the tool shows many errors, it can confuse the CI engineer because the real cause still needs to be identified. Instead, it should prioritize and rank errors by likelihood, highlighting the most probable cause of the build failure. For example, even if errors appear at lines 35, 37, 87, or 91, the failure at 87 or 91 is often more relevant since the build continues until then."
>
> **Participant 3**

> " Ranking is useful because it tells you what is the most likely cause. But underneath that, it should also show that it could possibly be this or that as well."
>
> **Participant 7**

> " Usually, it is useful when the LLM gives a ranking or a few different proposals for what should be tried first. That is often the best approach, because then I can quickly judge which option makes the most sense to me and try that one first."
>
> **Participant 9**

Another participant pointed out the importance of providing either a full explanation of multiple root causes or a structured method to indicate relevance, along with sequence tracking and confidence levels:

> " If multiple root causes are detected, I would expect the tool to either explain all of them so I can filter them or prioritize the most relevant one. It would also help if it showed a sequence of failures leading to the final error, like a log trajectory. Additionally, a confidence level based on past errors or a knowledge base would improve trust in the result."
>
> **Participant 1**

One respondent remarked that key expectations are the ability to rank or prioritize causes and provide guidance for next steps, especially when ambiguity is present:

> "Some prioritization would be helpful, such as indicating which errors are most likely. It would also be useful if the tool could suggest fixes, next steps, or hints for further investigation when the issue is ambiguous."
>
> **Participant 2**

A different participant highlighted the distinction between critical errors that immediately stop the build and less critical issues, emphasizing that the tool should identify and prioritize the main causes:

> “ Some errors, such as unit test failures, immediately stop the build and should be identified as the main cause of failure, while less critical issues that do not stop the build can be tolerated”
>
> **Participant 6**

Overall, participants emphasized that LLM-based log analysis tools should handle multiple errors and root causes with clear prioritization and guidance. Critical failures need to be distinguished from less urgent issues, and the tool should provide actionable suggestions, confidence levels, and error trajectories. These capabilities help practitioners quickly identify the most relevant causes and make informed debugging decisions.

5.1.2.4 Output Design and User Interaction

This theme explores how practitioners perceive the **output design and user interaction** of LLM-based log analysis tools, particularly focusing on how the structure and presentation of responses influence usability. Across participants, there was a clear consensus that response structuring and formatting are critical for efficient debugging, as poorly structured outputs increase cognitive load and slow down diagnosis.

Most practitioners expressed a strong preference for structured responses rather than flat summaries. They described an ideal output format that combines a direct answer identifying the root cause, a concise summary, and, optionally, a step-by-step explanation for deeper understanding. This structure allows them to quickly identify the issue and, if needed, explore additional context.

As one practitioner stated:

> “ For me, the most important thing is to first get a direct answer that clearly points out where the issue went wrong. After that, a brief summary is helpful to understand the overall context. Finally, step-by-step reasoning can provide deeper insight into how the problem occurred. This kind of structured output would be most helpful. ”
>
> **Participant 3**

> “First, I would want a summary and a direct answer. Step-by-step reasoning can come later if I want to explore further, but it does not need to appear immediately.”
>
> **Participant 7**

Participants highlighted that the importance of structure becomes even more evident in complex or ambiguous scenarios, where multiple factors may contribute to a failure. They pointed out that, in such cases, a simple summary is insufficient, and

practitioners rely on step-by-step reasoning to understand how the failure evolved and which components are involved.

> “ In some cases, there is ambiguity, especially when there are dependencies between different patches. In such situations, I would prefer step-by-step reasoning to understand the issue in depth. However, for straightforward cases, a direct answer is sufficient. ”
>
> **Participant 6**

The participants emphasized that they expect the tool to adapt its level of detail depending on the complexity of the issue, providing concise outputs for simple cases and more detailed reasoning when ambiguity is present.

In addition to formatting, participants emphasized that ease of use is an important factor, particularly in how they interact with the system. All participants showed a clear preference for interactive, chat-based interfaces over static, single-shot summaries. They highlighted that interactive systems allow them to iteratively refine their queries, clarify uncertainties, and obtain more precise explanations. As one participant explained:

> “ A chatbot interface is definitely preferable, as it allows for providing additional prompts and obtaining more precise information. ”
>
> **Participant 1**

> “ I think the one-time summary is already very useful, especially compared to manually searching through huge logs. But having more interaction could also be a good addition. ”
>
> **Participant 8**

> “ If I notice from the summary that the LLM is already moving in the wrong direction, I would rather guide it toward the correct direction and then ask for a proposal on how to fix it.”
>
> **Participant 9**

Participants also stated that the tool is easier to use when it fits smoothly into their existing development environments, like IDEs or CI pipelines. They highlighted that, tools that easily integrate into existing workflows help them reduce context switching and make it easier for them to use the LLM-based help in their daily activities and debugging.

In summary, the participants emphasized that well-structured outputs and interactive user interfaces are critical for usability. They underlined that these features enable them to efficiently interpret results, address ambiguity, and iteratively refine the LLM-based output to understand the CI pipeline failures.

5.1.2.5 Reliability and Long-Term Trust

This theme examines practitioners' views on the reliability and enduring trust of LLM-based log analysis tools. In interviews, participants stressed that trust is shaped not by isolated accuracy, but by consistent performance over time. They also emphasized that, consistent performance through repeated use is vital for lasting adoption.

The participants also underlined that output hallucinations and misleading information from LLM-based tools reduce tool reliability and long-term trust. They stated that the tool sometimes provides explanations that are wrong, incomplete, or not relevant to the actual log context. They said they sometimes noticed missing context, confident but incorrect explanations, and inconsistent results between tools or queries. They said these issues make it harder to trust the tool, and users have to spend more time checking the results.

One practitioner highlighted the need for cross-verification due to such inconsistencies:

> " Sometimes it gives misleading information, so we need to cross-verify it with other tools. "
>
> **Participant 6**

> " The main thing is that you should always be cautious about what you get and evaluate it yourself. "
>
> **Participant 8**

Participants stated that they do not rely only on the tool. Instead, they use it as a support and check its results with other tools, such as different LLMs, search engines, or internal tools.

They also pointed out that seeing misleading results again and again can hurt their long-term trust in the tool.

> " If it keeps misleading me, then I would probably stop using the tool."
>
> **Participant 6**

> " One issue I have seen is that the tool sometimes keeps repeating the same clearly wrong proposals, even when I try to guide it in the right direction, which makes it difficult to trust."
>
> **Participant 9**

Overall, participants stated they can accept occasional errors if the tool is usually helpful. They said that if it often makes debugging easier or helps speed up diagnosis, some flaws are acceptable. They highlighted that long-term trust depends on how reliable and useful the tool is; if it helps most of the time, they can overlook small mistakes. However, if it keeps giving wrong results, they might lose trust, they said.

5.1.2.6 Relevance of the tool

This theme analyzes how practitioners view the practical value of LLM-based log analysis tools in their daily CI workflows. Rather than judging the tool on its own, participants considered how it fits into their existing daily routines, the problems they encounter, and the real support it offers.

One common point in the interviews was how well the **tool can help with repetitive problems**. Participants stated that many CI failures follow familiar patterns. They stressed that it would be helpful if the tool could quickly identify known issues from history and provide their corresponding solutions, thereby reducing the need to investigate the same problems repeatedly.

As one participant described it:

> " Most of the time, we deal with repetitive issues. If the tool can handle those and analyze them automatically, it can significantly reduce the diagnosis time. "
>
> **Participant 6**

Another key point is the **size and complexity of CI logs**. Participants noted that logs from CI pipelines are often large and hard to navigate, which makes it difficult to find where things went wrong. In these situations, the tool is especially helpful because it highlights the most relevant parts of the log, they said.

One practitioner explained:

> " In CI, the logs are very large, and it becomes difficult to find the exact point where the build failed. "
>
> **Participant 4**

Participants also highlighted that these tools help them work more efficiently. By summarizing logs and highlighting possible root causes, the tool makes debugging faster and easier to understand.

As one participant noted:

> " It helps in finding the root cause faster and saves time compared to manually going through the logs. "
>
> **Participant 3**

> " I think it speeds up my own analysis, because before we had to search on Stack Overflow or Google, but now I can gather information much faster and continue asking follow-up questions."
>
> **Participant 8**

> "For new errors that I have not seen before, the LLM input can be a very good starting point for further investigation."
>
> **Participant 7**

Practitioners also highlighted that the tool is most relevant when it fits naturally into their existing CI and debugging workflows. Rather than being seen as a separate standalone system, it was viewed as useful when it could support developers during failure diagnosis and help them act more quickly within the development process. As one participant explained:

> “ There is definitely potential for this to become one part of the DevOps loop, where it can help developers by pointing out what broke the build and what needs to be fixed.”
>
> **Participant 9**

Another essential point raised by participants during the interviews is that tools like LLM-based log analysis tools are particularly valuable for junior practitioners compared to more experienced users. They noted that such tools help all practitioners work more efficiently. However, junior practitioners, especially when dealing with large log files, often find it difficult to identify failures and generate solutions quickly. In these situations, participants emphasized that LLM-based tools are especially helpful for juniors.

> “I think these tools are more helpful for junior practitioners than for experienced ones, especially when the log files are large.”
>
> **Participant 4**

In summary, participants indicated that the tool’s usefulness is highly context-dependent rather than universal. They suggested it is most beneficial when dealing with large or complex logs, recurring failures, or situations requiring quick initial insights. Many noted that it is particularly useful for new or unfamiliar errors in complex log files, where manual debugging (e.g., keyword search) can take several hours. They also emphasized its value when it integrates smoothly into existing workflows (e.g., within VS Code) and reduces the need for file uploads and context switching. However, for simpler issues or when high certainty is required, participants preferred to rely on their own analysis.

5.2 Evaluation and Agreement of Automated Metrics and Practitioner Evaluations (RQ2)

This section examines how an LLM-based log analysis tool can be evaluated in practice and assesses the degree of agreement between LLM-as-a-judge metrics and practitioner evaluations to validate their effectiveness.

5.2.1 Evaluation of the Log Analysis Tool Using Technical Metrics

The evaluation of the LLM-based log analysis tool using technical metrics was performed through structured testing. The selected LLM-as-a-judge methods were G-Eval and GPTScore. G-Eval was implemented with the GPT-4o-mini model.

GPTScore was implemented with both GPT-4o-mini and Gemini 2.0 Flash Lite, allowing comparison of scoring across different evaluator models. During the tool selection phase, existing evaluation frameworks such as DeepEval, Braintrust and OpenAI Evals were identified. However, the company's data security policies restrict the use of third-party frameworks, so custom scripts were developed and run locally. Evaluation prompts were designed based on the literature for G-Eval and GPTScore, with prompt templates provided in **Appendix C**.

For G-Eval evaluation, both single-shot and multi-step prompt variants were used. Recent literature shows that multi-step prompting gives more reliable results for reasoning-intensive tasks. The test evaluated the generated log analysis outputs with the defined criteria. Results were recorded and aggregated to compare the methods and models. Finally, **Figure 5.3** summarizes these evaluation results, showing the technical scores from the selected LLM-based methods.

```
1. Example of G-eval Evaluation Result of a single file
{
    "reasoning": "The model output identifies several key errors from the
    log, including Git checkout timeouts, invalid cache files, missing
    containers, XML validation failures, and file not found errors. The
    reasoning for these errors is mostly correct, as it aligns with the
    log signals. However, the model fails to mention the numerous
    warnings present in the log, which are critical for a complete
    analysis. The solutions provided are actionable and relevant,
    addressing the identified issues effectively. The clarity of the
    explanation is good, with a logical structure that makes it easy to
    follow. However, the completeness is affected by the omission of key
    warning signals that could influence system stability. Overall, the
    analysis is mostly correct and actionable but lacks completeness due
    to missing warnings.",
    "scores": {
        "correctness": 4,
        "completeness": 3,
        "clarity": 4,
        "actionability": 4
    }
},
2. Example of GPT-score Evaluation Result for Correctness for 3 Files
[ 4, 4, 4, ]
```

Figure 5.3: Example of G-Eval and GPT-Score evaluation results for technical metrics testing

5.2.2 Quantitative Comparison Between Technical Metrics and Human Evaluation

1. Descriptive Statistics This section provides an overview of how the different LLMs, as judge evaluation methods, assess output quality across key metrics, com-

paring central tendencies and variability with human judgments. We can identify systematic differences between automated approaches and human judgments. The details are presented in **Table 5.3,** followed by a discussion.

Table 5.3: Summary of descriptive statistics across evaluation methods and practitioner evaluation scores

Method	Metric	Mean	Median	Std	Min	Max
Multi-step G-Eval						
	Correctness	3.690	4.0	0.541	3	5
	Completeness	2.690	3.0	0.541	2	4
	Clarity	3.897	4.0	0.409	3	5
	Actionability	3.517	4.0	0.574	2	4
Single-step G-Eval						
	Correctness	3.483	3.0	0.509	3	4
	Completeness	2.759	3.0	0.511	2	4
	Clarity	3.586	4.0	0.501	3	4
	Actionability	3.655	4.0	0.670	2	5
Practitioner (Human) Evaluation						
	Correctness	3.276	3.0	0.960	1	5
	Completeness	3.517	3.0	0.871	2	5
	Clarity	2.966	3.0	0.944	2	5
	Actionability	3.138	3.0	1.093	2	5
GPT-score Evaluation						
	Correctness (GPT-4o-Mini)	4.034	4.0	0.186	4	5
	Correctness (Gemini 2.0 Flash Lite)	3.034	3.0	0.421	2	5

The results show clear differences between multi-step, single-step, and human evaluations. Multi-step G-Eval consistently scores higher in correctness (3.690), clarity (3.897), and actionability (3.517), with low variability (about 0.4 to 0.6), suggesting stable, positive assessments. Single-step G-Eval has slightly lower scores for correctness and clarity but matches or even exceeds multi-step in actionability (3.655), although its results are less consistent (standard deviation 0.670).

Human evaluations show considerably higher variance (standard deviation up to 1.093), with lower scores for clarity (2.966) and correctness (3.276), but higher scores for completeness (3.517), suggesting that human judges are more critical and apply more differentiated assessments across responses. The GPT-score results reveal a contrasting pattern: GPT-4o-Mini yields the highest correctness scores (4.034) with notably low variance (0.186), and even the min and max values span only a narrow range. Rather than indicating reliability, this low variance suggests limited discriminative power; the model assigns similar scores regardless of response

quality, much like a grader who gives everyone the same grade without carefully distinguishing strong from weak answers. Gemini 2.0 Flash Lite produces scores closer to human averages (3.034), but it still exhibits the same tendency toward compressed scoring ranges.

In summary, automated methods, especially multi-step G-Eval and GPT-4o-Mini, tend toward narrow, optimistic score distributions, reducing their ability to distinguish between good and poor responses. Human judgments, by contrast, are more critical and span a wider range, making them better suited to discriminating between response quality levels.

2. Spearman's and Kendall's Rank Correlation Analysis

To evaluate agreement between the LLM-as-a-judge evaluation metrics and practitioners' judgments, we computed paired rank correlations between the scores assigned by the LLM-based evaluators and the corresponding human assessment scores for the same evaluation tasks. We report both the Spearman rank correlation coefficient (ρ) and Kendall's rank correlation coefficient (τ) to provide a more robust assessment of agreement, as the two metrics capture rank consistency differently. Using both measures, therefore, reduces the risk of drawing conclusions from a single correlation metric and increases confidence in the consistency of the observed relationships. The detailed results are presented in **Table 5.4** and **Figures 5.4 and 5.5**, followed by a detailed discussion.

Table 5.4: Summary of Spearman (ρ) and Kendall's Tau (τ) correlations between LLM-as-a-Judge evaluation methods and practitioners' judgments across different evaluation metrics.

Method	**Model**	**Metric**	**Spearman (ρ)**	**Kendall (τ)**
Multi-step G-Eval	GPT-4o-Mini	Correctness	0.210	0.191
		Completeness	0.237	0.221
		Clarity	0.168	0.155
		Actionability	0.097	0.089
Single-step G-Eval	GPT-4o-Mini	Correctness	0.219	0.202
		Completeness	0.305	0.276
		Clarity	-0.169	-0.158
		Actionability	0.089	0.085
GPT-score	GPT-4o-mini	Correctness	0.312	0.288
	Gemini 2.0	Correctness	0.092	0.079

Note: **NA** in the below Heatmap denotes the evaluation criteria that were not assessed. GPT-score was evaluated only for **correctness**; therefore, the correlation values for completeness, clarity, and actionability are not applicable.

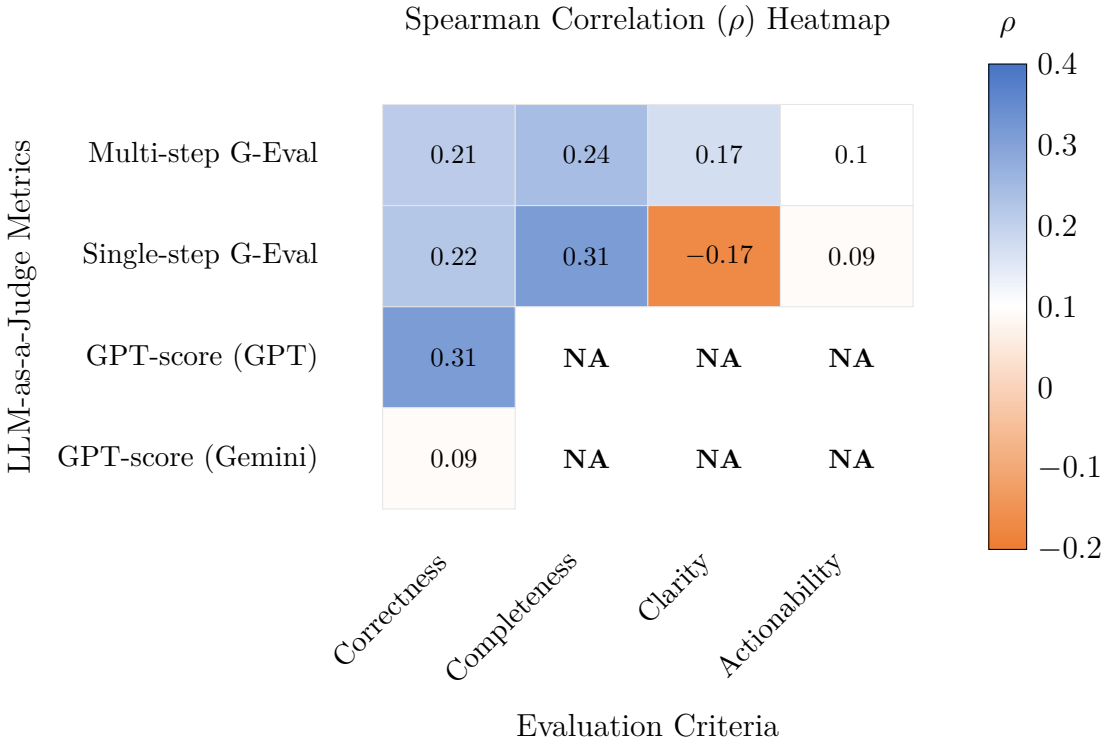


Figure 5.4: Spearman correlation (ρ) matrix showing the relationship between LLM-as-a-Judge Metrics and human judgments.

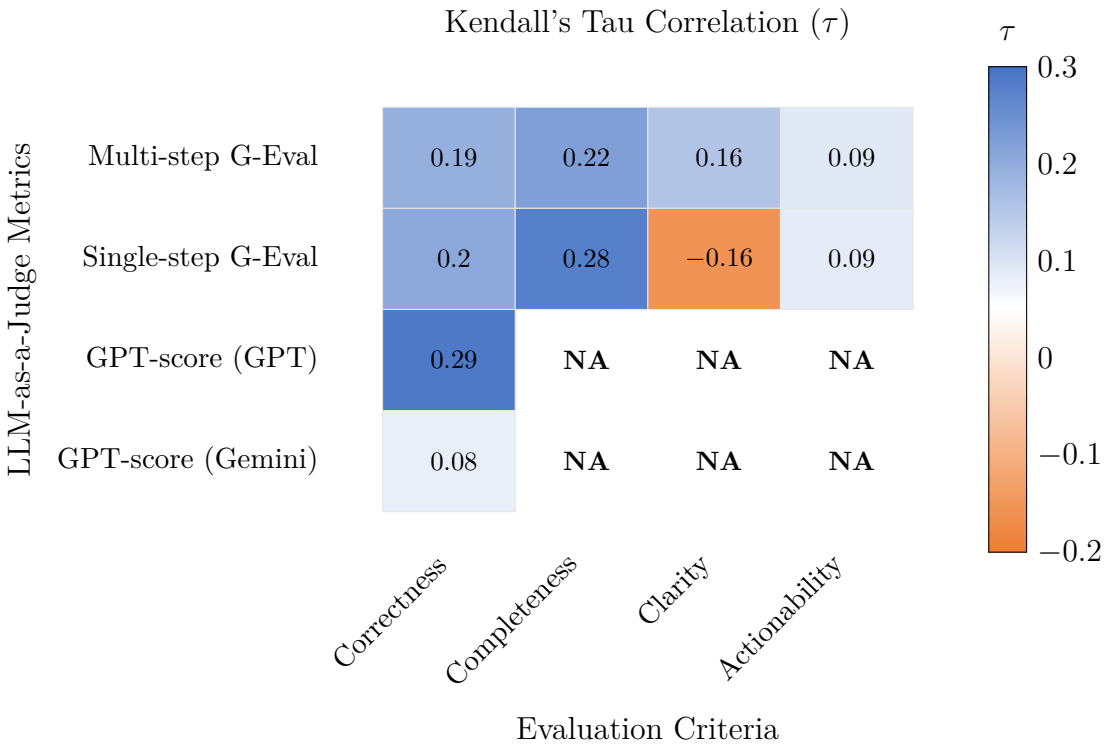


Figure 5.5: Kendall's Tau correlation (τ) matrix showing the relationship between LLM-as-a-Judge Metrics and human judgments.

The results show very weak to weak agreement between LLM-based evaluations and human practitioner judgments, based on the interpretation thresholds proposed by Evans [45], where 0.00–0.19 indicates very weak correlation, 0.20–0.39 weak, 0.40–0.59 moderate, 0.60–0.79 strong, and 0.80–1.00 very strong. Single-step G-Eval has slightly higher correlations for completeness. The multi-step variant is more consistent but shows generally lower correlations. Notably, clarity has a negative correlation in the single-step setting. This suggests misalignment with human perception. For GPT-score approaches, GPT-4o-mini outperforms Gemini 2.0 Flash Lite. However, overall agreement remains weak. Automated evaluation cannot fully substitute human judgment.

In comparison to prior work, Liu et al. [40] report substantially higher correlations (up to $\rho = 0.582$ for coherence), which is moderate. However, their focus was on structured natural language generation tasks, such as summarization, which contrasts with our setting of log analysis, which involves noisy, unstructured data and requires complex reasoning and root-cause identification. This difference in task complexity likely explains the lower correlations observed.

A further source of discrepancy lies in differing evaluation perspectives. Human practitioners primarily assess whether the main root cause is correctly identified and whether the solution is actionable. In contrast, LLM-based evaluators tend to reward broader coverage of issues, including less relevant log signals. We believe that this divergence between precision and actionability, and between coverage and completeness, contributes to ranking inconsistencies.

Finally, the limited sample size ($n = 29$), despite the large individual log files, reduces the statistical power and stability of the correlation estimates. Consequently, the observed correlations may partially reflect dataset constraints rather than solely limitations of the evaluation methods.

5.3 Thematic Findings: Practitioner Perceptions of the LLM-Based Log Analysis Tool Influence on CI Workflows (RQ3)

This section presents the inherent manual debugging challenges as an introduction and thematic analysis results from semi-structured interviews investigating practitioners' perceptions of the influence of an LLM-based log analysis tool on CI workflows in industrial CI logs. The analysis focuses on practitioners' perceptions and experiences with debugging efficiency, root cause analysis, and overall CI pipeline management when using the tool. Using thematic analysis, we identified **7** themes and **20** sub-themes capturing perceived workflow influences, as shown in **Figure 5.6**. The identified themes, with explanatory descriptions, are presented in **Table 5.5**, and their corresponding sub-themes are detailed in **Appendix D.2**.

5.3.1 Current CI Troubleshooting Practices and Challenges

The findings of the interviews indicate that existing CI log analysis remains fundamentally **manual, heuristic-driven, and cognitively demanding**, with debugging practices that are highly dependent on experience and ad hoc strategies such as keyword searches, backtracking, and visual inspection. The interviews revealed that debugging commonly starts with manually scanning logs and filtering error-related keywords while reasoning about recent code or configuration changes. A recurring challenge identified in the interviews was the **difficulty in identifying the actual root cause**, particularly when failures are indirect, hidden, or propagated between dependent components. The findings further show that the **scale and complexity of CI logs**, especially in large distributed systems, make the process more demanding, requiring developers to inspect thousands of log lines across multiple systems. As some of the practitioners explained:

> "The logs contain too many lines and become very complex, so you need to inspect logs from multiple systems to understand the full picture."
>
> **Participant 14**

> "Failures often occur as a chain reaction, where one component failure triggers another downstream, making root cause analysis difficult."
>
> **Participant 1**

Together, these accounts show that CI debugging is highly complex, time-intensive, and strongly dependent on human expertise.

5.3.2 Thematic Results

This section presents the thematic analysis findings on the tool's perceived usefulness and its influence on CI workflows as shown in **Table 5.5** and **Figure 5.6**.

Table 5.5: The seven main themes identified in the thematic analysis for RQ3. Each theme is complemented by a short description.

Name	Description
Efficiency Gains through Tool-Assisted Analysis	Explains how the tool reduces debugging effort and accelerates failure analysis.
Transformation of Debugging Workflows	Explains workflow changes from manual debugging to interactive AI-assisted analysis.
Influence on Decision-Making and Error Exploration	Shows how the tool affects root-cause prioritization and debugging decisions.
Integration & Workflow Fit	How the tool integrates into existing CI workflows.
Contextual Utility and Usage Patterns	Presents the practical usage patterns and contextual utility of the tool.
Risks and Unintended Consequences	Highlights the risks, limitations, and unintended consequences associated with the tool's usage.
Influence on Learning & Expertise Development	Discusses how the tool influences learning, expertise, and skill development.

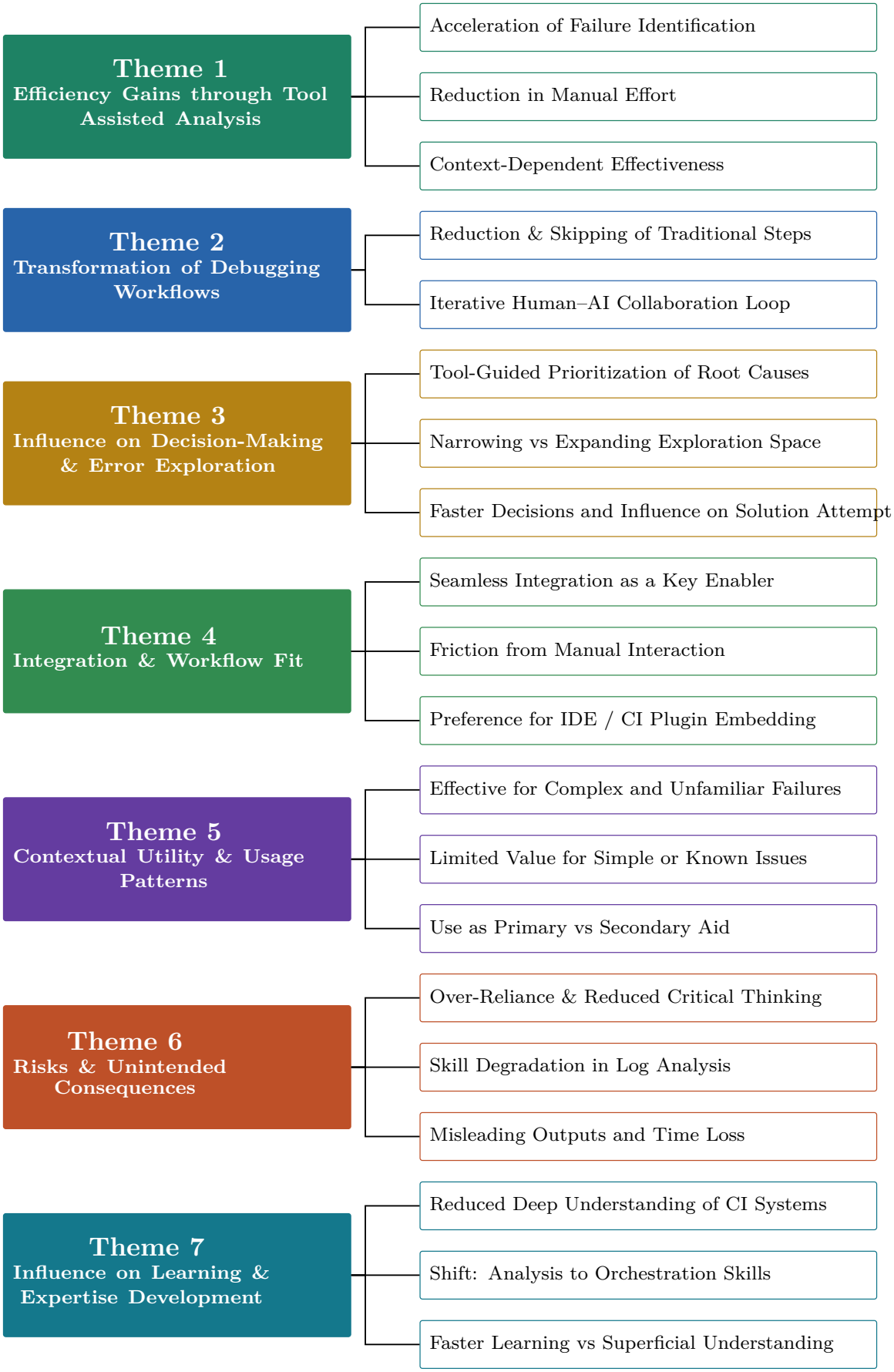


Figure 5.6: Thematic map of perceived CI workflow influences associated with the use of the LLM-based log analysis tool in industrial CI environments.

5.3.2.1 Efficiency Gains through Tool-Assisted Analysis

Participants perceived the LLM-based tool as improving debugging efficiency by **accelerating failure identification** and **reducing manual effort**. Practitioner responses indicate that the tool provides immediate hints, identifies relevant log sections, and narrows the search space, reducing the need for exhaustive log scanning. They also noted that the **efficiency gains are context-dependent**, with the tool being particularly valuable in complex or unfamiliar scenarios while offering more limited benefits for simple or recurring issues. For instance, one participant explained that:

> "The tool pinpoints where to look, so instead of spending 20 minutes, it might do it in two."
>
> **Participant 10**

Similarly, another participant estimated that:

> "It could reduce the debugging effort quite a lot, maybe by 75%."
>
> **Participant 7**

In contrast, another participant described more moderate improvements in simpler cases, stating that:

> "I probably save a few minutes per log because most errors are quite common."
>
> **Participant 13**

Together, these accounts suggest that the tool enhances debugging efficiency, although its influence varies depending on task complexity and practitioner expertise.

5.3.2.2 Transformation of Debugging Workflows

The findings show that participants perceived the LLM-based tool as shifting debugging workflows from a manual, linear process into a more interactive, AI-assisted one. They also stated that traditional debugging often involves sequential activities such as searching logs, filtering errors, and manually iterating through failures, whereas the tool enables a more **guided and iterative interaction**. The participants also noted that activities such as manually locating failures and scanning entire log files are reduced by the tool's root-cause highlighting, summary descriptions, solution suggestions, and hover-over explanations for log lines. They further described the workflow as a **human-AI collaboration loop** centered on suggestion, validation, and refinement. As one participant explained:

> "Instead of searching through Google, you use the tool, and usually the process becomes faster and more effective."
>
> **Participant 2**

Similarly, another participant described it as:

> "Debugging becomes a continuous loop where I first use the tool, then validate the results, and analyze further if needed."
>
> **Participant 12**

This shows that while workflows become more efficient, they also become more interactive and tool-mediated, rather than purely manual.

5.3.2.3 Influence on Decision-Making and Problem Exploration

Practitioners reported that the tool could influence how developers prioritize, explore, and act on potential root causes, effectively reshaping decision-making processes. Several participants also stated that the ranked and highlighted suggestions enable **tool-guided prioritization**, often leading developers to focus on specific causes early in the debugging process. They also noted that this **narrows the exploration space**, leading to fewer alternative hypotheses being considered than in manual debugging, though in some situations the tool expands exploration by presenting additional perspectives. They further explained that the tool affects which **solution is attempted first**, as developers tend to prioritize tool-suggested fixes before testing their own hypotheses. As a practitioner explained:

> "Initially, I prioritize the tool suggestion because it identifies the most probable cause more quickly."
>
> **Participant 5**

Similarly, another participant noted that:

> "I would directly follow what the tool mentions, which means I might miss other possible problems."
>
> **Participant 1**

This indicates that while decision-making becomes faster, it may also become more biased toward tool output.

5.3.2.4 Integration and Workflow Fit

The findings show that the tool's effectiveness and adoption are strongly influenced by how well it integrates into existing workflows. Participants highlighted that seamless integration within CI pipelines and development environments, such as IDEs, is critical for usability and workflow continuity. They also noted that when the tool requires manual actions such as copying or uploading logs, it introduces friction and interrupts the debugging process. In contrast, automation and embedding the tool within existing systems were described as factors that significantly improve adoption and usability. As one practitioner explained:

> "If you have to manually upload the logs, there is still some friction in the workflow."
>
> **Participant 2**

Similarly, another participant pointed out that:

> "I still have to copy and paste many things, so the tool could be more integrated."
> **Participant 13**

These findings suggest that integration is not only a usability concern but also a key factor influencing the practical effectiveness of the tool.

5.3.2.5 Contextual Utility and Usage Patterns

Participants stated that the tool's usefulness is highly context-dependent, leading to varied usage patterns among practitioners. They emphasized that the tool is most valuable for complex, large-scale, unfamiliar, or time-consuming failures where manual analysis becomes difficult. In contrast, participants also noted that it provides limited benefit for simple, repetitive, or well-understood issues, where developers can quickly identify the root cause themselves. As a result, practitioners described using the tool as a primary aid, secondary aid, or conditional support depending on the failure context and workflow integration. A participant explained that:

> "For failures I haven't seen before, it's really useful to explore possible causes, but for issues I already know, it doesn't really give additional benefit"
> **Participant 9**

Similarly, another participant described it as situational support, stating that:

> "You first try to understand the issue yourself, and if it becomes time-consuming, then the tool becomes useful"
> **Participant 2**

The participants also stressed that the tool is particularly effective for large logs, complex systems, and quick issue investigation, where it can provide fast overviews, alternative perspectives, and possible root causes. Overall, the findings suggest that the tool complements existing debugging practices rather than replacing developers' own expertise and investigation workflows.

5.3.2.6 Risks and Unintended Consequences

Participants emphasized that while the tool improves efficiency, it also introduces several risks and unintended consequences, particularly related to cognitive dependency and inaccurate outputs. A major concern was over-reliance, in which developers may depend too heavily on the tool and gradually reduce their own engagement in manual reasoning and debugging. They highlighted that this could negatively affect developers' analytical and log investigation skills over time, especially for junior engineers. A participant expressed this concern directly, stating that:

> "I'm worried that relying too much on the tool could make developers stop thinking critically and eventually forget how to investigate issues themselves."
> **Participant 13**

Similarly, another participant warned that:

> "If engineers keep asking the LLM for everything, they may gradually lose their own debugging skills."
>
> **Participant 14**

Participants also stressed the risk of hallucinations and misleading suggestions, where incorrect outputs may lead developers down irrelevant debugging paths and increase troubleshooting time rather than reduce it. A participant explained that:

> "People can easily trust the tool too much, even though its suggestions are sometimes incorrect."
>
> **Participant 7**

Another participant also noted that:

> "The tool can identify issues that are not actually errors, which may lead developers to spend time fixing the wrong problem."
>
> **Participant 9**

In addition, participants noted that the tool may struggle in project-specific environments and with internal dependencies. A participant explained that:

> "If the failure is related to internal tools or project-specific components that the model does not understand, the analysis can become inaccurate."
>
> **Participant 2**

Overall, the findings suggest that although the tool provides significant efficiency benefits, its effectiveness still depends on developers critically validating the generated outputs rather than relying on them unconditionally.

5.3.2.7 Influence on Learning and Expertise Development

Participants stated that adopting the tool could significantly influence how developers learn and develop expertise in CI systems over time. On one hand, participants highlighted that the tool can accelerate learning by providing immediate explanations, guidance, and faster issue identification, particularly for less experienced developers. A participant reflected this benefit, stating that:

> "If the tool explains the error clearly, it can help junior developers understand problems and learn faster."
>
> **Participant 2**

At the same time, participants expressed concerns that relying heavily on the tool may reduce developers' engagement with logs and system internals, potentially leading to a more superficial understanding of CI systems and debugging processes. Several participants stressed that continuous dependence on automated analysis could weaken developers' manual investigation and critical thinking skills. A participant raised this concern strongly, explaining that:

> "The tool can become so capable that developers may stop learning how the CI system actually works under the hood."
>
> **Participant 9**

Similarly, another participant noted that:

> "Young developers may trust the tool too much and miss the experience gained from manual log analysis."
>
> **Participant 7**

Participants also highlighted that expertise may gradually shift from deep technical debugging toward higher-level tasks such as orchestration, system understanding, and problem decomposition. A participant described this transition by stating that:

> "If the tool handles the difficult parts automatically, developers may lose some low-level skills but focus more on defining and breaking down problems."
>
> **Participant 10**

Overall, the findings suggest that the tool may reshape expertise development by balancing faster learning and productivity gains against reduced depth of technical understanding.

In general, the findings show that CI log analysis in industrial environments is manual, cognitively demanding, and highly dependent on developer expertise, particularly in large and distributed systems. Participants perceived the LLM-based tool as useful for improving debugging efficiency by accelerating failure identification, reducing manual effort, and supporting more interactive AI-assisted workflows, especially for complex and unfamiliar failures. At the same time, participants raised concerns about over-reliance, hallucinations, and reduced critical thinking, noting that excessive reliance on the tool may weaken developers' manual debugging skills and their understanding of CI systems. Overall, the findings suggest that the tool is most effective as a complementary support mechanism that enhances productivity and learning while still requiring developers to critically validate the generated outputs.

5.4 Thematic Findings: Practitioner Perceptions of MCP-Based IDE Integration Compared with the Web-Based Tool (RQ4)

This section presents findings from a thematic analysis of semi-structured interviews examining practitioners' perceptions of integrating an MCP-based agent into the development environment for CI troubleshooting workflows, compared with a standalone web-based log analysis tool. A thematic analysis method is applied as discussed in Section 4.5.2.1, and **5** themes and **10** sub-themes are identified related to workflow continuity, conversational debugging, troubleshooting interaction patterns, and tool integration within development environments. The identified themes

are summarized in **Figure 5.7** and **Table 5.6**, and their corresponding sub-themes are presented in **Appendix D.3**. The following sections discuss these findings and highlight key participant perspectives, with each theme discussion incorporating the relevant sub-themes and representative participant quotations.

Table 5.6: The five main themes identified in the thematic analysis for RQ4. Each theme is complemented with a short description.

Theme	Description
Continuous and Contextual Troubleshooting	Ability of MCP integration to support workflow continuity, reduce context switching, and connect troubleshooting directly with source code activities inside the IDE.
Iterative and Conversational Debugging	Support for iterative troubleshooting through conversational interaction, follow-up questioning, and continuous refinement during debugging.
Different Roles of MCP and Web-Based Tools	Differences in how practitioners use web-based and MCP-based approaches depending on troubleshooting complexity, monitoring needs, and code-level investigation.
Workflow Changes and Complementary Usage	Degree to which MCP integration reduces workflow friction while complementing existing troubleshooting practices rather than replacing them.
Usability and Adoption Factors	Influence of usability, familiarity with existing workflows, and conversational interaction quality on MCP adoption and usage preferences.

5.4.1 Continuous and Contextual Troubleshooting

Participants described the MCP agent as creating a more continuous and context-aware troubleshooting workflow compared to the standalone web-based tool. A recurring observation was that **reduced context switching improves workflow continuity**, with the tool integrated directly into VS Code minimizing interruptions when moving between browsers, logs, and development environments. Instead of repeatedly copying logs into external tools and then returning to the IDE, participants described being able to investigate failures, inspect logs, and modify code within the same workspace. This integrated workflow was perceived as helping practitioners maintain troubleshooting momentum and reduce disruptions during debugging. One participant explained that keeping troubleshooting activities within the IDE helped preserve execution context during debugging and enabled faster iteration between running code, analyzing failures, and evaluating fixes:

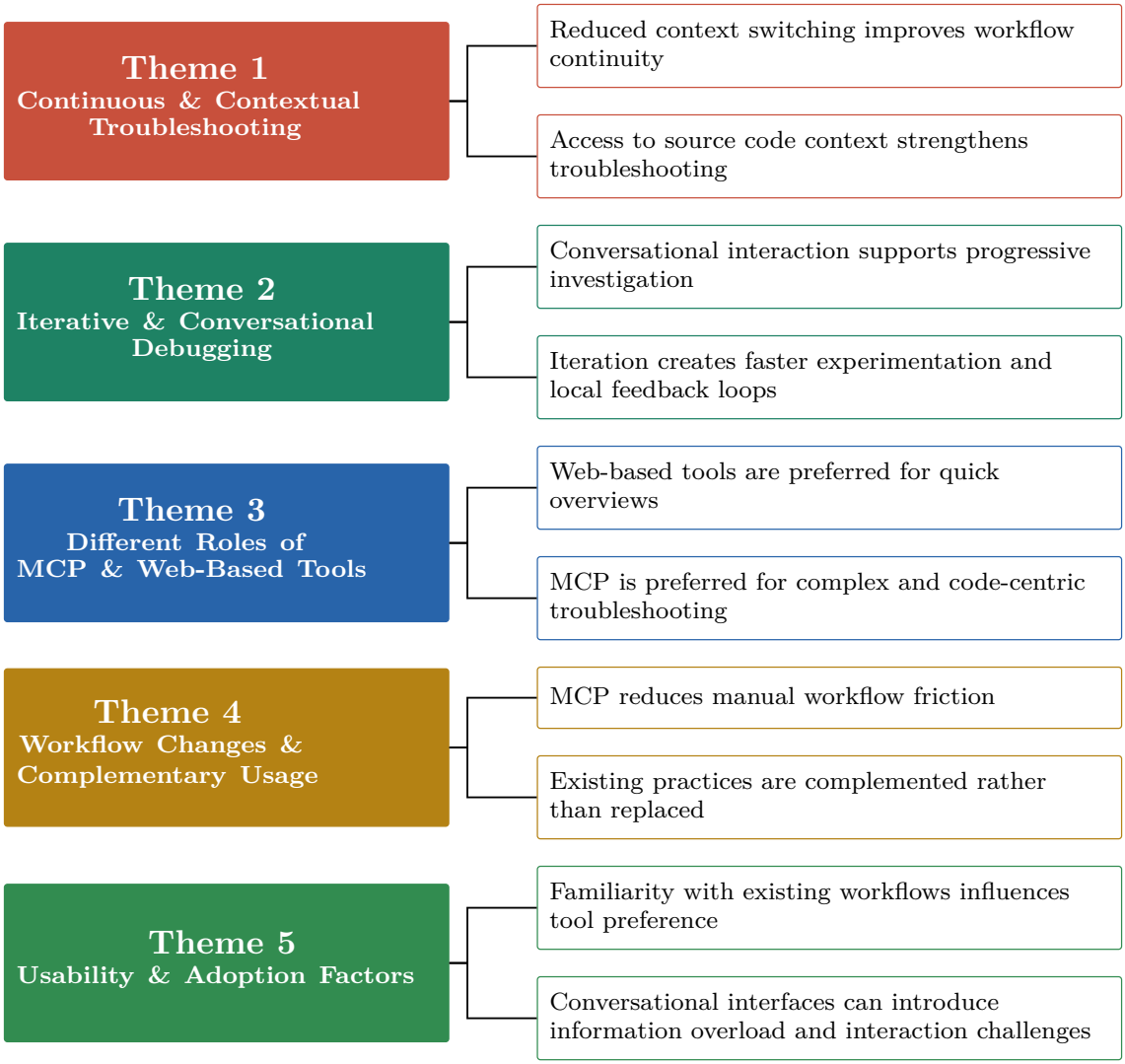


Figure 5.7: Thematic map of CI troubleshooting workflow influences associated with integrating an MCP agent into the development environment, compared to a standalone web-based log analysis tool.

> "If I have to run the code somewhere else, the MCP agent doesn't get the context of what happened during execution. Keeping it inside VS Code allows faster iteration because the agent can run the code, get the output, and evaluate it directly."
>
> **Participant 2**

Similarly, another participant described browser switching as interrupting the troubleshooting flow:

> "Every time you alt-tab to the browser to check what the log said, that's like a break, almost, like a mini break."
>
> **Participant 10**

Participants also highlighted that **access to source code context strengthens**

troubleshooting. The MCP integration enabled closer integration among logs, source code, and debugging activities by running the agent directly in the IDE. This allowed practitioners to connect CI failures to specific implementation details and source code locations while continuing troubleshooting in the same environment. Compared to the standalone web-based tool, which primarily focused on analyzing logs independently, the MCP-based workflow was perceived as supporting a more development-oriented troubleshooting process. One participant explained that the integration enabled direct interaction between the troubleshooting process and source code context:

> "I can ask in which part of my source code the problem has occurred because it has access to my source code as well."
>
> **Participant 4**

Another participant described how the MCP integration became more useful when troubleshooting required actual code modifications:

> "The agent can modify code as needed, because it has more context. The web page can't get that context on the code level."
>
> **Participant 10**

Together, these findings show that integrating MCP directly into the IDE improves workflow continuity by reducing context switching and enabling closer interaction between log analysis, source code, and debugging activities.

5.4.2 Iterative and Conversational Debugging

Participants frequently described the MCP-based workflow as enabling a more conversational and iterative troubleshooting process compared to the standalone web-based interface. A recurring aspect was **conversational interaction supports progressive investigation**, where practitioners explained that the MCP integration facilitated follow-up questioning, clarification, and continuous refinement of troubleshooting hypotheses. Instead of receiving a static, one-time explanation, participants reported progressively narrowing down root causes through dialogue and iterative questioning. One participant explained that the MCP integration changed troubleshooting into a dialogue-oriented interaction:

> "It becomes more like an interactive dialogue within VS Code, whereas the web-based tool feels largely like one-way communication."
>
> **Participant 7**

Similarly, Another participant highlighted the value of asking clarifying questions during debugging:

> "Sometimes things are not that clear, and then you need to ask questions to clarify if it really is what you are thinking."
>
> **Participant 1**

One participant also emphasized that troubleshooting was more effective through iterative interaction rather than relying on a single response:

> "I prefer iterations instead of one shot."
>
> **Participant 12**

Another recurring aspect was **iteration creates faster experimentation and local feedback loops**, in which participants explained that the MCP integration supported repeated cycles of analysis, modification, testing, and re-analysis without leaving the development environment. This iterative workflow was particularly beneficial for complex or unpredictable failures that require multiple refinement cycles before a solution is identified. One participant explained that MCP was particularly useful for challenging troubleshooting situations:

> "For more complex investigations, I would actually prefer using the MCP agent within VS Code."
>
> **Participant 9**

Another participant described how the MCP integration enabled a continuous cycle of proposing fixes, testing them, and refining solutions:

> "This is the error, and then it suggests a solution, and then you can apply the solution directly in code. Then maybe test it again and realize, okay, we should do it differently."
>
> **Participant 2**

Together, these accounts indicate that MCP-based integration supports a more iterative and conversational troubleshooting process, enabling developers to progressively refine investigations and experiment with fixes more efficiently.

5.4.3 Different Roles of MCP and Web-Based Tools

Participants consistently distinguished between situations where the standalone web-based tool and the MCP-based integration were most useful. One recurring aspect was that **web-based tools are preferred for quick overviews and monitoring**, in which participants described the standalone interface as more suitable for lightweight investigations, quick summaries, and high-level monitoring of CI failures. The web interface was often perceived as a faster means to understand issues without engaging in a deeper debugging workflow. Some participants also valued the visual representation and highlighted outputs provided by the web-based interface. One participant explained that the web tool was convenient when only a quick understanding of the issue was needed:

> "If I just want information quickly, the web-based tool might be faster."
>
> **Participant 10**

Another participant distinguished the tools based on their troubleshooting purpose:

> "The MCP-based approach is more suited for day-to-day code-level debugging, whereas the web-based tool is better for high-level overviews."
>
> **Participant 5**

One participant also associated the web interface with monitoring-oriented tasks:

> "The web interface feels more suitable for monitoring tasks, where you just want to check the pipeline results for the day."
>
> **Participant 2**

Concurrently, participants repeatedly emphasized that **the MCP integration is preferred for complex, code-centric troubleshooting**. The MCP integration supported iterative investigation, implementation-level modifications, and direct interaction with the source code. Because the MCP agent operated within VS Code, participants described it as enabling a smoother transition between diagnosing issues and applying fixes. One participant explained that MCP became particularly useful when troubleshooting required source code changes:

> "If I want to make the source code changes also along with the output, then I might prefer the MCP."
>
> **Participant 4**

Another participant differentiated between simple and difficult failures when discussing tool preference:

> "For something simple or a one-liner error, I would use the web-based tool. But for something more difficult, I would use the MCP."
>
> **Participant 9**

Together, these findings show that practitioners use MCP-based and web-based tools for different troubleshooting needs, where web interfaces support quick inspection while MCP integration is preferred for complex and code-centric debugging.

5.4.4 Workflow Changes and Complementary Usage

Participants described the MCP integration as altering troubleshooting practices and reducing workflow friction, although most did not view it as a complete replacement for existing debugging practices. One recurring aspect was that **MCP reduces manual workflow friction**, as participants explained that the integrated workflow minimized repetitive activities, such as switching between browser tabs, copying and pasting logs, manually uploading files, and moving between development and analysis environments. Embedding troubleshooting directly within VS Code simplified parts of the debugging process and reduced interaction overhead. One participant emphasized that the MCP integration reduced the manual effort required to search through large log files:

> "It directly addresses the primary goal and eliminates the tedious manual labor of searching through massive log files."
>
> **Participant 5**

Another participant noted that the interaction process felt simpler and easier to learn:

> "The learning curve might be smaller with MCP."
>
> **Participant 1**

One participant further highlighted that the integrated workflow eliminated the need for manual uploads:

> "One step is eliminated, as the upload stage is no longer required."
>
> **Participant 4**

Participants also consistently emphasized that **existing practices are complemented rather than replaced**. Most practitioners described the MCP integration and the standalone web-based tool as complementary approaches, each suitable for different troubleshooting situations rather than as competing replacements. The web-based tool remained useful for monitoring, lightweight inspection, and quick investigations, while the MCP integration was perceived as more beneficial for iterative and development-oriented debugging tasks. One participant summarized this complementary relationship clearly:

> "They are complementary approaches and are better suited to different troubleshooting situations."
>
> **Participant 7**

Another participant stressed that MCP-agent-assisted troubleshooting is supportive rather than replacing the standalone web tool:

> "I do not see the MCP agent as a replacement; rather, it functions more as an assistant."
>
> **Participant 10**

One participant similarly described the MCP integration as an extension of existing workflows rather than a complete transformation:

> "I view it as a complementary tool, as it does not fundamentally change the existing workflow."
>
> **Participant 13**

Together, these accounts suggest that MCP integration reduces manual workflow friction and streamlines troubleshooting activities, while still being perceived as complementary to existing debugging practices rather than a full replacement.

5.4.5 Usability and Adoption Factors

Participants also discussed several factors influencing the adoption and usability of the MCP-based agent. One recurring aspect was that **familiarity with existing workflows influences tool preference**, where some participants preferred the standalone web-based workflow because it aligned more naturally with their established habits and debugging routines. Existing familiarity with browser-based tools and established investigative practices shaped participants' perceptions of the usefulness of the MCP integration. One participant explained that the web-based interaction style felt more natural within their existing workflow:

> "Personally, the web interface feels more natural to me; I tend to use it as my primary tool."
>
> **Participant 14**

Another participant indicated a continued preference for conducting investigations through the browser-based workflow:

> "I prefer to carry out the investigation directly in the web interface."
>
> **Participant 3**

Participants also noted that **conversational interfaces can introduce information overload and interaction challenges**. While conversational and iterative debugging were generally appreciated, some participants raised concerns about excessive information, repetitive conversational loops, and friction in interactions. These concerns suggest that conversational integration alone does not necessarily improve troubleshooting effectiveness. One participant explained that conversational interactions could at times become overwhelming:

> "You can be overstimulated by the amount of information that you get in a text-based format."
>
> **Participant 1**

Another participant described situations in which conversational troubleshooting tended to repeat the same suggestions:

> "It can sometimes turn into a kind of repetitive loop where the same suggestions or steps are presented again."
>
> **Participant 7**

One participant also highlighted practical usability friction related to permissions and setup requirements:

> "The need to repeatedly grant various permissions can be inconvenient."
>
> **Participant 9**

Together, these findings suggest that the adoption of MCP-based workflows is shaped by an interplay between alignment with established working habits, perceived interaction efficiency, and the usability of conversational mechanisms.

Rather than being determined solely by technical capability, uptake appears to depend on how seamlessly the tool fits into existing practices and on whether the interaction model supports effective, non-disruptive troubleshooting.

Overall, the findings show that practitioners perceived MCP-based IDE integration as useful for supporting CI troubleshooting workflows by improving workflow continuity, enabling iterative and context-aware debugging, and supporting closer interaction between log analysis and source code modification. At the same time, practitioners continued to value standalone web-based tools for lightweight inspection and monitoring tasks, suggesting that both approaches serve complementary roles within industrial CI troubleshooting workflows.

6

Discussion

This chapter discusses the identification and application of technical and human-centered evaluation metrics, the alignment between automated and human evaluations, and practitioners' perceptions of the influence of both the log analysis tool and an MCP-based agent on CI troubleshooting workflows.

6.1 Evaluation Metrics

This section discusses the identification of technical and human-centered metrics for evaluating the quality of LLM-based log analysis tool outputs.

6.1.1 Technical Evaluation Metrics (RQ1.1)

This study sought to identify which technical criteria and metrics are relevant for evaluating the output of an LLM-based log analysis tool applied to industrial build logs. The literature review and literature synthesis yield the following answer: conventional text-overlap metrics are structurally ill-suited to this task, and LLM-as-a-judge methods represent a more appropriate evaluation paradigm, though not without limitations.

Statistical scorers are excluded because they require a fixed reference text, a canonical ground-truth that does not exist for industrial build log analysis, and the literature consistently documents their low correlation with human judgment [40]. Model-based scorers improve on this by capturing semantic similarity [44], but remain largely reference-dependent and, most critically, semantic similarity does not equate to the correctness or actionability of a technical analysis: a response can score well yet still misidentify the root cause of a build failure. Both categories are therefore insufficient as primary evaluation mechanisms for this setting.

The findings support LLM-as-a-judge approaches as the most appropriate evaluation paradigm for this study. Unlike the preceding categories, these approaches do not require reference texts and assess outputs against explicit, domain-relevant quality criteria: correctness, completeness, clarity, and actionability, which directly reflect what constitutes a useful log analysis in industrial practice.

Among the metrics identified in the literature, this study selects **G-Eval** and

GPTScore as the primary evaluation metrics. Both produce absolute scores rather than relative rankings, and both support reference-free evaluation underpinned by structured prompting and chain-of-thought reasoning, a prerequisite for evaluating outputs whose value lies precisely in the reasoning they present. Pairwise Comparison was excluded because it yields only relative scores, and MT-Bench was excluded for its inconsistent scoring behavior and its focus on chatbot settings rather than technical log analysis.

In answer to RQ1.1, the evaluation of LLM-based log analysis tool output requires metrics that (1) operate without reference outputs, (2) assess reasoning quality rather than surface-level text similarity, and (3) produce absolute scores that can be interpreted against defined quality criteria. Statistical scorers satisfy none of these requirements. Model-based scorers satisfy only the third partially, and only when a reference is available. G-Eval and GPTScore satisfy all three and are therefore the technically appropriate choices for this evaluation context. This finding has broader implications: as LLM-based tools are increasingly deployed in software engineering workflows, evaluation frameworks must shift from legacy NLP metrics toward reasoning-aware, reference-free methods that reflect the nature of the tasks these tools are asked to perform.

6.1.2 Human-Centered Evaluation Metrics (RQ1.2)

The thematic analysis reveals that CI practitioners evaluate LLM-based log analysis outputs through practical debugging needs rather than abstract notions of output correctness. The most fundamental expectation is accurate root cause identification, with a clear distinction between infrastructure and code failures, as presenting all errors with equal weight increases cognitive load and slows diagnosis. Trustworthiness of explanations is closely tied to specificity; generic outputs that do not reference concrete log content reduce confidence, while explanations linked to specific log lines with concrete fixes are considered genuinely useful. Trust itself is not formed at first use but develops through consistent performance over time, and repeated hallucinations or mismatched solutions are sufficient to drive practitioners away from the tool entirely.

Handling ambiguity also received particular attention, as CI failures frequently involve cascading interdependent errors. Practitioners expected prioritization and ranking of potential causes rather than flat summaries. Output design was equally significant, with a preferred structure combining a direct answer, a brief summary, and optional step-by-step reasoning that adapts to the complexity of the failure. The perceived relevance of the tool is highly context-dependent, being most pronounced for large logs, recurring failures, and junior practitioners.

In answer to RQ1.2, practitioners judge tool quality in relation to how effectively it reduces debugging effort and integrates into existing CI workflows, with key criteria being precise root-cause identification, specificity and trustworthiness of explanations, structured handling of ambiguity, adaptive output formatting, consistent reliability over time, and contextual relevance to the practitioner's experience and

log complexity.

6.2 Alignment Between Automated Metrics and Practitioner Judgments (RQ2)

The evaluation results show that automated LLM-as-a-judge methods produce narrow, optimistic score distributions with limited discriminative power, while practitioner judgments are stricter and show considerably higher variance. Multi-step G-Eval yields the most consistent automated scores, and GPTScore with GPT-4o-Mini produces the highest correctness scores with very low variance, contrasting with the broader spread observed in human assessments. Rather than indicating reliability, this low variance suggests that automated methods assign similar scores regardless of response quality, reducing their ability to distinguish between strong and weak outputs. This divergence in score distributions signals a structural difference in how the two approaches assess the quality of the output.

The correlation analysis confirms weak alignment across all criteria, with Spearman and Kendall coefficients ranging from near zero to at most **0.312**. The most notable anomaly is a negative correlation for clarity under single-step G-Eval, suggesting that what automated evaluators reward as clear is not necessarily what practitioners find interpretable and useful. The fundamental source of this misalignment is a difference in evaluation perspective: human practitioners assess whether the primary root cause is correctly identified and whether the output enables them to act, while LLM-based evaluators tend to reward broader coverage, including less relevant log signals. This divergence between precision-oriented human judgment and coverage-oriented automated scoring drives the ranking inconsistencies observed. The lower correlations compared to prior work on structured NLP tasks [40], which reported coherence scores up to **0.582**, further reflect the added complexity of log analysis, which involves noisy, unstructured data and context-dependent reasoning that standard evaluation prompts are not designed to capture. The limited sample size ($n = 29$) also reduces the statistical stability of the estimates and should be considered when interpreting the magnitude of the correlations.

Despite the weak correlations observed, the findings do not suggest that LLM-as-a-judge metrics should be discarded. Rather, they should be used as decision-support tools rather than replacements for practitioner evaluation. While RQ1.1 identified G-Eval and GPTScore as suitable metrics for reasoning-intensive tasks due to their ability to assess outputs without reference answers, RQ2 shows that their judgments do not align strongly with practitioner assessments. Consequently, organizations should adopt a human-in-the-loop approach, using automated metrics for large-scale screening, trend monitoring, and benchmarking, while reserving human review for outputs that influence operational decisions.

A practical benefit of these metrics is their ability to identify potential quality issues and detect performance regressions across model updates. They can also reduce evaluation effort by prioritizing outputs for human review rather than requiring

experts to assess every generated analysis. However, organizations should avoid using automated scores as standalone acceptance criteria. The weak correlations observed (maximum Spearman $\rho = 0.312$ and Kendall $\tau = 0.288$) indicate that high automated scores do not consistently correspond to high practitioner ratings. Therefore, LLM-as-a-judge metrics should be interpreted as indicators of potential quality rather than objective measures of correctness.

In answer to RQ2, the results suggest that LLM-as-a-judge methods are most valuable as complementary evaluation tools. They provide scalable and repeatable assessments, but their current level of agreement with practitioners is insufficient for autonomous evaluation. Companies can therefore benefit from these metrics when they augment, rather than replace, expert judgment.

6.3 Practitioner Perceptions of the Tool Influence on CI Workflows (RQ3)

The thematic analysis confirms that CI log analysis in industrial environments is inherently manual, cognitively demanding, and heavily dependent on practitioner expertise, particularly in large and distributed systems where failures frequently manifest as cascading chains across multiple components. Against this baseline complexity, practitioners perceived the tool as introducing meaningful efficiency gains by accelerating failure identification, narrowing the search space, and enabling a more interactive and iterative debugging process in place of exhaustive log scanning. These benefits are most pronounced for complex, large-scale, or unfamiliar failures, while the tool offers limited additional value for simple or well-understood issues that practitioners can resolve quickly through their own expertise. These findings align with prior research on AI-assisted software engineering workflows, which suggests that intelligent developer-support tools provide the greatest value in tasks characterized by high cognitive load, uncertainty, and large information spaces rather than routine or deterministic activities [46, 47]. In the context of CI troubleshooting, the tool is perceived to function primarily as a cognitive support mechanism that reduces the effort required to navigate noisy and interdependent build logs.

Beyond efficiency, the tool is perceived to reshape how practitioners make decisions during debugging. Tool-guided prioritization leads developers to act on suggested root causes earlier, which speeds up resolution but also narrows the exploration space and introduces a risk of anchoring on the tool's output before considering alternative hypotheses. Integration friction further moderates the perceived usefulness of the tool: seamless embedding within CI pipelines and IDEs amplifies adoption and usability, while manual steps such as copying and uploading logs introduce interruptions that reduce practical effectiveness. The tool also carries risks that must be acknowledged. Over-reliance may gradually erode practitioners' manual debugging skills and critical thinking, a concern raised most strongly for junior engineers whose expertise development could be shaped by early dependence on automated

analysis. This reflects broader concerns in human-AI collaboration research regarding automation bias, where users may become increasingly inclined to trust system recommendations without sufficient verification [46]. Misleading outputs and hallucinations further compound this risk by potentially directing debugging effort toward incorrect causes, particularly in situations where practitioners lack sufficient domain expertise to independently validate the generated explanations.

From an industrial perspective, these findings suggest that LLM-based log analysis tools are most effective when deployed as decision-support systems rather than autonomous debugging agents. Their practical value appears strongest in environments involving large-scale CI infrastructures, distributed systems, and recurring high-volume failures, where reducing investigation effort and narrowing the search space can substantially improve debugging efficiency. At the same time, the findings indicate that maintaining human oversight remains essential to ensure that efficiency gains do not come at the cost of reduced critical evaluation or misplaced trust in generated outputs.

In answer to RQ3, practitioners perceived that the LLM-based log analysis tool primarily influences CI workflows by reducing manual effort, accelerating root cause identification, and shifting debugging from a linear, individual process to an interactive human-AI collaboration. However, its influence is context-dependent and entails trade-offs: perceived efficiency gains are reported but concentrated in complex scenarios; decision-making becomes faster but is more susceptible to tool bias; and long-term adoption raises legitimate concerns about skill degradation and uncritical reliance on generated outputs.

6.4 Practitioner Perceptions of MCP-Based IDE Integration Compared with the Web-Based Tool (RQ4)

The findings from RQ4 show that practitioners perceive that integrating an MCP-based agent into the development environment supports CI troubleshooting by enabling a more continuous, iterative debugging process. Compared with the standalone web-based tool, the MCP integration reduced context switching and enabled practitioners to combine log analysis, conversational interaction, and source code modification in a single IDE. This suggests that the primary benefit of the MCP-based approach lies not only in tool accessibility but in preserving workflow continuity during debugging. By minimizing transitions between separate systems, practitioners can maintain contextual focus more effectively while iteratively moving between failure investigation, hypothesis refinement, and code modification.

Participants particularly valued the ability to ask follow-up questions, refine investigations iteratively, and directly connect build failures to implementation-level changes. The findings further indicate that conversational and context-aware interaction is particularly valuable in debugging scenarios where failures evolve dynam-

ically and require iterative exploration rather than one-time analysis. This aligns with prior research suggesting that interactive AI systems are more effective when they support continuous human-AI collaboration rather than isolated recommendation delivery [46].

From an industrial workflow perspective, the findings suggest that MCP-based integration may be perceived as especially beneficial in environments where developers frequently alternate between CI analysis and implementation tasks. Embedding AI-assisted troubleshooting directly within the IDE enables tighter coupling between diagnosis and remediation, potentially reducing workflow fragmentation and improving debugging efficiency for complex failures. However, the findings also indicate that integration quality and interaction design remain critical factors influencing practical adoption.

However, the findings also indicate that the MCP-based workflow complements rather than replaces the standalone web-based tool. Participants still preferred the web interface for quick inspections, lightweight troubleshooting, and monitoring-oriented tasks, while the MCP integration was considered more useful for complex and code-centric debugging scenarios. Some practitioners additionally reported challenges related to conversational overload, interaction fatigue, and usability friction, suggesting that continuously interactive workflows may also introduce cognitive overhead when conversational exchanges become excessively long or insufficiently focused.

In answer to RQ4, the findings suggest that practitioners perceive MCP-based IDE integration as useful because it improves workflow continuity and supports more iterative, context-aware debugging practices. Rather than replacing standalone web-based analysis tools, practitioners viewed the MCP-based approach as complementary, particularly valuable for complex and code-centric troubleshooting scenarios where continuous interaction between log analysis and source code modification is beneficial.

6.5 Threats to Validity

This study evaluates threats to validity following the framework proposed by Wohlin et al. [48], which categorizes threats into construct, internal, external, and conclusion validity. Each category addresses a distinct aspect of research quality: whether the study measures what it intends to, whether results are free from confounding factors, whether findings generalize beyond the study context, and whether the conclusions are statistically and analytically supported. The following subsections discuss the identified threats and the mitigation strategies applied.

6.5.1 Construct Validity

The technical evaluation criteria used in RQ1.1 (correctness, completeness, clarity, and actionability) were derived from prior literature. While these provide a struc-

tured basis for evaluation, they may not fully capture all aspects of output quality in the context of CI log analysis. Findings from RQ1.2 indicate that practitioners also consider factors such as trust, relevance, and workflow fit, suggesting that the predefined criteria only partially represent real-world evaluation.

Although the Systematic Literature Review followed a structured, documented procedure, complete reproducibility may be limited by several factors. The selection of search terms, inclusion of specific databases, and subjective judgment during screening (e.g., relevance assessment during title/abstract and full-text review) may introduce bias. Additionally, the inclusion of Google Scholar and arXiv may affect consistency due to differences in indexing and ranking algorithms. To mitigate these threats, clearly defined inclusion and exclusion criteria were applied, and the search process was documented to enhance transparency.

For RQ1.2, RQ3, and RQ4, the use of semi-structured interviews introduces the risk that the collected data reflects participants' interpretations rather than objective evaluation criteria. This was mitigated by designing the interview guide around the research question and by providing a consistent introduction to the tool across participants.

For RQ2, the use of LLM-as-a-judge metrics (G-Eval and GPTScore) introduces construct validity concerns, as evaluation outcomes are sensitive to prompt design, model behavior, and potential bias toward fluent LLM-generated text [40]. To mitigate this, structured and fixed prompt templates aligned with the evaluation criteria were consistently used across all evaluations.

6.5.2 Internal Validity

For RQ1.2, RQ3, and RQ4, participant responses may be affected by differences in experience levels, familiarity with CI systems, or prior exposure to similar tools. These factors may influence how participants interpret outputs and express their evaluations. To mitigate this, participants were selected using purposive sampling to ensure relevant experience, and all interviews followed a consistent procedure.

For RQ2, the non-deterministic nature of LLMs introduces variability in both generated outputs and automated evaluation scores. This may affect the consistency of results, particularly when comparing human and LLM-based evaluations. While this behavior cannot be fully controlled, the study used the same configurations and evaluation setup across all samples to maintain consistency.

Furthermore, the limited number of evaluated logs may influence the observed relationships between human and automated scores. Although the dataset reflects real industrial logs, its size may affect the stability of statistical results.

6.5.3 External Validity

The study was conducted within a single industrial setting, involving CI practitioners from one organization, Bosch. While participants represent a range of experience levels and roles, the findings may not fully generalize to other organizations or domains.

In addition, the log data used in the study originates from a limited number of projects within the same industrial context. As a result, the observed results may be influenced by project-specific characteristics, such as logging practices, system architecture, or failure patterns.

The number of participants was relatively small (14 practitioners), which is typical for qualitative studies but may limit the transferability of the findings to a broader population of CI practitioners. Although thematic saturation was achieved and the sample included practitioners with relevant experience, it may not fully represent the diversity of CI practices and organizational contexts, where additional perspectives could emerge.

These limitations were mitigated by using real-world industrial data and involving practitioners with direct experience in CI workflows. However, further studies across multiple organizations, projects, and larger participant groups would be required to validate the findings more broadly.

6.5.4 Conclusion Validity

For RQ2, statistical analysis was used to examine the relationship between human and LLM-based evaluation scores. The relatively small sample size (e.g., number of log outputs and participants) may affect the stability and statistical power of the results. As a result, the observed correlations should be interpreted with caution, as they may not fully represent broader patterns.

In addition, non-parametric correlation measures (Spearman's rank and Kendall's tau) are appropriate for ordinal data; however, they capture only monotonic relationships and may not reflect more complex dependencies between human and automated evaluations.

Another factor is the variability introduced by LLM-based evaluators. Since these models are non-deterministic, repeated evaluations may produce slightly different scores, which can influence the strength and consistency of the observed relationships.

To mitigate these threats, consistent evaluation procedures and configurations were used across all samples. Furthermore, the results were interpreted in conjunction with the qualitative findings from RQ1.2, providing a more grounded understanding rather than relying solely on statistical measures.

7

Conclusion

This thesis explored the evaluation and practical use of an LLM-based log analysis tool within industrial CI environments through an exploratory case study conducted in collaboration with Bosch. The study investigated which technical and human-centered evaluation criteria are relevant for assessing LLM-based log analysis outputs, how automated evaluation metrics align with practitioner judgments, and practitioners' perceptions of both the tool's influence on CI workflows and how integrating an MCP-based agent into the IDE affects CI workflows compared to a standalone web-based tool.

The findings from RQ1 demonstrate that conventional text-overlap metrics such as BLEU and ROUGE are not suitable for evaluating industrial log analysis outputs due to the absence of reference answers and the reasoning-intensive nature of log analysis. Instead, LLM-as-a-judge approaches, particularly G-Eval and GPTScore, were identified as more appropriate evaluation methods because they support reference-free and reasoning-aware assessment. However, the comparison between automated evaluation and practitioner judgments from RQ2 revealed only weak alignment, indicating that automated metrics cannot reliably replace human evaluation in industrial CI contexts. The thematic analysis further showed that practitioners evaluate outputs based not only on correctness, but also on factors such as precise root-cause identification, trustworthiness, actionability, handling of ambiguity, workflow relevance, and long-term reliability. These findings highlight that evaluating LLM-based log analysis systems is fundamentally socio-technical and closely tied to practical debugging workflows.

The RQ3 findings show that practitioners perceive the LLM-based log analysis tool as useful for reducing manual effort, accelerating failure investigation, and supporting more interactive human-AI collaboration. The tool was perceived as particularly valuable for large, complex, and unfamiliar failures where manual debugging becomes cognitively demanding. At the same time, participants raised concerns regarding hallucinations, over-reliance, reduced critical thinking, and potential degradation of manual debugging skills over time. The findings, therefore, suggest that such systems are most effective as collaborative decision-support tools that augment practitioner expertise rather than replace it.

The findings from RQ4 indicate that practitioners perceive MCP-based IDE integra-

tion as improving workflow continuity by reducing context switching and enabling iterative, conversational, and code-centric troubleshooting directly within the development environment. Participants valued the ability to combine log analysis, source code inspection, and debugging within a single workflow. However, the findings also showed that the MCP-based approach complements rather than replaces standalone web-based tools, which practitioners still preferred for lightweight inspection and monitoring tasks.

Overall, this thesis makes four main contributions. First, it proposes a technical and human-centered evaluation framework for LLM-based log analysis tools. Second, it develops a practical evaluation framework for comparing automated and practitioner-based assessments. Third, it provides empirical insights into how practitioners perceive the influence of LLM-based log analysis tools on industrial CI workflows. Fourth, it presents an exploratory investigation of how practitioners perceive MCP-based IDE integration for AI-assisted troubleshooting. The study also highlights the importance of explainability, trust, workflow integration, and human oversight when deploying LLM-based support systems in industrial software engineering environments.

This study has several limitations. The research was conducted as a single-company industrial case study with a limited participant pool and a relatively small number of evaluated logs. In addition, the MCP-based integration was evaluated as an exploratory prototype using primarily qualitative methods. Despite these limitations, the study provides practical, empirical insights into evaluating and integrating LLM-based troubleshooting systems in industrial CI environments.

Future work could extend this research through larger-scale and longitudinal industrial studies across multiple organizations, CI pipelines, and software domains to evaluate the generalizability of the findings. Further research could also establish standardized industrial benchmarks for LLM-based log analysis and improve alignment between automated evaluation metrics and practitioner judgments. Additionally, as this study focused on single-turn LLM-based log analysis without retrieval-augmented or agentic capabilities, future work could investigate RAG-oriented metrics, such as faithfulness and contextual relevance, alongside agentic evaluation metrics, including task completion, tool correctness, and reasoning consistency, and repository-aware debugging performance, as log analysis systems evolve toward more autonomous and retrieval-enhanced architectures.

In conclusion, the findings demonstrate that LLM-based log analysis systems have strong potential to support industrial CI troubleshooting when designed around real developer workflows and supported by appropriate evaluation approaches. However, their effectiveness depends not only on model capability, but also on trustworthiness, workflow integration, explainability, and the continued involvement of human expertise in the debugging process.

Bibliography

[1] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, 2016.

[2] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1285–1298. Association for Computing Machinery, 2017.

[3] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pages 4739–4745. International Joint Conferences on Artificial Intelligence Organization, 2019.

[4] Yongjun Xu, Fei Wang, and Tangtang Zhang. Artificial intelligence is restructuring a new world. *The Innovation*, 5(6), 2024.

[5] Jiajia Yuan, Peng Bao, Zifan Chen, Mingze Yuan, Jie Zhao, Jiahua Pan, Yi Xie, Yanshuo Cao, Yakun Wang, Zhenghang Wang, Zhihao Lu, Xiaotian Zhang, Jian Li, Lei Ma, Yang Chen, Li Zhang, Lin Shen, and Bin Dong. Advanced prompting as a catalyst: Empowering large language models in the management of gastrointestinal cancers. *The Innovation Medicine*, 1(2):100019, 2023.

[6] Jiarui Ji, Runlin Lei, Xuchen Pan, Zhewei Wei, Hao Sun, Yankai Lin, Xu Chen, Yongzheng Yang, Yaliang Li, Bolin Ding, and Ji-Rong Wen. Leveraging llm-based agents for social science research: Insights from citation network simulations. *Humanities and Social Sciences Communications*, 13(1):127, 2026.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code, 2021.

[8] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software*

Engineering and Methodology, 33(8), 2024.

[9] Siraaj Akhtar, Saad Khan, and Simon Parkinson. Llm-based event log analysis techniques: A survey, 2025.

[10] Peng Cai, Reza Ryan, and Nickson M. Karie. Llmloganalyzer: A clustering-based log analysis chatbot using large language models, 2025.

[11] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12), 2023.

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[13] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81. Association for Computational Linguistics, 2004.

[14] Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[15] Junjie Huang, Zhihan Jiang, Zhuangbin Chen, and Michael Lyu. No more labelled examples? an unsupervised log parser with llms. *Proceedings of the ACM on Software Engineering*, 2(FSE), 2025.

[16] Zhuoyi Yang and Ian G. Harris. Logllama: Transformer-based log anomaly detection with llama, 2025.

[17] Wei Guan, Jian Cao, Shiyou Qian, Jianqi Gao, and Chun Ouyang. Logllm: Log-based anomaly detection using large language models, 2025.

[18] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pages 273–280, 2023.

[19] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yuhang Chen, Yanqing Zhao, Hao Yang, and Yanfei Jiang. Interpretable online log analysis using large language models with prompt strategies. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, pages 35–46, 2024.

[20] Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang. When large language models meet vector databases: A survey, 2025.

[21] Justin K. Miller and Wenjia Tang. Evaluating llm metrics through real-world capabilities, 2025.

[22] Chris Egersdoerfer, Di Zhang, and Dong Dai. Early exploration of using chatgpt for log-based anomaly detection on parallel file systems logs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 315–316. Association for Computing Machinery, 2023.

[23] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1482–1494, 2023.

[24] Sungmin Kang, Gabin An, and Shin Hye Yoo. A quantitative and qualitative evaluation of llm-based explainable fault localization. *Proceedings of the ACM on Software Engineering*, 1(FSE), 2024.

[25] Md Nakhla Rafi, Dong Jae Kim, Tse-Hsun Chen, and Shaowei Wang. Enhancing fault localization through ordered code analysis with LLM agents and self-reflection, 2024.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[27] Google. Gemini 2.5 flash, 2026. Accessed: 2026-03-12.

[28] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NeurIPS '22. Curran Associates Inc., 2022.

[29] Fatemeh Lashkari, Ebrahim Bagheri, and Ali A. Ghorbani. Neural embedding-based indices for semantic search. *Information Processing & Management*, 56(3):733–755, 2019.

[30] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on LLM-as-a-judge, 2024.

[31] Miguel De la Cruz Cabello, Tiago Prince Sales, and Marcos R. Machado. AIOps

for log anomaly detection in the era of LLMs: A systematic literature review. *Intelligent Systems with Applications*, 28:200608, 2025.

[32] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009.

[33] Jane Webster and Richard T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii, 2002.

[34] Hannah Snyder. Literature review as a research methodology: An overview and guidelines. *Journal of Business Research*, 104:333–339, 2019.

[35] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, 2006.

[36] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72. Association for Computational Linguistics, 2005.

[37] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. BERTscore: Evaluating text generation with BERT. In *Proceedings of the 8th International Conference on Learning Representations*, ICLR '20, 2020.

[38] Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer, and Steffen Eger. MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, EMNLP-IJCNLP '19, pages 563–578. Association for Computational Linguistics, 2019.

[39] Thibault Sellam, Dipanjan Das, and Ankur Parikh. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL '20, pages 7881–7892. Association for Computational Linguistics, 2020.

[40] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522. Association for Computational Linguistics, 2023.

[41] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. GPTScore: Evaluate as you desire. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, NAACL '24, pages 6556–6576. Associ-

ation for Computational Linguistics, 2024.

[42] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23. Curran Associates Inc., 2023.

[43] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios N. Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML '24. JMLR.org, 2024.

[44] Marvin Kaster, Wei Zhao, and Steffen Eger. Global explainability of BERT-based evaluation metrics by disentangling along linguistic factors. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, pages 8912–8925. Association for Computational Linguistics, 2021.

[45] James D. Evans. *Straightforward Statistics for the Behavioral Sciences*. Brooks/Cole Publishing, Pacific Grove, CA, 1996.

[46] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz. Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13. Association for Computing Machinery, 2019.

[47] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267:1–38, 2019.

[48] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, 2012.

A

Interview Instrument

Participants are asked two demographic questions prior to the main interview questions in both rounds of the semi-structured interviews. In the first round, participants are asked questions to identify which human evaluation metrics they use. In the second round, participants are asked to assess how the use of the tool affects their workflow.

Consent: This interview will last approximately 45 - 60 minutes, will be conducted and recorded via Microsoft Teams, and will be transcribed. The original record will remain in the Bosch device and network only; however, portions of the interview may be quoted verbatim in the thesis report. All data will be anonymized, and no personally identifiable information (PII) will be included. Participation is voluntary, and you may withdraw from the interview at any time without consequence.

A.1 Demographic Questions

1. How long have you been working as a CI engineer (total experience, including before Bosch)?

2. For how long have you used LLM-based tools (ex, Knut GPT, ChatGPT, GitHub Copilot) for log analysis?

A.2 Questions Related to Human Evaluation Metrics (RQ1.2)

Goal: This interview aims to systematically identify the human-centered evaluation criteria used by CI practitioners when assessing LLM-generated log analysis outputs. We investigate how practitioners judge output summaries given by an LLM. By uncovering implicit and explicit human oracles, the study seeks to inform the design of more usable, trustworthy, and effective AI-assisted log analysis tools in CI environments.

Scenario: *You are working on a CI pipeline. A build fails, and an LLM-powered tool analyzes the logs and provides an explanation along with error names, suggested*

solutions, and root cause.

1. The tool provides a summary of the failure and possible causes.
   (a) How would you decide whether this explanation is good enough for you?
   (b) What specific elements must be present (or absent) for you to consider it useful?
2. The explanation sounds reasonable, but you have not verified it yet.
   (a) What would make you trust (or distrust) this explanation?
   (b) What signals, evidence, or structure would increase your confidence in it?
3. Assume that the explanation appears clear and reliable.
   (a) How do you decide whether you have enough information to take action?
   (b) What would you expect in terms of suggested fixes, validation steps, or level of certainty before acting?
4. Now imagine that the failure could have multiple potential root causes.
   (a) How should the tool communicate ambiguity?
   (b) Do you expect ranked causes, probabilities, explicit uncertainty signals, or something else?
5. You encounter similar failures across different builds, and the tool generates explanations each time.
   (a) What format do you prefer (summary, step-by-step reasoning, direct answer, structured sections, etc.)?
   (b) How important is consistency in wording or structure in similar issues?
6. Imagine the tool works most of the time, but occasionally: It misses important context; Overstates confidence; Provides inconsistent explanations; Or gives unnecessary information when builds succeed.
   (a) What kinds of mistakes or patterns would make you stop relying on the tool?
   (b) Also, would you want insights even when the build succeeds?

A.3 Questions Related to Practitioner Perceptions of the Tool Influence on CI Workflows (RQ3)

Goal: This interview explores how practitioners perceive that LLM-based log analysis tools affect CI workflows when diagnosing build failures in industrial environments. It specifically examines how the LLM-powered tool's existing web-based interface influences these workflows.

1. Could you briefly describe how you usually investigate a CI failure?
2. What are the most time-consuming or difficult parts of the current log analysis process?
3. How do you decide where to start when a CI job fails?
   Follow-up questions:
   (a) What signals do you rely on?
   (b) How long does it usually take?
4. Compared to your usual approach, does the tool reduce the time needed to understand the failure?
   (a) Can you estimate how?
5. Would the tool change the way you normally investigate a failure? If so, how would your troubleshooting steps change?
   Follow-up questions:
   (a) Do you think you would skip steps?
   (b) Do you explore fewer or more possible causes?
   (c) Does it reduce or increase manual log inspection?
6. Does the tool influence your confidence in identifying the root cause?
   (a) Would you change your initial decision based on the tool's output?
   (b) Does it affect which solution you try first?
7. Does the tool help you resolve issues faster, or mainly improve your understanding of the failure?
   (a) Does it help you identify solutions you might not have considered?
8. Does the tool integrate smoothly into your existing workflow, or does it intro-

duce friction?

9. Would you use this tool as a primary aid, a secondary aid, or only for certain types of failures?

10. In what situations do you find the tool useful, and when is it less useful or not useful at all?

11. What risks or concerns do you see in using this tool, and what would need to improve for it to better fit into your workflow?

12. Do you think using this tool could change how developers learn or understand CI systems over time?

13. Could it reduce the need to deeply understand logs? Is that a concern?

A.4 Questions Related to Practitioner Perceptions of MCP-Based IDE Integration on CI Workflows (RQ4)

Goal: This interview examines how practitioners perceive the MCP-based form of an LLM-powered log analysis tool affects CI workflows when diagnosing build failures in industrial settings, compared to its web-based counterpart. It specifically investigates the influence of integrating an MCP-based agent into Visual Studio Code via GitHub Copilot on log analysis workflows. We now focus on how integrating the tool into Visual Studio Code via GitHub Copilot influences workflows compared to the web-based tool.

1. After using both approaches, how does the MCP-based tool fit into your workflow compared to the web-based tool?
   Follow-up questions:

   (a) Where would you use it (early vs later)?

   (b) Does it change how you start investigating?

2. How does working inside the IDE with MCP compare to switching to a web tool in terms of maintaining context during troubleshooting?
   Follow-up questions:

   (a) Do you lose or retain focus?

   (b) Does one feel more continuous?

3. How does interacting through chat in GitHub Copilot compare to using the web interface?
   Follow-up questions:

   (a) Iteration vs one-shot

   (b) Flexibility in asking questions

4. Did using MCP change the steps you take when diagnosing a failure compared to the web tool? If yes, how?
   Follow-up questions:

   (a) Steps removed?

   (b) Steps reordered?

   (c) New steps introduced?

5. In which situations would you prefer the MCP-based approach, and in which would you prefer the web-based tool?

6. If both tools were available, how would you integrate the MCP-based approach into your daily workflow?
   Follow-up questions:

   (a) Replace or complement?

   (b) Frequency of use?

B

CI Practitioners Tool Output Evaluation

LLM-based log analysis tool summary practitioner evaluation scoring

Hello CI Practitioners, I hope you are doing well. As part of our ongoing work on a **log-based analysis tool for Continuous Integration (CI) systems**, we are conducting a small evaluation study and would greatly appreciate your expertise. We are sharing **four log files** per individual with you. These files are **output summaries generated by an LLM-based log analysis tool**, which analyzes CI logs and highlights important information such as:

- **Error lines**
- **Warnings and hints**
- **Potential root causes**
- **Suggested solutions**

The summaries consolidate key information from the logs, including **highlighted error messages, identified root causes, and suggested fixes**, to help developers quickly understand and diagnose CI failures.

For this evaluation, you do not need to analyze the full raw logs. Instead, please review the provided summaries with highlighted log lines and provide the file name, your reasoning (justification for each metric score), and a score for each metric. based on the following criteria.

1. **Correctness (1-5)**
   - The analysis must correctly identify the root cause and log events.
   - Do not reward fluent but incorrect reasoning.

2. **Completeness (1-5)**
   - The analysis should cover all important log signals.
   - Missing key events should reduce the score.

3. **Clarity (1-5)**
   - The explanation should be easy to understand.
   - Logical structure and readability matter.

4. **Actionability (1-5)**
   - The analysis should provide useful recommendations or insights.

Your evaluations will help us assess how effective such summaries are for supporting CI debugging and will also help improve our **LLM-based log analysis tool**.

Note: Kindly ensure that the file names remain unchanged and are used as provided in the email attached files during the form submission.

We greatly appreciate your time and feedback. Please feel free to reach out if you have any questions. Thank you for your valuable contribution.

Evaluation / Scoring for Log file #K (K = 1 - 4)
(Please use the email attached file name as it is)

1. **File name:** _____

2. **Reasoning** *(Your motivation behind the scoring of each metric):*

   _____

   _____

   _____

3. **Score Metrics**

Correctness	1	☐	2	☐	3	☐	4	☐	5	☐
Completeness	1	☐	2	☐	3	☐	4	☐	5	☐
Clarity	1	☐	2	☐	3	☐	4	☐	5	☐
Actionability	1	☐	2	☐	3	☐	4	☐	5	☐

C

Technical Metrics Evaluation Prompt Designs

1. Multi-step G-Eval Evaluation Prompt

Multi-step G-Eval Evaluation Prompt
(Design of the Multi-step G-Eval Evaluation Prompt for Reasoning and Scoring)

_GEVAL_SYSTEM_INSTRUCTION = """ You are an expert evaluator of log analysis systems. Please make sure you read and understand these instructions carefully. Please keep this document open while reviewing, and refer to it as needed. """

_GEVAL_EVALUATION_CRITERIA = """
Evaluation Criteria:
1. Correctness (1-5): The analysis must correctly identify the root cause and log events. Do not reward fluent but incorrect reasoning.
2. Completeness (1-5): The analysis should cover all important log signals. Missing key events should reduce the score.
3. Clarity (1-5): The explanation should be easy to understand. Logical structure and readability matter.
4. Actionability (1-5): The analysis should provide useful recommendations or insights.
"""
GEVAL_STEP1_EXTRACT_LOG_SIGNALS = f"""
{_GEVAL_SYSTEM_INSTRUCTION}
You are tasked with the first step of evaluating a log analysis system. Your goal is to identify key events, anomalies, warnings, success or error signals from the provided log input. Extract and list these signals clearly and concisely.

Log Input: {log_text} """

GEVAL_STEP2_COMPARE_MODEL_OUTPUT = f"""
{_GEVAL_SYSTEM_INSTRUCTION} You are tasked with the second step of evaluating a log analysis system. You have identified key signals from the log, and now you need to compare these signals against the model's generated summary of the log analysis.
Perform the comparison by checking for: Correctness of root cause reasoning identified by the model. Any important signals that are missing from the model's output. Any facts or information that the model hallucinated (made up). Provide a detailed comparison reasoning, highlighting strengths and weaknesses.
Key Log Signals (extracted from original log): {log_signals}
Model's Log Analysis Summary: {model_summary} """

GEVAL_STEP3_EVALUATE_EXPLANATION_QUALITY = f"""
{_GEVAL_SYSTEM_INSTRUCTION} You are tasked with the third step of evaluating a log analysis system. Based on the comparison reasoning from the previous step, assess the overall quality of the model's explanation. Consider the following aspects:
- Logical flow: Is the explanation structured well and easy to follow?
- Technical accuracy: Are the technical details described correctly?
- Practical usefulness: Does the explanation provide insights that are helpful for troubleshooting or understanding?

Provide a concise assessment of the explanation quality.
Comparison Reasoning (from previous step): comparison_reasoning """

GEVAL_STEP4_GENERATE_SCORES_AND_REASONING_JSON = f"""
{_GEVAL_SYSTEM_INSTRUCTION}
{_GEVAL_EVALUATION_CRITERIA}
You are tasked with the final step of evaluating a log analysis system. Based on all the information gathered and assessed so far, you need to assign a score for each of the evaluation criteria (Correctness, Completeness, Clarity, Actionability) on a scale of 1 to 5, where 1 is the lowest and 5 is the highest. You must also provide an overall detailed reasoning that summarizes all your findings from the previous steps, justifying the scores you assign.

Evaluation Summary: {evaluation_summary}

```
Return ONLY JSON in this format. You MUST replace the '0' place
holders with the actual scores (1-5) and provide a detailed rea
soning.

{
"reasoning": "your detailed overall evaluation reasoning, incorpor
```

```
ating all previous steps and justifying the scores you assign for
correctness, completeness, clarity, and actionability",
"scores": {
    "correctness": <score_1_to_5>,
    "completeness": <score_1_to_5>,
    "clarity": <score_1_to_5>,
    "actionability": <score_1_to_5>,
}
}
"""
```

2. Single-Shot G-Eval Evaluation Prompt

Single-Shot G-Eval Evaluation Prompt
(Design of the Single-Shot G-Eval Evaluation Prompt for Reasoning and Scoring)

SINGLE_STEP_GEVAL_EVALUATION_PROMPT = f"""
You are an expert evaluator of log analysis systems. You will be given one summary of the log analysis output and the log input file.
Your task is to evaluate the quality of the model-generated log analysis. Please make sure you read and understand these instructions carefully. Please keep this document open while reviewing, and refer to it as needed.

—
Evaluation Criteria:

1. Correctness (1-5): The analysis must correctly identify the root cause and log events. Do not reward fluent but incorrect reasoning.
2. Completeness (1-5): The analysis should cover all important log signals. Missing key events should reduce the score.
3. Clarity (1-5): The explanation should be easy to understand. Logical structure and readability matter.
4. Actionability (1-5): The analysis should provide useful recommendations or insights.

Evaluation Steps:
Step 1: Read the log input carefully and identify key events, anomalies, warnings, successes, or error signals.
Step 2: Compare the model output with the log signals. Check for Correct root cause reasoning, missing information, and hallucinated facts.
Step 3: Evaluate explanation quality, Logical flow, Technical accuracy, and Practical usefulness.
Step 4: Assign a score for each evaluation criterion on a scale of 1 to 5, where 1 is the lowest and 5 is the highest based on the Evaluation Criteria.

Log Input: log_text
Model Output: model_summary

```
Return ONLY JSON in this format:
{
    "reasoning": "your detailed evaluation reasoning",
    "scores": {
        "correctness": <score_1_to_5>,
        "completeness": <score_1_to_5>,
        "clarity": <score_1_to_5>,
        "actionability": <score_1_to_5>,
    }
}
"""
```

3. GPT-Score Evaluation Prompt

GPTScore Evaluation Prompt
(Design of the Evaluation Prompt for GPTScore Based Root Cause Accuracy Assessment)

GPT_SCORE_EVALUATION_PROMPT = f""" You are an expert evaluator of DevOps log analysis systems. Your task is to evaluate the model-generated analysis based on Root Cause Accuracy.

Evaluation Metric: Root Cause Accuracy (1-5)
5 - The root cause is completely correct and fully aligned with the logs.
4 - The root cause is mostly correct with minor inaccuracies.
3 - The root cause is partially correct but misses important details.
2 - The root cause is mostly incorrect or misleading.
1 - The root cause is completely incorrect or unrelated to the logs.

Evaluation Instructions:
1. Ensure the predicted root cause is supported by log evidence.
2. Penalize hallucinated or fabricated information. Focus only on root cause accuracy.
3. Do NOT consider writing style or explanation quality.

Logs: log_text
Model Output: model_summary
Return ONLY one number (1, 2, 3, 4, or 5).
Score:
"""

D

Sub-Themes from Thematic Analysis

D.1 Explanation for Sub-Themes of Human Evaluation Metrics

Theme 1: Spotting Exact Error and Location

Differentiate Infrastructure and Development Error: Ability to distinguish whether the error originates from infrastructure issues or development-related problems.

Exact Error Identification and Relevance: Ability to accurately identify the root error within noisy logs and pinpoint its exact relevant location.

Reduction of Overall Diagnosis and Debugging Time: Helps reduce the total time required for diagnosing and debugging issues in log files.

Theme 2: Trustworthiness of Explanations

Specific and Meaningful Explanations: Provides clear, specific, and meaningful explanations that directly relate to the identified issue.

Actionable Suggested Solutions: Offers practical and actionable solutions that users can apply to fix the identified problem.

Trust & Confidence Building: Builds user trust and confidence through consistent, accurate, and reliable explanations.

Theme 3: Handling Complexity and Ambiguity

Handling Multiple Issues and Ambiguity: Ability to manage multiple errors, complex logs, and ambiguous situations effectively.

Theme 4: Output Design and User Interaction

Response Structuring and Formatting: Presents responses in a well-structured and clearly formatted manner for easy understanding.

Ease of Tool Usability: Ensures the tool is easy to use, interact with, and integrate into the user workflow.

Theme 5: Reliability and Long-Term Trust

Output Hallucinations and Misleading Information: Evaluates whether the tool avoids hallucinated or misleading information in its output.

Theme 6: Relevance of the Tool

Relevance of Tool Based on User Experience: Measures how relevant and useful the tool is based on user experience and expectations.

Relevance of Tool Based on Log File Size: Measures how the tool's usefulness changes as the log file size increases.

D.2 Explanation for Sub-Themes of Practitioners' Perspectives of LLM-Based Log Analysis Tool Influence on CI Workflows

Theme 1: Efficiency Gains through Tool-Assisted Analysis

Acceleration of Failure Identification: The tool speeds up failure investigation by highlighting relevant errors and narrowing the search space.

Reduction in Manual Effort: Automated summaries and root-cause suggestions reduce repetitive manual log inspection activities.

Context-Dependent Effectiveness: The usefulness of the tool varies depending on issue complexity, familiarity, and practitioner expertise.

Theme 2: Transformation of Debugging Workflows

Reduction & Skipping of Traditional Steps: The tool minimizes traditional debugging activities such as exhaustive log scanning and manual filtering.

Iterative Human-AI Collaboration Loop: Debugging becomes a collaborative process where developers iteratively validate and refine tool-generated suggestions.

Theme 3: Influence on Decision-Making and Error Exploration

Tool-Guided Prioritisation of Root Causes: Developers tend to prioritize investigation paths and probable causes suggested by the tool.

Narrowing vs Expanding Exploration Space: The tool can either narrow the investigation focus or broaden the exploration by introducing additional debugging perspectives.

Faster Decisions and Influence on Solution Attempt: Tool-generated suggestions accelerate troubleshooting decisions and influence which fixes are attempted first.

Theme 4: Integration & Workflow Fit

Seamless Integration as a Key Enabler: Smooth integration within IDEs and CI environments improves usability, workflow continuity, and adoption.

Friction from Manual Interaction: Manual steps such as copying, pasting, and uploading logs introduce workflow interruptions and usability friction.

Preference for IDE / CI Plugin Embedding: Participants preferred embedded integrations that support direct interaction within existing development workflows.

Theme 5: Contextual Utility and Usage Patterns

Effective for Complex and Unfamiliar Failures: The tool is particularly useful for large-scale, complex, or previously unseen CI failures.

Limited Value for Simple or Known Issues: The tool provides less additional benefit for repetitive, familiar, or straightforward debugging scenarios.

Use as Primary vs Secondary Aid: Practitioners use the tool either as a primary debugging assistant or as conditional support depending on context.

Theme 6: Risks and Unintended Consequences

Over-Reliance & Reduced Critical Thinking: Heavy dependence on the tool may reduce developers' manual reasoning and critical debugging practices.

Skill Degradation in Log Analysis: Long-term reliance on automated analysis may weaken traditional log investigation and troubleshooting skills.

Misleading Outputs and Time Loss: Incorrect or hallucinated suggestions can misdirect debugging efforts and increase troubleshooting time.

Theme 7: Influence on Learning & Expertise Development

Reduced Deep Understanding of CI Systems: Extensive reliance on automated analysis may reduce developers' understanding of CI internals and system behavior.

Shift: Analysis to Orchestration Skills: Expertise gradually shifts from low-level debugging toward orchestration, coordination, and problem decomposition skills.

Faster Learning vs Superficial Understanding: The tool can accelerate learning while also creating risks of shallow or surface-level understanding.

D.3 Explanation for Sub-Themes of Practitioner Perceptions of MCP-Based IDE Integration on CI Workflows

Theme 1: Continuous & Contextual Troubleshooting

Reduced context switching improves workflow continuity: The ability of the MCP agent to reduce interruptions by minimizing the switching between development, debugging, and analysis tools.

Access to source code context strengthens troubleshooting: Direct access to source code improves debugging precision and contextual understanding of failures.

Theme 2: Iterative & Conversational Debugging

Conversational interaction supports progressive investigation: Supports step-by-step investigation through conversational and follow-up interactions.

Iteration creates faster experimentation and local feedback loops: Enables rapid experimentation, testing, and refinement during troubleshooting activities.

Theme 3: Different Roles of MCP & Web-Based Tools

Web-based tools are preferred for quick overviews: Web-based tools support fast monitoring, lightweight inspection, and high-level understanding of CI failures.

MCP is preferred for complex and code-centric troubleshooting: MCP integration supports deeper investigation, iterative debugging, and source code-oriented troubleshooting.

Theme 4: Workflow Changes & Complementary Usage

MCP reduces manual workflow friction: Integration reduces repetitive manual activities and streamlines troubleshooting workflows.

Existing practices are complemented rather than replaced: Integration of MCP improves existing troubleshooting practices rather than completely replacing them.

Theme 5: Usability & Adoption Factors

Familiarity with existing workflows influences tool preference: Existing habits and prior workflow experience influence adoption and tool preference.

Conversational interfaces can introduce information overload and interaction challenges: Conversational workflows may increase cognitive load, repetitive interactions, and usability friction.