Automated Tool Selection and Execution: Once a solution is generated, LogSage leverages LLM tool-calling guided by the generated solution, the available tool set, and a specially designed prompt template. The LLM then selects and invokes the most appropriate tools from the internal automation toolkit. Currently, these tools include:

- `lint_fix`: automatically resolves lint-related issues without human intervention by leveraging the LLM's contextual understanding and code generation capabilities;
- `image_fix`: addresses container image version mismatches and automatically retrieves the correct images;
- `clean_cache`: clears build or dependency caches;
- `apply_resource`: requests access to internal resources.

After selecting the appropriate tool, the model fills in the required parameters and returns an interactive tool card with an execution button to the user interface. Upon user confirmation, the system invokes the selected tool and automatically re-triggers the CI/CD pipeline. If the remediation succeeds, the pipeline proceeds normally.

Solution Generation Prompt Template

# Role: You are a CI/CD pipeline failures fix assistant. Your task is to generate repair suggestions based on error logs, RCA report, and retrieved troubleshooting documents.

# Skills:

- **Targeted Suggestions:** Use RCA report and relevant documents to propose solutions. Each solution must align with the diagnosed error.
- **Multiple Options:** For each issue, list all applicable solutions. Choose document content most relevant to the CI/CD context.
- **Cite Source:** Every solution must mention the source document and include its official link (if provided). Do not fabricate URLs.
- **Command-level Guidance:** Provide actionable suggestions (e.g., install commands or CI config changes), not vague descriptions.

# Output Format:

- Use section: `### Solutions`.
- For a single root cause:

```
### Solutions
#### Problem: <summary>
##### 1. <Suggestion>
... Refer to [Doc Name](file_0)
```

- For multiple root causes:

```
### Solutions
#### Problem 1: ...
##### 1. ...
##### 2. ...
#### Problem 2: ...
```

- Always include test name, file path, or dependency name when available.

# Notes:

- Be concise and avoid redundant descriptions.
- Do not suggest solutions not backed by retrieved documents.
- Do not add file links; only use document links if provided.

# Constraints:

- DO NOT generate new URLs — only use those in the documents.
- DO NOT omit document attribution in solutions.
- DO NOT provide vague advice like "check the config".

Fig. 3: Prompt template for LogSage's solution generation stage.

IV. Experimental Evaluation for RCA Stage

In our survey of related work, we found that existing research provides suitable baselines for comparison with LogSage's first stage (LLM-based RCA), but no appropriate counterparts exist for the second stage. Due to this limitation, we designed our experiments in two parts: for the first stage, we collected CI/CD task data from public GitHub repositories to compare LogSage with existing LLM-based RCA approaches, highlighting its theoretical performance advantages. The evaluation of the second stage, as well as the end-to-end effectiveness of LogSage, is presented in the following section V using data from ByteDance's real-world industrial deployments.

The research objectives for RCA in this section are:

- **RQ1**: How does LogSage perform in root cause analysis precision compared to existing LLM-based baselines?
- **RQ2**: How does LogSage's cost efficiency in the root cause analysis stage compare to LLM-based baselines?

We conducted the following experiments around the above questions.

A. Experimental Setup

To comprehensively validate the performance of LogSage across various CI/CD scenarios, we built a public dataset for comparative evaluation against other LLM-based root cause analysis baselines. The models selected for this experiment are GPT-4o, Claude-3.7-Sonnet and Deepseek V3, with hyperparameters set to temperature = 0.1, and all other settings using the default values of the respective models.

1) ***Dataset Description****:* To obtain representative and diverse CI/CD failure log cases, referring to [56], we crawled the top 1,000 GitHub repositories by star count and filtered non-engineering-related repositories to ensure the cases have practical software engineering relevance. We then used the GitHub Action public API to retrieve the latest 300 CI/CD run logs for each qualifying repository and identified success and failure run pairs with the same workflow ID. After filtering out cases where no suitable run pairs exist, we obtained 367 cases from 76 repositories, each manually analyzed to identify the key failing log lines and the root causes. The first 117 cases were used to train the *Drain* algorithm and build the few-shot log lines pool, with the remaining 250 cases used as the test set. Data collection was completed on April 25, 2025, and the dataset will be open-sourced on GitHub (see Appendix A).

2) ***Baseline Setup****:* According to our survey in Section II, LogSage is the first method specifically designed for CI/CD root cause analysis, and thus there are no existing baselines that can be directly compared. As the best available alternatives, we adopt LogPrompt [33] and LogGPT [35] as baselines, since they represent recent LLM-based approaches for general log analysis and anomaly detection. We do not compare against non-LLM methods, as such approaches are typically restricted to specific log formats or require training models from scratch, which contrasts with LogSage's training-free, plug-and-play integration into arbitrary CI/CD systems.

- **LogPrompt**: LogPrompt is an interpretable log analysis framework that utilizes prompt engineering techniques to guide LLMs in detecting anomalies without requiring any fine-tuning. It is designed for real-world online log analysis scenarios and emphasizes interpretability by generating natural language explanations alongside detection results. Evaluations across nine industrial log datasets show that LogPrompt outperforms traditional deep learning methods in zero-shot settings.
- **LogGPT**: LogGPT explores ChatGPT for log-based anomaly detection using a structured prompt-response architecture. It integrates chain-of-thought reasoning and domain knowledge injection to improve detection accuracy. The system generates structured diagnostic outputs with explanations and remediation suggestions. Experiments on benchmark datasets such as BGL demonstrate its competitiveness against state-of-the-art deep learning baselines under few-shot and zero-shot conditions.

We evaluate LogPrompt and LogGPT using their original prompt templates, with optimal settings: LogPrompt (few-shot = 20, window = 100) and LogGPT (few-shot = 5, window = 30). As shown in Table I, LogSage is the only method that supports the full pipeline from interpretable root cause analysis to automated solution execution. LogPrompt can produce interpretable diagnostic outputs but lacks the ability to generate solutions. Although LogGPT claims to generate remediation suggestions, it operates without any external knowledge integration and relies solely on the LLM's pre-trained knowledge, often resulting in hallucinated or impractical outputs.

TABLE I: Comparison of LogSage and Baseline Methods

Method	Anomaly Detection	Interpretable Root Cause	Solution Generation	File-level Processing	Automated Execution
LogSage	✓	✓	✓	✓	✓
LogGPT	✓	✓	○	×	×
LogPrompt	✓	✓	×	×	×

*3) **Metrics Description**:* We evaluate RCA task using standard metrics: Precision, Recall and F1-Score, with TP, FP, FN and TN defined specifically for this scenario as follows:

- **TP**: A correct detection. For LogSage, critical error log lines must overlap $\geq 90\%$ with ground truth; for LogPrompt and LogGPT, they must fall within a predefined context window.
- **FP**: Detected lines that fail to meet the above criteria.
- **FN**: LogSage produces no output; LogPrompt/LogGPT fail to detect anomalies despite overlap.
- **TN**: Only applicable to LogPrompt and LogGPT when no detection intersects the context window.

B. ***RQ1**: Root Cause Analysis Precision Comparison*

To compare the RCA performance of LogSage with the two baseline methods and four prompt settings, we conducted experiments on three mainstream LLMs. The results are shown in Table II. We conducted Wilcoxon signed-rank tests comparing LogSage with each baseline variant, and all F1 score improvements are statistically significant ($p < 0.001$).

It is evident that LogSage performs consistently well across all models, with near-perfect scores and minimal fluctuation, indicating high precision and robustness in handling diverse CI/CD log formats, lengths, and error types. In contrast, LogPrompt's performance suffers in precision, while LogGPT shows significant variability, with both methods exhibiting low recall, which suggests that their window-based sampling might miss key contextual information, leading to errors in anomaly detection.

RQ1 Result: LogSage significantly outperforms the baseline methods in root cause analysis, demonstrating high precision, recall, and F1-score across various models, ensuring stability and reliability.

C. ***RQ2**: Root Cause Analysis Cost Comparison*

Considering the cost and time consumption associated with LLMs, we analyzed the token usage and query rounds for each method from the baseline experiments. The distribution of the data is shown in the violin plots Figure 4 and 5.

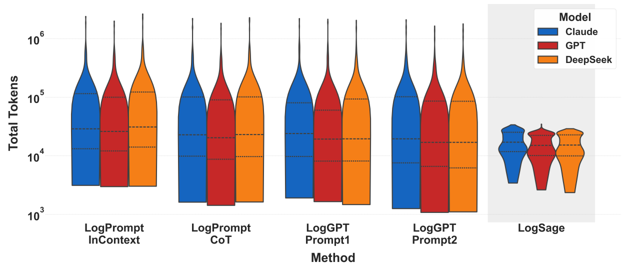

Fig. 4: Token usage for RCA across methods and LLMs.

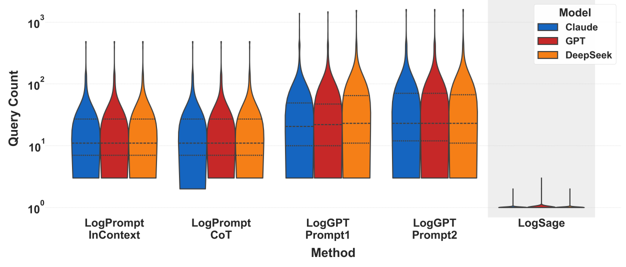

Fig. 5: Query rounds for RCA across methods and LLMs.

As shown in the token graph, the majority of CI/CD logs in the dataset have a length ranging from 10^4 to 10^5 tokens, with a long-tail distribution for some projects with unusually large logs. LogPrompt and LogGPT, which use stream-based processing through a sliding window without token length limits, show a rapid increase in token consumption as log length increases, leading to unnecessary cost in real-world scenarios. In contrast, LogSage not only achieves the best overall performance, but also benefits from a predefined token

TABLE II: Comparison of methods performance (Precision, Recall, F1-score) across different LLMs. [*] indicates F1 scores that are statistically significant at $p < 0.001$ compared with all baselines (Wilcoxon signed-rank test).

Group	Prompt Type	GPT-4o			Claude-3.7-Sonnet			Deepseek V3		
		P	R	F1	P	R	F1	P	R	F1
LogSage	N/A	0.9798	**1.0000**	**0.9898**[*]	0.9837	0.9918	**0.9878**[*]	**0.9838**	0.9959	**0.9898**[*]
LogPrompt	CoT	0.8580	0.4654	0.6035	0.8467	0.3361	0.4812	0.8620	0.4477	0.5893
	InContext	0.6120	0.4530	0.5206	0.6864	0.4046	0.5091	0.6923	0.5018	0.5819
LogGPT	Prompt-1	0.7280	0.3081	0.4330	0.8140	0.3075	0.4464	0.8006	0.3394	0.4767
	Prompt-2	0.7185	0.3484	0.4693	0.7715	0.3073	0.4396	0.7380	0.4438	0.5543

limit, maintaining stable token consumption across different models and ensuring more predictable costs.

As shown in the query round graph, LogPrompt and LogGPT exhibit a linear relationship between query rounds and token length. For most logs, they require nearly 10 query rounds to complete root cause analysis, with some extreme cases needing up to 1,000 rounds. Such a high number of query rounds is not only highly inefficient but also prone to failure in real-world application scenarios. In comparison, LogSage efficiently limits the query rounds to around 1 for most cases, with minimal retries in edge cases, offering significant time advantages.

TABLE III: Efficiency Comparison Across Methods

Method	Avg. Tokens	Avg. Queries	Token Variability
LogSage	**17,853**	**1.0**	**14.46%**
LogPrompt	152,615	31.7	26.30%
LogGPT	122,353	89.4	19.00%

We also analyzed the average token consumption and query rounds across the methods in Table III. The average token consumption of LogPrompt and LogGPT are 152k and 122k tokens respectively, whereas LogSage maintains an average consumption of under 18k tokens—amounting to only 11.84% of LogPrompt's and 14.75% of LogGPT's. Moreover, LogSage demonstrates stable cross-model efficiency, with a normalized token variability of just 14.46%, significantly lower than LogPrompt's 26.30% and LogGPT's 19.00%.

In terms of query rounds, LogSage exhibits strong practical applicability by requiring only 1.0 query round on average to perform accurate and actionable RCA for CI/CD failures across the entire test set. In contrast, LogPrompt, benefiting from a large window size, requires an average of 31.7 queries, while LogGPT, due to its smaller window size, incurs an average of 89.4 queries, rendering it nearly infeasible in real-world scenarios.

RQ2 Result: LogSage efficiently completes root cause analysis in an average of 1 query round with only 11. 84% to 14. 75% token consumption compared to baseline methods, making it a highly cost-effective solution in real-world production environments.

V. Industrial Deployment Validation

We conducted a year-long online deployment and manual evaluation of LogSage to assess its accuracy and effectiveness in real-world CI/CD environments.

*1) **Integration Method**:* LogSage was directly embedded into the company's internal CI/CD platform (Fig. 8). When a CI/CD run failed, users could invoke LogSage from the failure page, review the diagnosed root cause and suggested fix, optionally trigger the repair tool, and provide feedback. Successful fixes automatically retriggered the pipeline, ensuring seamless integration and minimal workflow disruption.

*2) **Deployment Scope**:* Since May 2024, LogSage has processed **1,070,613** CI/CD failures across the company, serving **36,845** developers, and has been made available to all R&D teams using CI/CD services company-wide. Weekly active users exceeded **5,000** and coverage has stayed above **80%** since Oct 2024 (Fig. 6), showing broad and sustained adoption.

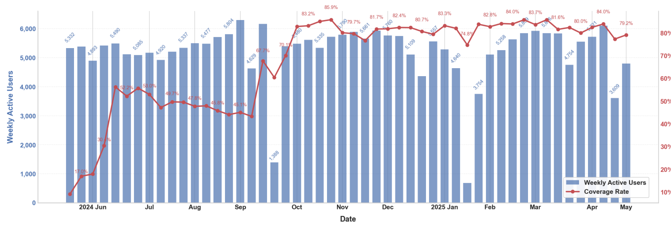

Fig. 6: Weekly active user count and coverage rate.

*3) **Online Effectiveness**:* During the online deployment phase of LogSage, we evaluated its two-stage accuracy through manual assessment and measured the availability of its four automated remediation tools via automatic logging.

Two-Stage Accuracy. The two-stage accuracy was evaluated based on weekly random sample of 150 online cases, which were manually reviewed and scored. For the RCA stage, experienced engineers examined the corresponding CI/CD task records and failure logs, manually debugging each case to determine the true root cause, which was then compared against LogSage's output. The evaluation criteria for the solution generation stage are summarized in appendix A, focusing primarily on the relevance of the generated solution to the actual root cause, the relevance of the retrieved knowledge, and the executability of the final recommendation.