LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation

Weiyuan Xu[1,2,†], Juntao Luo[2,†,‡], Tao Huang[2], Kaixin Sui[2], Jie Geng[2], Qijun Ma[2], Isami Akasaka[2], Xiaoxue Shi[2], Jing Tang[2], Peng Cai[1,*]
[1]East China Normal University, Shanghai, China
[2]ByteDance, Shanghai, China

***Abstract*—Continuous Integration and Deployment (CI/CD) pipelines are critical to modern software engineering, yet diagnosing and resolving their failures remains complex and labor-intensive. We present LogSage, the first end-to-end LLM-powered framework for root cause analysis (RCA) and automated remediation of CI/CD failures. LogSage employs a token-efficient log preprocessing pipeline to filter noise and extract critical errors, then performs structured diagnostic prompting for accurate RCA. For solution generation, it leverages retrieval-augmented generation (RAG) to reuse historical fixes and invokes automation fixes via LLM tool-calling.**

On a newly curated benchmark of 367 GitHub CI/CD failures, LogSage achieves over 98% precision, near-perfect recall, and an F1 improvement of more than 38% points in the RCA stage, compared with recent LLM-based baselines. In a year-long industrial deployment at ByteDance, it processed over 1.07M executions, with end-to-end precision exceeding 80%. These results demonstrate that LogSage provides a scalable and practical solution for automating CI/CD failure management in real-world DevOps workflows.

***Index Terms*—Continuous Integration, Continuous Deployment, Large Language Models, Log Analysis, Root Cause Diagnosis, Failure Remediation**

I. Introduction

As the backbone of modern software engineering, Continuous Integration and Continuous Deployment (CI/CD) has become critical infrastructure for reliable, rapid software delivery [1]. It enables higher release frequency, faster iteration, and reduced operational risk, and is widely adopted across leading technology companies [2]–[4]. Large-scale empirical studies also confirm its growing adoption in open-source communities and its pivotal role in modern development practices [5].

In parallel, large language models (LLMs) have achieved impressive results across a range of software engineering tasks [6], including requirements engineering [7], code retrieval [8], automated code review [9], and unit test enhancement [10]. Encouraged by these advances, recent work has begun exploring LLM-based approaches for failure detection and remediation—opening up new possibilities for intelligent automation in software delivery.

Yet despite progress in automated software engineering, real-world CI/CD pipelines still suffer frequent failures. Diagnosing and fixing them typically requires on-call engineers to inspect lengthy, semi-structured logs filled with irrelevant or misleading information, a process that is time-consuming and detrimental to both development velocity and product stability. While many failures follow recurring patterns and organizations accumulate substantial resolution knowledge, this knowledge is often stored in unstructured formats that are difficult to retrieve or reuse with traditional non-LLM methods, leading to redundant work and wasted resources.

A major barrier is the limited academic focus on CI/CD log analysis. Although log-based anomaly detection is well studied, most work targets streaming system logs [11] and assumes structured formats or consistent templates [12]–[16], which generalize poorly to noisy, context-rich, file-level CI/CD logs. Even recent CI/CD log analysis efforts [17] rely on template-based deep learning techniques with poor generalizability. These approaches often require large labeled datasets for training, lack interpretability, and cannot produce actionable remediation strategies. While LLMs offer promising reasoning capacity, their application to CI/CD root cause analysis remains underexplored [18], and existing baselines leave considerable room for improvement in diagnostic accuracy, repair guidance, and end-to-end integration.

To address these gaps, we present **LogSage**, the first end-to-end LLM-powered framework for CI/CD failure detection and remediation. LogSage conducts fine-grained root cause analysis (RCA) and automated resolution directly from raw logs, producing human-readable diagnostic reports and applying executable fixes through tool-calling. Its two-stage pipeline combines token-efficient log preprocessing for RCA with multi-route retrieval-augmented generation (RAG) for solution generation, retrieving domain knowledge from diverse internal sources and integrating it with RCA report to synthesize executable remediation strategies. This enables accurate RCA, interpretable reasoning, and automated recovery with minimal

[*] Corresponding author.[†] Equal contribution.[‡] Work conducted during the internship at ByteDance.

developer effort. Our key contributions are as follows:

- **First end-to-end LLM framework:** LogSage is the first LLM-based framework for fully automated CI/CD failure detection and remediation. It integrates a token-efficient log preprocessing strategy that filters noise, expands context, and adheres to token constraints without relying on templates, together with a multi-route RAG-based solution generator that reuses historical fixes and leverages LLM tool-calling for executable remediation.
- **Curated dataset:** We release a dataset of **367** real-world CI/CD failures from GitHub repositories, each annotated with key log evidence, enabling reproducible RCA evaluation and fostering future research (see Appendix A).
- **Extensive validation:** On the curated dataset, LogSage improves RCA F1 score by over **38%** compared with prompt-based LLM baselines, achieving **98% precision** and near-perfect recall. In large-scale industrial deployment at ByteDance, it processed over **1.07M executions** with sustained adoption and **80%+ end-to-end precision**, demonstrating practical usability in production.

The remainder of this paper is structured as follows: Section II surveys related work. Section III introduces the LogSage framework, including both its high-level architecture and implementation details. Section IV presents experiments and evaluates the RCA stage. Section V reports on the real-world deployment of LogSage at ByteDance. Finally, Section VI concludes the paper and outlines directions for future research.

II. Related Work

With the rise of AI techniques in software engineering, CI/CD pipelines have increasingly adopted machine learning (ML), deep learning (DL), and LLMs to improve efficiency, reliability, and fault tolerance [19]. In this section, we review prior work from three perspectives: traditional AI methods in CI/CD pipelines, log anomaly detection, and the emerging application of LLMs for failure detection and remediation.

A. AI in CI/CD Pipelines

Traditional ML approaches in CI/CD focus primarily on predicting test outcomes, build success, or optimizing test execution to reduce cost. These methods typically rely on structured features such as code metrics and commit history, with limited support for runtime failure analysis or recovery. For instance, CNNs have been used to identify false positives in static code analysis [20], while other studies aim to skip unnecessary tests or prioritize test selection to save time and cost [21]–[25]. Additionally, defect prediction has been explored as an indirect way to reduce failures, though these methods do not directly address fault localization or remediation [26], [27].

DL methods enhance predictive accuracy and generalization in failure prediction tasks [28], [29]. However, most models lack semantic understanding of failure causes. Mahindru et al. [30] proposed the LA2R system, which combines log anomaly detection with metadata prediction to retrieve relevant resolutions. While effective, this approach depends heavily on templates and rule-based clustering, limiting its adaptability to evolving log formats and unseen failures.

B. Log Anomaly Detection

Among CI/CD tasks, log anomaly detection is particularly critical and challenging due to the unstructured and context-dependent nature of log data. Early deep learning work by Du et al. [13] used LSTMs to learn normal log sequences and detect anomalies. Meng et al. [14] extended this idea with Template2Vec to capture sequential and quantitative anomalies. Yang et al. [15] employed a GRU-based attention model and semi-supervised learning to reduce labeling cost, while Zhang et al. [16] addressed unstable logs using attention-based Bi-LSTMs. Recent work by Le et al. [11], [12] introduced parser-independent Transformer-based methods that improve generalization by avoiding reliance on log parsing.

Given their strong semantic understanding and reasoning abilities, LLMs are well suited for log anomaly detection. Studies have shown that even prompt-based LLMs can detect anomalies in streaming logs [31]–[34]. Qi et al. [35] proposed a GPT-4-based detection framework, while Shan et al. [36] leveraged configuration knowledge for anomaly detection. Almodovar et al. [37] fine-tuned LLMs for better performance, and Ju et al. [38] adopted RAG to enhance log interpretation. Hybrid methods have also been proposed: Guan et al. [39] combined BERT and LLaMA for semantic vector extraction, and Hadadi et al. [40] used LLMs to compensate for missing context in unstable environments. In-context learning techniques were further used to refine log representation and guide fine-tuning [41], [42].

Despite these advances, most studies focus on streaming logs, with limited attention to file-based CI/CD logs. Moreover, few works provide large-scale industrial validation or full pipeline integration, leaving a gap for further exploration.

C. LLMs for Failure Detection and Remediation

Beyond anomaly detection, LLMs offer unique potential in end-to-end failure diagnosis and automated remediation. A recent survey identified root cause analysis and remediation as key use cases for LLMs in AIOps [43]. For root cause analysis, Roy et al. [44] and Wang et al. [45] proposed tool-augmented LLM agents, while Zhang et al. [46] and Goel et al. [47] used in-context learning and RAG for incident diagnosis with real-world data. On the remediation side, Sarda et al. [48], [49] and Ahmed et al. [50] automated fault recovery in cloud environments. Wang et al. [51] incorporated Stack Overflow into LLM-based solution generation, and Khlaisamniang et al. [52] applied generative AI for self-healing systems.

In the CI/CD context, however, LLM-based solutions remain underexplored. Chaudhary et al. [53] proposed an LLM-based assistant for CI/CD operations, and Sharma et al. [54] outlined key challenges in adopting LLMs for fault handling in build pipelines. While promising, these efforts have yet to address the complexity of real-world CI/CD logs or provide comprehensive remediation workflows.

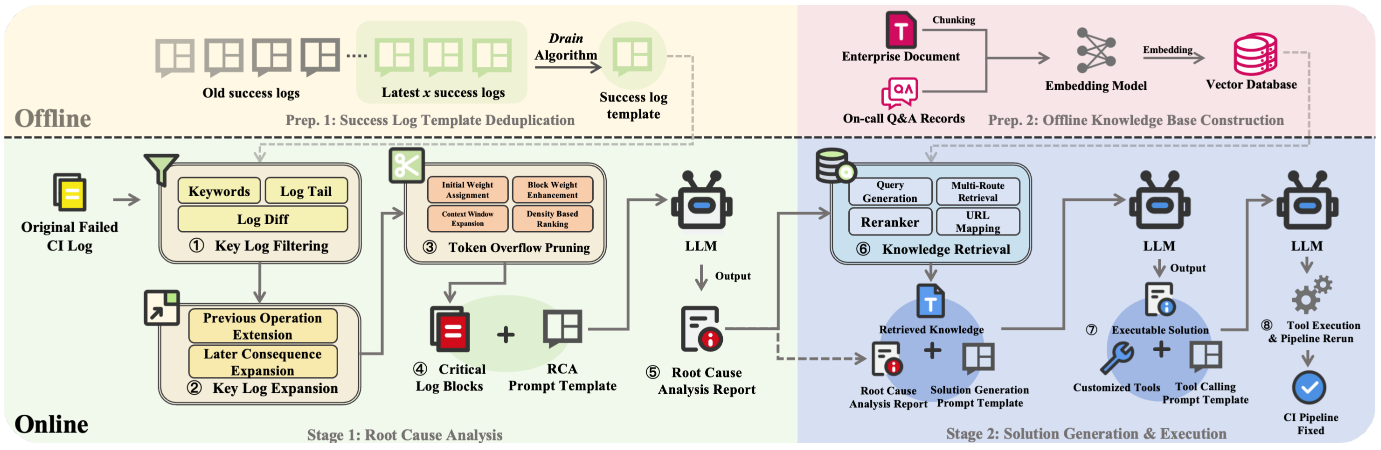

Fig. 1: Overview of the **LogSage framework**, consisting of an offline preparation phase for log template deduplication and knowledge base construction, and an online operational phase for RCA and solution generation with execution.

Overall, while ML and DL methods have laid the foundation for predictive CI/CD analytics, LLMs bring new opportunities for building robust, context-aware, and automated failure detection and remediation systems. However, systematic integration of LLMs into CI/CD pipelines, especially for file-based logs and knowledge-driven automated remediation, remains an open research challenge.

III. Methodology

In this section, we first provide an overview of the LogSage framework architecture in Section III-A, then we elaborate on the detailed workflows and key components of the two stages of LogSage in Section III-B and Section III-C, respectively.

A. Overview of LogSage

The two-stage architecture of LogSage is illustrated in Figure 1. The system consists of both offline and online phases. In the offline phase, success log templates are deduplicated via the Drain algorithm (Prep. 1). Enterprise documents and on-call Q&A records are embedded into vector databases to construct a knowledge base (Prep. 2). In the online phase, Stage 1 processes failed logs through (1) key log filtering, (2) expansion, and (3) token pruning, yielding (4) critical log blocks that form RCA prompt to generate (5) RCA report. Stage 2 leverages (6) knowledge retrieval and LLM-based tool selection to generate (7) executable solutions, followed by (8) automated tool execution and pipeline rerun, ultimately restoring the CI/CD pipeline.

B. Root Cause Analysis Stage

In this section, we present the design and implementation of the RCA stage. The goal of this stage is to extract the most relevant portions of raw CI/CD failure logs and enable accurate reasoning by the LLM, which then returns a structured diagnostic report containing key error lines and an interpretable root cause summary. This stage must address several practical challenges inherent in real-world CI/CD log analysis:

First, **log heterogeneity**: CI/CD pipelines differ widely in their execution steps, commands, output formats, and logging styles, making it infeasible to rely on a unified, structured template to accommodate all scenarios.

Second, **informational noise and misleading lines**: Raw logs often contain numerous irrelevant `WARNING` or `ERROR` messages that do not reflect the actual cause of failure. Naively feeding the full log into the LLM can lead to degraded reasoning quality and even hallucinations. Conversely, filtering logs too aggressively (e.g., by extracting only lines containing keywords such as `ERROR` or `FAILED`) may omit critical contextual information, causing the model to speculate and increasing the risk of misdiagnosis.

Third, **input length constraints**: LLMs cannot accept arbitrarily long inputs. Excessively long log sequences not only increase inference cost and latency but also reduce reasoning accuracy due to context dilution. In industrial CI/CD environments, these limitations become particularly critical, as practical deployments require LLM applications to maintain a stable context length and predictable response latency in order to ensure system robustness and operational usability.

To address these challenges, we design a dedicated log preprocessing pipeline prior to model invocation. This pipeline consists of the following modules:

- **Key Log Filtering**: Extracts candidate log lines from the *failed log* based on keyword matching, log tail prioritization and log diff against recent *success log template*.
- **Key Log Expansion**: Adds contextual lines surrounding the extracted errors to preserve semantic coherence and prevent information loss.
- **Token Overflow Pruning**: Ranks expanded log blocks by weight calculation and prunes low-priority blocks to ensure input remains within a predefined token limit.
- **Dynamic Prompt Assembly**: Constructs RCA prompt by integrating role-playing, chain-of-thought reasoning, few-shot learning, and output-format constraints. Then dynamically combines them with the processed critical