LogSieve: Task-Aware CI Log Reduction for Sustainable LLM-Based Analysis

Marcus Emmanuel Barnes
Faculty of Information
University of Toronto
Toronto, Ontario, Canada
marcus.barnes@utoronto.ca

Taher A. Ghaleb
Department of Computer Science
Trent University
Peterborough, Ontario, Canada
taherghaleb@trentu.ca

Safwat Hassan
Faculty of Information
University of Toronto
Toronto, Ontario, Ontario
safwat.hassan@utoronto.ca

Abstract

Logs are essential for understanding Continuous Integration (CI) behavior, particularly for diagnosing build failures and performance regressions. Yet their growing volume and verbosity make both manual inspection and automated analysis increasingly costly, time-consuming, and environmentally costly. While prior work has explored log compression, anomaly detection, and LLM-based log analysis, most efforts target structured system logs rather than the unstructured, noisy, and verbose logs typical of CI workflows.

We present *LogSieve*, a lightweight, *RCA-aware and semantics-preserving* log reduction technique that filters low-information lines while retaining content relevant to downstream reasoning. Evaluated on CI logs from 20 open-source Android projects using GitHub Actions, *LogSieve* achieves an average 42% reduction in lines and 40% reduction in tokens with minimal semantic loss. This pre-inference reduction lowers computational cost and can proportionally reduce energy use (and associated emissions) by decreasing the volume of data processed during LLM inference.

Compared with structure-first baselines (*LogZip* and random-line removal), *LogSieve* preserves much higher semantic and categorical fidelity (Cosine = 0.93, GPTScore = 0.93, 80% exact-match accuracy). Embedding-based classifiers automate relevance detection with near-human accuracy (97%), enabling scalable and sustainable integration of semantics-aware filtering into CI workflows. *LogSieve* thus bridges log management and LLM reasoning, offering a practical path toward greener and more interpretable CI automation.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; *Software maintenance tools*; • **Computing methodologies** → *Information extraction.*

Keywords

Continuous Integration, Log reduction, Large Language Models, LLMs, Semantic fidelity, Sustainability, Android

ACM Reference Format:
Marcus Emmanuel Barnes, Taher A. Ghaleb, and Safwat Hassan. . LogSieve: Task-Aware CI Log Reduction for Sustainable LLM-Based Analysis. In . ACM, New York, NY, USA, 12 pages.

1 Introduction

Logs are indispensable to Continuous Integration (CI) workflows, supporting tasks such as failure diagnosis, pipeline debugging, and observability. Prior work has shown that CI outcomes (e.g., breakages) interact with pipeline characteristics and their evolution, reinforcing the need for scalable diagnostics [12]. Yet their growing volume and verbosity make both manual inspection and automated analysis costly, time-consuming, and increasingly energy-intensive at scale [14, 19, 42]. These challenges are particularly acute in ecosystems like GitHub Actions, where logs are voluminous, noisy, and loosely structured. The problem is even more pronounced in Android projects, whose multi-stage build and testing pipelines produce heterogeneous, verbose logs dominated by low-information content [10]. Prior studies have shown that Android apps often contain less-maintained logs and are more complex to parse [16, 46], suggesting these problems carry over to their CI workflows.

The increasing adoption of large language models (LLMs) in software engineering has enabled a wide range of automation and reasoning tasks, but has also raised concerns about inference cost, latency, and environmental impact due to their sensitivity to input length [22]. In the context of CI, recent studies have explored the use of LLMs to automate tasks such as generating CI configurations [6, 13] and migrating pipelines across CI services [21]. LLMs are also increasingly used for failure and anomaly detection with explanation [40], where CI logs serve as the primary input. In these settings, inference cost and latency scale with log length, meaning that every redundant line increases computational, financial, and carbon overhead. Existing log-reduction methods focus primarily on *structure-first* goals—compression, deduplication, or template clustering—to save storage or enable indexing [8, 18, 26, 28]. Such techniques improve infrastructure efficiency but overlook the *task relevance* of individual lines for downstream reasoning. In contrast, a *task-aware* perspective prioritizes semantic fidelity: retaining information essential for diagnosing and categorizing failures while discarding contextual noise, consistent with task-aware reduction strategies explored in other LLM pipelines [2]. This shift from structure-first to semantics-first reduction is crucial for sustainable, interpretable CI automation, where reducing token budgets directly lowers both inference cost and environmental impact [25, 30, 38]. In this paper, we focus on root-cause analysis (RCA) and failure categorization for GitHub Actions CI logs. Accordingly, task relevance is defined in terms of preserving the information necessary to support these diagnostic activities.

We present *LogSieve*, a lightweight, RCA-aware log reduction technique that filters out low-information lines while retaining content relevant to downstream automation. Unlike compression-based approaches such as LogZip, *LogSieve* performs *pre-inference, semantics-aware* reduction to minimize token count without compromising interpretability. The goal is to preserve diagnostic fidelity while reducing redundant computation, enabling scalable and sustainable CI analytics.

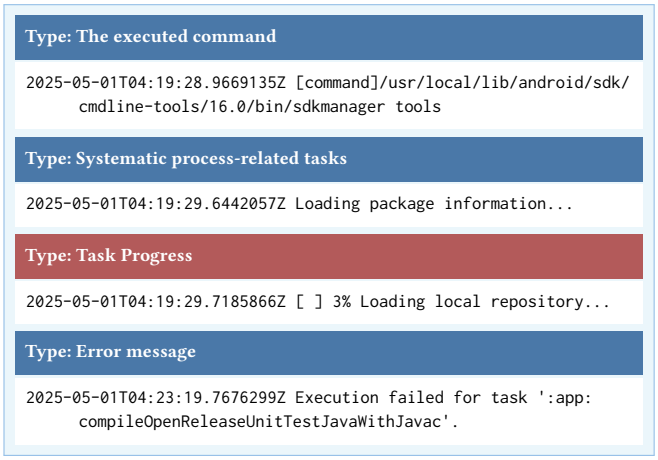

Figure 1: Examples of LogSieve reductions on Android CI log lines. RCA relevance: red = irrelevant, blue = relevant.

To summarize, this paper makes the following contributions:

- We propose *LogSieve*, a *RCA-aware, semantics-preserving* log reduction technique that filters low-information lines while retaining content relevant to failure diagnosis, bridging the gap between structure-first compression and LLM reasoning.
- We evaluate *LogSieve* on CI logs from 20 open-source Android projects using GitHub Actions, achieving an average 42% reduction in lines and 40% reduction in tokens with minimal loss of diagnostic fidelity.
- We assess GPT-4o performance on failure explanation and categorization tasks, finding strong semantic alignment (Cosine = 0.93, GPTScore = 0.93) and 80% exact-match accuracy, demonstrating that reduced logs preserve the information needed for downstream reasoning.
- We demonstrate that embedding-based classifiers can automate log-line relevance detection with near-human accuracy (97%), enabling scalable, sustainable integration of semantics-aware filtering into CI workflows.

Section 2 reviews related work, Section 3 describes our study design, and Section 4 presents our research questions and analyses. Section 5 discusses implications, Section 6 outlines threats to validity, and Section 7 concludes.

2 Related Work

2.1 CI/CD Log Analysis and Mobile CI Context

Logs play a central role in CI/CD workflows, enabling developers to diagnose failures, monitor pipelines, and assess build health. However, CI logs, especially in mobile ecosystems, tend to be long, unstructured, and noisy. This reflects the broader complexity and heterogeneity of CI workflows [1], which is also evident in Android projects [10] and can result in logs containing many low-information lines. Prior studies of mobile applications have shown that Android apps often exhibit less-maintained and harder-to-parse logs [16, 46], suggesting these tendencies extend to their CI workflows. In root cause analysis for CI/CD workflows, logs are typically examined manually to find recurring patterns linked to different failure or error types [4, 11]. This becomes increasingly challenging in mobile CI environments, where logs are large, noisy, and heterogeneous. Despite these issues, the empirical analysis and reduction of CI logs remain relatively underexplored. Prior studies have examined how CI outcomes, such as build breakages, relate to pipeline characteristics and evolution, highlighting the importance of scalable diagnostic signals within CI workflows [12].

2.2 Classical Log Reduction and Compression

Traditional log analysis has focused on redundancy removal and anomaly detection. Yao et al. [44] improved compression methods for log management, while Xu et al. [43] applied template mining to detect large-scale system anomalies. Techniques such as *LogZip* [28] and *LogShrink* [26] aim to improve storage and indexing efficiency but often retain task-irrelevant content. Likewise, Drain [18] and Spell [8] abstract logs structurally but remain agnostic to the *semantic relevance* of individual lines for developer tasks. These compression and parsing techniques serve infrastructure needs (e.g., log indexing, deduplication) but are ill-suited to LLM-based reasoning, where unnecessary tokens inflate inference cost and obscure failure context.

2.3 LLM-based Log Understanding

Recent work has begun to explore large language models for log interpretation. Qi et al. [35] introduced *LogGPT*, a transformer-based approach for anomaly detection, while Huang et al. [23] proposed *LoFI*, a prompt-driven method for extracting fault-indicating lines. However, these systems rely on LLM inference during the reduction process itself, increasing computational cost and limiting scalability. In contrast, *LogSieve* performs pre-inference reduction, removing irrelevant lines before LLM reasoning while maintaining semantic fidelity.

2.4 Sustainability and Software Engineering Implications

Beyond efficiency, CI log verbosity also poses sustainability challenges. Each additional token processed by an LLM contributes to computational and energy overhead, compounding the environmental cost of model-assisted analysis [25, 30, 38]. By reducing token counts prior to inference, *LogSieve* demonstrates how sustainability can be operationalized within software engineering workflows through lightweight, semantics-aware preprocessing.

2.5 Summary and Research Gap

Task-aware reduction has recently been explored in other LLM-based pipelines to control inference cost by prioritizing task-relevant inputs, though not in the context of CI logs or failure diagnosis [2]. Prior work has primarily optimized logs for storage compression, structural abstraction, or inference-time filtering. While such methods improve infrastructure efficiency, they often neglect the *semantic relevance* of log content for downstream developer tasks. *LogSieve* addresses this gap by introducing RCA-aware, pre-inference reduction that explicitly models semantic relevance rather than syntactic redundancy.

Building on this distinction, our study explores four key research questions that evaluate *LogSieve* from multiple perspectives: (1) the

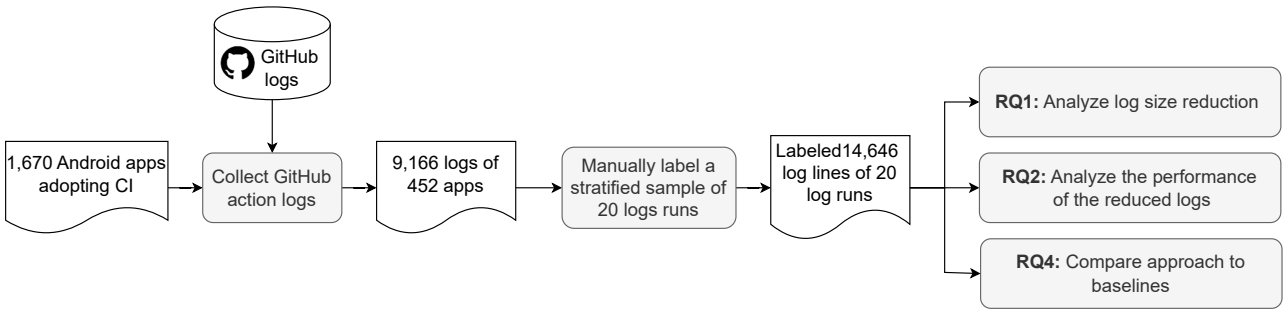

Figure 2: An overview of LogSieve evaluation.

potential for reducing CI log size while preserving diagnostic information (RQ1), (2) the degree to which reduced logs retain semantic fidelity for LLM-based analysis (RQ2), (3) the feasibility of automating relevance classification using embeddings and traditional ML models (RQ3), and (4) the comparative advantages of RCA-aware filtering over structural or random baselines (RQ4). Together, these questions bridge the gap between structure-first log management and semantics-first, sustainable CI analytics. Table 7 summarizes how *LogSieve* compares to existing techniques.

3 Study Design

This study investigates how RCA-aware log reduction affects the accuracy and efficiency of LLM-based automation in Continuous Integration (CI) workflows. Figure 2 and Figure 3 summarize the end-to-end workflow, from data collection and manual labeling to model training and evaluation. The design emphasizes empirical replicability, quantitative rigor, and fair comparison across reduction strategies. This staged structure mirrors how automation typically evolves in practice: manual annotation establishes a trusted ground truth, automated classification enables scalable deployment, and LLM-based evaluation verifies that the reduced logs preserve diagnostic fidelity. By progressing from human judgment to machine learning and finally to LLM validation, each phase builds on the previous one to ensure that efficiency gains never compromise interpretability or practical relevance.

3.1 Dataset Collection

Motivated by prior Android CI/CD studies [10], we analyzed CI logs from 20 open-source Android projects that use GitHub Actions, selected to ensure diversity in workflow size, complexity, and logging patterns. To construct this corpus, we first collected 9,166 GitHub Actions workflow runs from 452 Android repositories; among these, 7,045 runs completed successfully, 1,806 failed, 219 were cancelled, and 96 were skipped. Because *LogSieve* targets failure diagnostics, we focused on the failed runs. We drew a stratified random sample of 100 failed workflows to ensure coverage across project sizes, workflow configurations, and failure types (e.g., compilation, dependency, and environment errors). From this sample, we randomly selected 20 failed runs—one per repository—for detailed manual analysis, forming the annotated corpus used throughout RQ1–RQ4. Logs were collected via the GitHub REST API as of March 11, 2025; given GitHub's 90-day retention policy [15], this dataset provides a timely snapshot of active CI usage. The dataset and supporting artifacts used in this study are publicly available as part of our replication package [3].

3.2 Manual Annotation and Ground Truth Construction

Each log line was manually labeled as RCA-relevant (1) or RCA-irrelevant (`0`) for understanding the cause of CI failure. Labeling followed a guideline adapted from prior CI/CD studies [10], emphasizing the retention of diagnostic content (e.g., error traces, exception messages, task-level summaries) and removal of boilerplate or setup information (e.g., timestamps, dependency downloads, environment variables). Two co-authors independently labeled all 20 logs (14,646 lines in total). Inter-rater reliability was high (Cohen's $\kappa = 0.80$) [7]. Disagreements were resolved in consensus meetings with a third co-author, yielding a validated ground truth dataset that serves as the foundation for automation experiments. In total, the annotators initially disagreed on 41 log lines. Representative contentious cases included whether lines related to accepting Android SDK licenses or specific `GITHUB_TOKEN` permission settings should be retained as RCA-relevant in a CI context, as opposed to routine command or configuration details that were ultimately deemed irrelevant. Annotation was conducted independently by the two co-authors using a shared spreadsheet environment, ensuring that identical line ordering and contextual information were available during review. After labeling, we compared annotations programmatically to identify disagreements, which were then discussed in consensus meetings with the third author until full alignment was reached. This process not only standardized decisions on borderline cases but also reinforced inter-rater consistency, yielding a reproducible ground-truth dataset for downstream automation experiments.

The labeling guidelines were informed by the authors' collective experience in software engineering and debugging. When reviewing lines, we looked for cues indicating whether a line merely described routine processes or environment setup versus one that conveyed actionable diagnostic information such as error traces, exception messages, or test failures. Lines serving readability purposes only—such as progress indicators or decorative separators—were labeled as irrelevant, as they increase verbosity without aiding root-cause reasoning. These heuristics operationalize practical developer intuition about what constitutes meaningful versus superficial log

content, anchoring the study's focus on RCA-aware reduction for failure diagnosis.

3.3 Model Training and Evaluation Setup

To evaluate whether relevance classification can be automated at scale (RQ3), we trained a suite of supervised machine-learning classifiers using embeddings derived from three representation families. (1) **TF–IDF** captured lexical statistics and co-occurrence patterns, providing a strong and computationally inexpensive baseline for sparse text data. (2) **BERT** (`bert-base-uncased`) represented dense contextual semantics pre-trained on large natural language corpora, enabling sensitivity to surrounding log context such as error traces and stack frames. (3) **LLaMA3** (`Meta-Llama-3-8B-Instruct`) offered instruction-tuned embeddings that reflect how modern LLMs represent task prompts. Embeddings were precomputed and used as fixed feature vectors without fine-tuning, ensuring reproducibility and avoiding retraining overhead that would undermine *LogSieve*'s lightweight and sustainable design goals.

The classifier suite spanned diverse learning paradigms: linear models (*Logistic Regression*, *Linear SVM*, *SGD Logistic*); kernel and neural models (*SVM RBF*, *MLP*); ensemble methods (*Random Forest*, *XGBoost*, *LightGBM*); and baselines (*Nearest Centroid*, *Dummy Classifier*). This diversity balances interpretability and scalability: linear and tree models can be efficiently deployed in CI pipelines, while kernel and neural variants test the upper bound of achievable accuracy. For models benefiting from normalization or dimensionality reduction, pipelines applied *StandardScaler* and optionally *PCA*, with component counts tuned via grid search over {32, 64, 128, 256}. These component sizes approximate the dimensionalities of typical log-parsing embeddings, ensuring comparability across model families.

Training used stratified 10-fold cross-validation with an 80/20 train–test split to preserve class balance between relevant and irrelevant lines. This procedure produced 10 train/test partitions (one per fold), minimizing sampling bias and providing stable estimates of generalization accuracy. We report accuracy, weighted F1, precision, and recall on the held-out test set.

All models were implemented using scikit-learn [34] and open-source gradient-boosting libraries XGBoost [5] and LightGBM [24].

3.4 Evaluation Metrics and LLM Tasks

We computed four primary metrics to evaluate reduction performance and semantic preservation:

Line-level reduction (*Line-Red*): The percentage of log lines removed ($Removed\_Lines \times 100 / Total\_Lines$).

Token-level reduction (*Token-Red*): The percentage of tokens removed ($Removed\_Tokens \times 100 / Total\_Tokens$), measured using OpenAI's `tiktoken`[1] tokenizer.

Semantic similarity: We compared GPT-4o responses generated from reduced versus full logs using cosine similarity [37], BERTScore [45], ROUGE-1/L [27], BLEU [31], and GPTScore [9]. These metrics collectively capture both semantic and lexical fidelity.

Classification agreement: Exact-match accuracy between full- and reduced-log predictions for the categorization task (Prompt 2), quantifying whether reduction alters the categorical outcome of LLM reasoning.

The following rationale clarifies why each metric was chosen and how they collectively capture both surface and semantic fidelity.

These metrics were selected to capture complementary aspects of information preservation. Cosine similarity quantifies vector-level alignment between embeddings and reflects coarse semantic overlap, while BERTScore measures fine-grained contextual correspondence using token-level attention. ROUGE and BLEU evaluate lexical overlap and surface variability, ensuring that semantic alignment is not achieved by ignoring syntactic fidelity. GPTScore provides an LLM-judged perspective on response quality that better correlates with human preferences than purely statistical metrics. Together, these measures balance lexical and semantic dimensions of fidelity, offering a holistic assessment of how much diagnostic information is retained after reduction.

We selected GPT-4o for this study because it represented the state of the art among accessible large language models at the time of experimentation and was available via a stable API interface. Our goal was to assess whether the *LogSieve* approach yields meaningful gains with a high-performing frontier model before extending evaluation to other architectures. In future work, we plan to replicate these experiments with both open-source and proprietary LLMs to determine whether *LogSieve*'s benefits are range-bound across model families. To assess information preservation, we used two GPT-4o prompts:

Prompt 1 (Explanation):

```
Here is the log output from a GitHub Action workflow
run: [log.txt]. In at most 500 words, please explain
why this workflow failed.
```

Prompt 2 (Categorization):

```
Here is the log output from a GitHub Action workflow
run: [log.txt]. Please provide a category for the
run failure. Only provide the answer. Do not include
any additional reasons or details.
```

Prompt 1 reflects GitHub's "Explain Error" Copilot use case, while Prompt 2 captures coarse-grained failure categorization as used in CI dashboards. Model responses for full versus reduced logs were compared using the metrics above. The resulting measurements (Tables 1–2) form the basis for RQ1–RQ4.

3.5 Baseline Comparison Design

Finally, for RQ4, we benchmarked *LogSieve* against two baselines: the compression-based technique *LogZip* [28] and a random-line removal control. Each baseline was applied to the same CI logs, and the resulting reduced logs were re-evaluated using the same GPT-4o prompts and similarity metrics. This design enables a fair comparison of *LogSieve* against structural and non-semantic reductions in terms of both compression ratio and semantic fidelity.

4 Research Questions

This study addresses four research questions designed to evaluate the effectiveness and scalability of *LogSieve*.

[1] https://github.com/openai/tiktoken

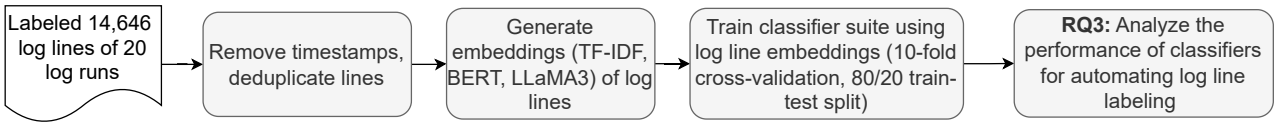

Figure 3: An overview of automating LogSieve.

4.1 RQ1: How much can CI log size be reduced by removing RCA-irrelevant lines?

Motivation. CI logs are often verbose and contain a mix of relevant and irrelevant information, particularly in failed workflows. Reducing this noise can lower processing costs and improve the focus of downstream tools. This RQ investigates the extent to which log size can be reduced by removing RCA-irrelevant lines (root cause analysis in this context), providing insight into the potential efficiency gains of lightweight pre-reduction.

Approach. We began with 9,166 GitHub Actions (GA) workflow runs collected from 452 unique Android repositories that adopted CI pipelines. Of these, 7,045 runs completed successfully, 1,806 failed, 219 were cancelled, and 96 were skipped. Because LogSieve targets failure-related diagnostics, we focused on failed runs only. From this subset, we created a stratified sample of 100 failed runs to ensure representation across diverse project sizes, workflow configurations, and failure types (e.g., compilation, dependency, and environment errors). We then randomly selected 20 runs from this stratified sample for detailed manual analysis, forming the dataset used for the root-cause analysis (RCA) and failure categorization tasks evaluated in this paper.

Each log line within these 20 runs was manually annotated as *RCA-relevant* (1) or *RCA-irrelevant* (0) for understanding the failure. Annotation guidelines were adapted from prior CI/CD empirical studies [10], emphasizing the retention of diagnostic evidence (e.g., error traces, exception messages, task-level summaries) and the removal of routine or boilerplate information (e.g., timestamps, dependency downloads, environment setup). Two co-authors independently labeled all lines using these guidelines. Inter-rater reliability, computed via Cohen's κ, was 0.80, indicating strong agreement [7]. Disagreements were discussed in consensus meetings with a third co-author until full alignment was reached, ensuring a consistent ground truth. The finalized annotations were then used to measure both line-level and token-level reductions after removing RCA-irrelevant content.

Results. Table 1 summarizes both line- and token-level reductions across projects. On average, 42% of log lines and 40% of tokens were removed. Some logs were highly reducible (e.g., `marunjar/anewjku`, 85% line reduction), while others retained most content (e.g., `cyb3rko/flashdim`, 11%).

These results suggest that CI logs—especially those associated with failed workflows—contain substantial portions of non-essential lines. *LogSieve* offers a generalizable method for reducing log verbosity prior to LLM-based processing, without requiring system-specific tuning.

Table 1: Log and token reduction across Android repositories.

Repository	Total Lines	Removed Lines	Lines Kept	Line Red.	Full Tokens	Tokens Kept	Token Red.
rafayali/movies	277	109	168	39%	8,834	5,775	35%
mcastillof/FakeTraveler	276	101	175	37%	8,372	6,025	28%
meditohq/medito	41	10	31	24%	1,090	813	25%
SecUSo/backup	2,052	1,253	799	61%	74,157	28,373	62%
fm-sys/snapdrop	268	85	183	32%	10,598	7,416	30%
alex/MonsterComp	188	37	151	20%	5,442	4,501	17%
thesandipv/watchdone	2,192	291	1,901	13%	67,613	60,155	17%
Vishnu/Quotes	866	489	377	56%	29,130	13,835	53%
cyb3rko/flashdim	386	43	343	11%	15,814	14,641	7%
hide1202/MovieDB	1,400	603	797	43%	45,367	26,784	41%
marunjar/anewjku	522	444	78	85%	16,737	2,792	83%
CrazyM/ToDont	196	58	134	27%	6,459	5,091	21%
dashpay/wallet	721	518	203	72%	23,868	7,595	68%
Graphene/PdfView	401	133	268	33%	12,231	8,771	28%
aivan/kpassnotes	1,157	949	208	82%	36,626	7,100	81%
ygg/android	27	4	23	15%	1,069	792	26%
metabrainz/MBZ	200	79	121	40%	6,470	4,304	33%
lollipopkit/flutter	206	84	122	41%	6,955	4,316	38%
TrianguloY/URL	1,714	941	773	55%	100,344	29,723	70%
ofalvai/HabitBuilder	1,550	787	772	50%	53,681	28,296	47%
Average	**732**	**350**	**381**	**42%**	**26,543**	**13,355**	**40%**

RQ1 summary: CI logs are highly redundant. On average, *LogSieve* removed **42% of lines** and **40% of tokens** while preserving key diagnostic context. Pre-inference RCA-aware filtering can reduce cost and proportionally cut energy use (and associated emissions) without sacrificing interpretability.

4.2 RQ2: To what extent do reduced logs preserve semantic information necessary for LLM-based root cause analysis?

Motivation. LLMs are increasingly used for root cause analysis and failure triage in CI and cloud systems. Recent studies have shown that LLM-based agents can match or outperform traditional methods in diagnosing incidents [36, 41]. However, these approaches typically rely on full log data. It remains unclear whether reduced logs preserve enough diagnostic information to support accurate LLM-driven analysis.

Approach. We evaluated GPT-4o on two CI-related tasks: failure explanation and failure classification. As detailed in Section 3, we used two prompts and applied them to both full and reduced logs: *Prompt 1* for generating explanations, and *Prompt 2* for classifying failure types. We assessed *Prompt 1* using semantic similarity metrics (Cosine Similarity, BERTScore, GPTScore) and lexical metrics (ROUGE-1/L, BLEU). We used exact match accuracy for *Prompt 2*. This setup quantifies whether reduced logs retain the context necessary for effective LLM responses.

Table 2: LLM Response Evaluation Across Repositories: Semantic Similarity and Classification Accuracy

Repository	CosSim	BERT-F1	R1-F1	RL-F1	BLEU	GPT Score	Exact Match
rafayali/movies	0.98	0.85	0.82	0.66	0.53	1.00	1
mcastillof/FakeTraveler	0.92	0.77	0.72	0.34	0.34	0.90	1
meditohq/medito	0.94	0.76	0.73	0.49	0.38	0.95	1
SecUSo/backup	0.84	0.74	0.68	0.49	0.40	0.90	0
fm-sys/snapdrop	0.94	0.77	0.68	0.32	0.32	0.90	1
alex/MonsterComp	0.94	0.80	0.74	0.45	0.41	0.90	0
thesandipv/watchdone	0.97	0.82	0.78	0.57	0.51	0.95	1
Vishnu/Quotes	0.95	0.82	0.79	0.56	0.49	0.95	1
cyb3rko/flashdim	0.96	0.82	0.73	0.57	0.54	0.90	0
hide1202/MovieDB	0.94	0.85	0.79	0.56	0.56	0.95	1
marunjar/anewjku	0.86	0.72	0.62	0.29	0.28	0.80	0
CrazyM/ToDont	0.96	0.78	0.74	0.45	0.38	1.00	1
dashpay/wallet	0.97	0.85	0.79	0.62	0.51	0.95	1
Graphene/PdfView	0.91	0.76	0.65	0.37	0.32	0.90	1
aivan/kpassnotes	0.92	0.74	0.65	0.30	0.23	0.90	1
ygg/android	0.96	0.79	0.76	0.50	0.46	1.00	1
metabrainz/MBZ	0.85	0.78	0.70	0.44	0.37	0.90	1
lollipopkit/flutter	0.95	0.79	0.74	0.50	0.50	0.95	1
TrianguloY/URL	0.95	0.78	0.71	0.43	0.37	0.90	1
ofalvai/HabitBuilder	0.93	0.81	0.75	0.44	0.44	0.90	1
Min	**0.84**	**0.72**	**0.62**	**0.29**	**0.23**	**0.80**	**Total 16 / 20 (80%)**
Mean	**0.93**	**0.79**	**0.73**	**0.47**	**0.42**	**0.93**	
Median	**0.94**	**0.78**	**0.73**	**0.47**	**0.41**	**0.90**	
Max	**0.98**	**0.85**	**0.82**	**0.66**	**0.56**	**1.00**	

Table 3: Performance of ML classifiers across embeddings for automated RCA-relevance classification. Bold values indicate the best-performing model within each embedding family.

Embedding	Model	Accuracy	F1 (w)	Precision (w)	Recall (w)
TF–IDF	Logistic Regression (L2)	**0.971**	**0.971**	**0.971**	**0.971**
	MLP (small)	0.968	0.968	0.968	0.968
	XGBoost	0.967	0.967	0.968	0.967
	LightGBM	0.967	0.967	0.967	0.967
	Random Forest	0.967	0.967	0.967	0.967
	SVM RBF	0.966	0.966	0.966	0.966
	Nearest Centroid	0.963	0.963	0.963	0.963
	Linear SVM	0.958	0.958	0.959	0.958
	SGD Logistic	0.944	0.944	0.945	0.944
	Dummy (stratified)	0.504	0.504	0.504	0.504
BERT	Logistic Regression (L2)	**0.971**	**0.971**	**0.971**	**0.971**
	MLP (small)	0.968	0.968	0.968	0.968
	XGBoost	0.967	0.967	0.968	0.967
	LightGBM	0.967	0.967	0.967	0.967
	Random Forest	0.967	0.967	0.967	0.967
	SVM RBF	0.966	0.966	0.966	0.966
	Nearest Centroid	0.963	0.963	0.963	0.963
	Linear SVM	0.958	0.958	0.959	0.958
	SGD Logistic	0.944	0.944	0.945	0.944
	Dummy (stratified)	0.504	0.504	0.504	0.504
LLaMA3	SVM RBF	**0.972**	**0.972**	**0.973**	**0.972**
	SGD Logistic	0.972	0.972	0.973	0.972
	MLP (small)	0.971	0.971	0.971	0.971
	LightGBM	0.971	0.971	0.971	0.971
	XGBoost	0.971	0.971	0.971	0.971
	Logistic Regression (L2)	0.969	0.969	0.969	0.969
	Linear SVM	0.969	0.969	0.969	0.969
	Random Forest	0.966	0.966	0.967	0.966
	Nearest Centroid	0.854	0.855	0.855	0.854
	Dummy (stratified)	0.500	0.500	0.500	0.500

Results. Table 2 presents semantic similarity scores between responses generated from full and reduced logs. Cosine Similarity averaged 0.93, BERTScore (F1) was 0.79, and the GPTScore—derived from a secondary GPT judgment—averaged 0.93. While lexical metrics such as ROUGE-L (0.47) and BLEU (0.42) were more sensitive to surface variation, overall semantic alignment remained strong.

For Prompt 2, the last column of Table 2 shows that 16 out of 20 reduced-log responses matched the full-log classification exactly (80% accuracy). This result is notable given an average token reduction of 40%, demonstrating that *LogSieve* preserves key diagnostic information required for LLM-driven analysis.

These findings suggest that *LogSieve* enables meaningful reductions in log size without sacrificing the quality of LLM responses for CI failure diagnosis.

RQ2 summary: Despite a 40% token reduction, GPT-4o responses maintained high fidelity (**CosSim = 0.93**, **GPTScore = 0.93**, and **80% exact-match accuracy**). RCA-aware reduction preserves essential semantics, enabling efficient and accurate LLM-based CI log analysis.

4.3 RQ3: How effective are machine learning models using different embeddings at automating log reduction?

Motivation. Building on the manually labeled ground truth from RQ1 and RQ2, we next investigated whether embeddings could automate the relevance classification process. While manual labeling is useful for pilot evaluation, it is not feasible in practice. To make *LogSieve* applicable to real-world CI workflows, relevance classification must be automated. Embedding-based models provide a natural mechanism for this task by capturing semantic features of log lines and supporting binary classification into RCA-relevant and RCA-irrelevant categories.

Approach. We evaluated three embedding families to determine their effectiveness for automated log reduction:

- **TF–IDF**: a sparse lexical representation capturing term frequency and inverse document frequency,
- **BERT (bert-base-uncased)**: dense contextual embeddings pre-trained on large text corpora, and
- **LLaMA3 (Meta-Llama-3-8B-Instruct)**: instruction-tuned embeddings representing modern large language model features.

For each embedding type, we trained ten supervised classifiers spanning linear, kernel, neural, and ensemble families: Logistic Regression (L2), Linear SVM, SGD Logistic, SVM with RBF kernel, a small MLP, Random Forest, XGBoost, LightGBM, Nearest Centroid, and a Dummy (stratified) baseline. Embeddings were treated as fixed feature vectors (not fine-tuned), and all models were trained under identical 10-fold stratified cross-validation. Hyperparameters for PCA-based models were tuned over {32, 64, 128, 256} components using grid search. Model performance was evaluated on a held-out 20% test set using accuracy, weighted F1, precision, and recall.

Results. Table 3 presents the performance of all classifiers across the three embedding families. All embeddings produced strong results, confirming that both lexical and contextual representations can support accurate automation of log-line relevance classification.

For **TF–IDF** embeddings, *Logistic Regression (L2)* achieved the best accuracy of **0.97**, with tree- and boosting-based models (*Random Forest*, *XGBoost*, *LightGBM*) and the *SVM (RBF)* kernel following closely (≥ 0.96). This shows that sparse lexical features already capture much of the discriminative signal between relevant and irrelevant lines, such as the co-occurrence of failure keywords, file-path patterns, or stack-trace indicators.

For **BERT** embeddings, *Logistic Regression (L2)* again obtained the highest accuracy (**0.97**), and most other models performed within a small margin. The similarity of BERT and TF–IDF scores suggests that contextual embeddings provide only marginal gains for this binary task, likely because CI-log language and structure are relatively stable across projects and contain limited ambiguity compared with natural text.

For **LLaMA3** embeddings, the best result was produced by *SVM (RBF)* (**0.97**), followed closely by *SGD Logistic* and *MLP* (0.97 each). Although LLaMA3 embeddings are high-dimensional and instruction-tuned, they did not yield substantial improvements over simpler representations for this classification setting. Nevertheless, their comparable performance confirms that *LogSieve* can integrate seamlessly with instruction-tuned embedding services if such representations are already available within an organization's LLM infrastructure.

Across all embeddings, weighted F1, precision, and recall remained consistently high (≈ 0.96–0.97), indicating balanced performance and minimal class bias.

Overall, these results demonstrate that classical machine-learning classifiers can emulate human relevance judgments with near-human accuracy and low computational cost, providing a practical path for scaling *LogSieve* to large industrial CI environments without invoking expensive LLM inference.

RQ3 summary: Classical machine-learning classifiers, such as *Logistic Regression* and *SVM (RBF)*, achieve near-human labeling accuracy across all embedding types. These results demonstrate that *LogSieve* can reliably automate RCA-relevance detection at scale within CI pipelines.

4.4 RQ4: How does *LogSieve* compare to baseline approaches?

Motivation. While RQ1–RQ3 examined *LogSieve*'s intrinsic reduction and automation performance, this question contextualizes its effectiveness against two alternative baselines:

- **LogZip**. A widely used compression-based technique that clusters repetitive log templates to reduce redundancy without considering RCA-relevance [28]. This baseline tests whether structural compression alone can preserve the information required for LLM-based analysis.
- **Random-line removal**. A sanity-check baseline that randomly removes lines to match *LogSieve*'s average 42% reduction rate. Following standard practice in model evaluation [29], this ensures that improvements are due to semantic relevance rather than reduction volume.

Approach. For each repository, we produced three reduced versions of the logs: (i) *LogSieve*-reduced (RCA-aware filtering), (ii) *LogZip*-reduced (template-based compression), and (iii) *Random*-reduced (random line removal). We re-evaluated these logs on the same GPT-4o downstream tasks from RQ2—*Prompt 1* (failure explanation) and *Prompt 2* (failure categorization)—using identical metrics: Cosine Similarity (CosSim), BERTScore F1, ROUGE-1/L F1, BLEU, GPTScore, and exact-match accuracy (Prompt 2 only). We report the mean values across the 20 repositories to compare semantic and categorical fidelity under equivalent reduction ratios.

Table 4: RQ4 comparison of *LogSieve*, *LogZip*, and *Random* baselines against full logs. Means across 20 repositories (macro-averages).

Prompt 1 (Failure Explanation): mean similarity to Full						
	CosSim	BERT-F1	R1-F1	RL-F1	BLEU	GPTScore
LogSieve	**0.93**	**0.79**	**0.73**	**0.47**	**0.42**	**0.93**
LogZip	0.70	0.68	0.51	0.26	0.09	0.41
Random	0.90	0.76	0.66	0.38	0.23	0.86
Prompt 2 (Failure Categorization): mean scores vs Full						
	CosSim	BERT-F1	R1-F1	RL-F1	BLEU	GPTScore
LogSieve	**0.92**	**0.93**	**0.87**	**0.87**	**0.29**	**0.94**
LogZip	0.51	0.67	0.23	0.23	0.07	0.47
Random	0.87	0.89	0.77	0.77	0.29	0.89
Prompt 2: Exact-match Accuracy (Full vs. Reduced)						
LogSieve	0.80					
LogZip	0.20					
Random	0.70					

Results. Table 4 compares the three reduction strategies. Across both tasks, *LogSieve* consistently outperformed the structural (*LogZip*) and random baselines. For *Prompt 1* (failure explanation), *LogSieve* achieved the highest semantic alignment with full logs (CosSim = 0.93, GPTScore = 0.93), compared to *LogZip* (CosSim = 0.70) and *Random* (CosSim = 0.90). For *Prompt 2* (failure categorization), *LogSieve* preserved categorical accuracy (exact-match = 0.80), compared to 0.20 for *LogZip* and 0.70 for *Random*. Although the random baseline occasionally retained surface similarity, it degraded categorical fidelity and interpretability, confirming that *LogSieve*'s RCA-aware filtering—rather than reduction magnitude—drives superior preservation of LLM reasoning fidelity. Table 4 reports macro-averaged performance across repositories; per-repository breakdowns for the baselines appear in Tables 5 and 6.

To further understand variation across projects, we analyzed per-repository results for the baseline methods. As shown in Tables 5 and 6, *LogZip*'s performance varies widely across repositories (CosSim = 0.54–0.95; EM = 20%), reflecting sensitivity to template diversity, whereas *Random* reduction remains comparatively stable (CosSim $\approx$ 0.90; EM = 70%). This dispersion underscores the advantage of *LogSieve*'s RCA-aware filtering, which maintains both high semantic similarity and consistent classification fidelity across heterogeneous projects.

Table 5: LLM response evaluation by repository for *LogZip*: semantic similarity to Full (Prompt 1) and exact-match classification agreement (Prompt 2).

Repository	CosSim	BERT-F1	R1-F1	RL-F1	BLEU	GPT Score	Exact Match
rafayali/movies	0.56	0.63	0.35	0.19	0.01	0.00	0
mcastillof/FakeTraveler	0.88	0.75	0.69	0.34	0.21	0.90	1
meditohq/medito	0.63	0.70	0.55	0.27	0.11	0.20	0
SecUSo/backup	0.91	0.71	0.66	0.43	0.26	0.90	1
fm-sys/snapdrop	0.60	0.66	0.41	0.17	0.02	0.30	0
alex/MonsterComp	0.94	0.79	0.73	0.48	0.28	0.90	1
thesandipv/watchdone	0.68	0.69	0.54	0.24	0.11	0.00	0
Vishnu/Quotes	0.71	0.66	0.51	0.24	0.07	0.60	0
cyb3rko/flashdim	0.86	0.72	0.54	0.31	0.12	0.90	0
hide1202/MovieDB	0.66	0.65	0.45	0.19	0.01	0.30	0
marunjar/anewjku	0.84	0.73	0.63	0.32	0.12	0.90	0
CrazyM/ToDont	0.59	0.66	0.47	0.21	0.02	0.20	0
dashpay/wallet	0.56	0.61	0.45	0.18	0.01	0.20	0
Graphene/PdfView	0.62	0.64	0.44	0.18	0.04	0.00	0
aivan/kpassnotes	0.95	0.79	0.72	0.46	0.34	0.90	1
ygg/android	0.54	0.64	0.44	0.20	0.05	0.00	0
metabrainz/MBZ	0.62	0.64	0.43	0.20	0.01	0.20	0
lollipopkit/flutter	0.56	0.64	0.45	0.19	0.04	0.20	0
TrianguloY/URL	0.78	0.65	0.46	0.22	0.02	0.30	0
ofalvai/HabitBuilder	0.56	0.61	0.39	0.18	0.01	0.20	0
Min	**0.54**	**0.61**	**0.35**	**0.17**	**0.01**	**0.00**	**Total 4 / 20 (20%)**
Mean	**0.70**	**0.68**	**0.51**	**0.26**	**0.09**	**0.41**	
Median	**0.65**	**0.66**	**0.46**	**0.21**	**0.05**	**0.25**	
Max	**0.95**	**0.79**	**0.73**	**0.48**	**0.34**	**0.90**	

Table 6: LLM response evaluation by repository for *Random* reduction: semantic similarity to Full (Prompt 1) and exact-match classification agreement (Prompt 2).

Repository	CosSim	BERT-F1	R1-F1	RL-F1	BLEU	GPT Score	Exact Match
rafayali/movies	0.92	0.78	0.70	0.41	0.27	0.90	0
mcastillof/FakeTraveler	0.92	0.80	0.74	0.41	0.36	0.90	1
meditohq/medito	0.93	0.80	0.76	0.52	0.38	1.00	1
SecUSo/backup	0.93	0.76	0.67	0.42	0.27	1.00	0
fm-sys/snapdrop	0.91	0.75	0.66	0.30	0.18	0.90	1
alex/MonsterComp	0.93	0.77	0.68	0.43	0.28	0.90	1
thesandipv/watchdone	0.90	0.75	0.69	0.35	0.22	0.90	1
Vishnu/Quotes	0.88	0.72	0.63	0.29	0.15	0.90	1
cyb3rko/flashdim	0.91	0.75	0.60	0.36	0.22	0.90	1
hide1202/MovieDB	0.93	0.80	0.66	0.39	0.23	0.90	1
marunjar/anewjku	0.91	0.75	0.65	0.36	0.20	0.90	1
CrazyM/ToDont	0.88	0.79	0.69	0.37	0.21	0.80	0
dashpay/wallet	0.95	0.76	0.68	0.37	0.15	0.90	1
Graphene/PdfView	0.90	0.78	0.70	0.39	0.26	0.90	1
aivan/kpassnotes	0.93	0.74	0.62	0.37	0.20	0.90	0
ygg/android	0.58	0.64	0.44	0.20	0.03	0.00	0
metabrainz/MBZ	0.90	0.78	0.70	0.39	0.26	0.90	1
lollipopkit/flutter	0.91	0.77	0.67	0.41	0.27	0.90	1
TrianguloY/URL	0.94	0.76	0.68	0.45	0.27	0.90	1
ofalvai/HabitBuilder	0.92	0.79	0.70	0.47	0.26	0.90	0
Min	**0.58**	**0.64**	**0.44**	**0.20**	**0.03**	**0.00**	**Total 14 / 20 (70%)**
Mean	**0.90**	**0.76**	**0.66**	**0.38**	**0.23**	**0.86**	
Median	**0.92**	**0.76**	**0.67**	**0.38**	**0.24**	**0.90**	
Max	**0.95**	**0.80**	**0.76**	**0.52**	**0.38**	**1.00**	

RQ4 summary: The baseline comparison confirms that *LogSieve* achieves much higher semantic and categorical fidelity than both structural and random reductions at equivalent reduction ratios. While *LogZip* effectively compresses redundant patterns, it retains RCA-irrelevant lines that reduce interpretability. The Random baseline shows that volume reduction without modeling semantic relevance yields lower categorical fidelity and interpretability. Thus, *LogSieve*'s RCA-aware filtering, not reduction magnitude, drives superior preservation of LLM reasoning fidelity.

5 Discussion

This section interprets our findings across RQ1–RQ4 and discusses their implications for automated, sustainable CI analytics. Together, the results show that RCA-aware log reduction can substantially decrease inference cost while preserving diagnostic fidelity.

5.1 Automation Feasibility

Our automation experiments (RQ3) show that log-line relevance can be reliably inferred using embedding-based classifiers, suggesting that manual labeling can be eliminated for most CI contexts. Embedding features capture both syntax and semantics of log text, achieving up to 97% accuracy across models. These results highlight the practicality of integrating *LogSieve* as a continuously learning pre-filter in CI systems, where embeddings can be updated incrementally as logs evolve.

In practice, *LogSieve* can operate as an intermediate step between log collection and artifact storage, automatically tagging or filtering lines with confidence scores produced by the relevance classifier. A configurable threshold determines whether a line is retained or suppressed, allowing practitioners to balance interpretability and compression according to project risk or compliance needs. Low-confidence cases can be routed to full logs or developer review, providing a human-in-the-loop safeguard that maintains transparency during adoption. This flexible integration model enables gradual rollout in production pipelines without disrupting existing CI infrastructure.

To sustain performance over time, retraining can occur periodically or be triggered when classifier confidence drops below a set threshold, signaling drift in workflow or log structure. Because training and inference are lightweight, retraining can run offline or on dedicated CI runners with minimal overhead. Together, these features position *LogSieve* as a practical, low-maintenance automation component that reduces human effort while preserving the interpretability and auditability essential for modern software delivery.

5.2 Effectiveness Depends on Log Quality

Aggressive reduction, when guided by RCA-relevance, lowers input size without sacrificing LLM effectiveness. Most classification mismatches in Table 2 stemmed from vague or underspecified logs rather than information loss. This suggests that the quality and structure of the source logs fundamentally bound the achievable fidelity of reduction. Logs that contain clear stack traces, compiler diagnostics, or structured error messages allow both humans and models to identify relevant lines reliably, whereas loosely formatted or incomplete logs make relevance prediction inherently ambiguous. Future CI pipelines could therefore benefit from adopting structured logging conventions or schema-based instrumentation to maximize the downstream utility of semantics-aware reduction.

Building on these observations, *LogSieve* can also be paired with lightweight domain heuristics (e.g., compiler or toolchain detectors) to recover task context that is missing or weakly expressed in unstructured logs. Integrating such heuristics into preprocessing would ensure that key diagnostic tokens are never removed while still allowing aggressive reduction of routine setup and environment information. This trade-off between log quality and

Table 7: Conceptual comparison of *LogSieve* with log compression, structural parsing, and LLM-in-the-loop techniques.

Property	LogSieve	LogZip	Drain	Spell	LoFI
Semantic / task-aware relevance	✓	×	×	×	✓
Structural parsing or templates	–	✓	✓	✓	–
Token reduction before LLM	✓	×	×	×	×
Needs LLM during reduction	×	×	×	×	✓
Online / streaming capable	–	–	✓	✓	–
Main objective	Semantics	Compression	Parsing	Stream parsing	Fault signals
Evaluated on CI logs	✓	×	×	×	×
LLM inference overhead	Low	Low	Low	Low	High

reduction aggressiveness offers a promising direction for practical deployment, where developers can tune thresholds based on the predictability and maturity of their build environment.

5.3 Relevance Varies by Failure Type

Manual inspection revealed that noise levels differ across failure categories. Compilation and dependency errors tend to produce dense diagnostic traces, whereas permission or configuration issues often yield boilerplate output. This heterogeneity suggests that a single global reduction ratio may not be optimal for all workflows. Adaptive reduction strategies that condition on failure type, pipeline stage, or project domain could further improve both precision and recall of relevant lines. For example, during dependency resolution, a higher retention threshold may capture transient network or package errors, whereas build or test phases can tolerate stronger filtering once patterns of compiler diagnostics are learned. Incorporating such context-aware adaptation would allow *LogSieve* to evolve from a static pre-filter into a dynamic component that continuously learns relevance distributions as CI pipelines mature.

5.4 Empirical Comparison with LogZip

RQ4 demonstrates that *LogSieve* preserves semantic fidelity far better than *LogZip* at equivalent reduction levels. For explanation (Prompt 1), *LogSieve* improves similarity to full-log responses by roughly 20–35 percentage points across CosSim, ROUGE-1/ROUGE-L, and BLEU (0.93 vs. 0.70; 0.73 vs. 0.51; 0.47 vs. 0.26; 0.42 vs. 0.09), by 11 points on BERTScore (0.79 vs. 0.68), and by 52 points on GPTScore (0.93 vs. 0.41). For categorization (Prompt 2), exact-match accuracy increases from 0.20 to 0.80; the *Random* baseline reaches 0.70, reflecting higher surface similarity but lower categorical fidelity. These gaps indicate that structure-first compression can discard contextual cues essential for LLM reasoning, whereas RCA-aware filtering maintains interpretability while reducing token cost.

5.5 Position Relative to Existing Techniques

Traditional techniques such as *LogZip* [28] and *LogShrink* [26] focus on compression or deduplication for storage efficiency but overlook semantic relevance. *Drain* [18] and *Spell* [8] abstract syntax without modeling task importance, while *LoFI* [23] integrates LLMs into reduction but incurs inference overhead. In contrast, *LogSieve* performs semantics-aware filtering *before* inference, complementing structural tools in token-sensitive workflows.

Table 8: Resource summary (means across 20 repositories). Δ reports the macro-averaged percentage reduction; absolute means are shown in the first two columns. Token reduction dominates inference-cost and latency improvements.

	Full	LogSieve	Δ (%)
Mean input tokens / run	26,543	13,355	−40
Mean lines / run	732	381	−42

5.6 Sustainability and Cost Implications

RCA-aware pre-reduction directly lowers resource use for LLM inference and CI infrastructure. We do not directly measure energy consumption or emissions; instead, we report token reductions and discuss proportional cost/energy implications using established Green AI models. In our dataset, the average prompt size fell from 26,543 to 13,355 tokens per run (−40%), and the number of lines dropped from 732 to 381 (−42%). Because inference cost and latency scale with input tokens, these reductions translate to immediate savings in computation and can yield proportional reductions in energy use (and associated emissions) [20, 25, 30, 38, 42].

Parametric cost model. Following prior work on Green AI and energy-aware computation [20, 38, 42], we model inference cost as a linear function of token usage:

$$\text{Cost} = \frac{T_{\text{in}}}{1000}\, p_{\text{in}} + \frac{T_{\text{out}}}{1000}\, p_{\text{out}}.$$

Pre-reduction primarily decreases T_{in}. With T_{full}=26,543, T_{red}=13,355, and T_{rem}=13,188,

$$\Delta\text{Cost}_{\text{in}} = \frac{T_{\text{rem}}}{1000}\, p_{\text{in}} \approx 13.188\, p_{\text{in}} \text{ per run.}$$

If response length remains similar (as in categorical tasks), ΔT_{out} is negligible; otherwise the full expression applies.

Energy and carbon perspective. Following established energy–carbon estimation models for machine learning [32, 33, 39], we estimate emissions as

$$\Delta\text{CO}_2 = \Delta E \cdot CI_{\text{grid}}, \qquad \Delta E \propto T_{\text{in}}.$$

A 40 % token reduction therefore yields a proportional decrease in energy use for LLM-assisted CI analysis while also reducing bandwidth and storage requirements for log artifacts. These efficiency gains require no model retraining and complement structural tools by filtering semantically irrelevant content before inference.

Beyond immediate cost and energy considerations, such efficiency gains are particularly relevant in long-lived software ecosystems. Prior empirical studies have shown that ecosystems can undergo periods of slowdown and abandonment, during which maintenance and diagnostic responsibilities increasingly fall on a shrinking set of contributors [17]. In these contexts, reducing routine CI analysis overhead becomes critical, as developers must diagnose failures and maintain pipelines under constrained time and resource budgets. Lightweight, semantics-aware pre-reduction helps lower this burden without requiring changes to downstream tools or CI workflows.

5.7 Integrative Reflection

Across all analyses, *LogSieve* demonstrates that RCA-aware reduction can bridge the gap between traditional log compression and LLM-based understanding. By retaining semantics rather than syntax, it enables efficient, scalable, and environmentally conscious automation within CI workflows.

6 Threats to Validity

As with most empirical studies, our findings are subject to several threats to validity. We outline potential limitations and describe the measures taken to mitigate them.

Internal Validity. Potential bias arises from the manual labeling of log lines. Labels were assigned using heuristic criteria (e.g., prioritizing error messages and de-emphasizing setup steps), reflecting how reduction rules might be constructed in practice. To mitigate subjectivity, two co-authors labeled independently, disagreements were resolved with a third, and analyses emphasized aggregate trends. A residual concern is *heuristic anchoring*—the possibility that labelers and models rely on similar cues. We mitigate this through publicly stated heuristics, transparent inter-rater reliability reporting (Cohen's κ=0.80 in RQ1), and evaluation on held-out folds.

Cross-validation also introduces potential for *data leakage* if lines from the same workflow appear across folds, which could inflate accuracy. We used stratified 10-fold cross-validation to maintain class balance, but future work will employ group-stratified CV by repository to further reduce this risk. Hyperparameter tuning on test folds can overstate performance; nested CV or a held-out validation set would be stricter. Finally, embeddings were treated as fixed features—fine-tuning or domain-adaptive pretraining could change results and should be explored in replication.

External Validity. Our dataset covers 20 open-source Android projects using GitHub Actions (failed runs). Findings may not generalize to other CI ecosystems (e.g., GitLab CI, Azure DevOps), programming languages, project scales, or log types (system, server, telemetry), nor to proprietary pipelines. The focus on failed runs and GitHub's 90-day retention window may bias toward recent, failure-heavy activity. Similarly, reliance on a single LLM (GPT-4o) limits generalizability; replication with other models and human judgments would strengthen conclusions. Broader validation across CI platforms, languages, and failure modes will enhance generalizability.

Construct Validity. We operationalize relevance using a binary retain/remove label, which simplifies how lines contribute to different diagnostic or summarization tasks. Prompts modeled two specific tasks (failure explanation and categorization); other CI analyses (e.g., repair suggestion) may require different evidence. Similarity metrics (Cosine, BERTScore, ROUGE, BLEU, GPTScore) approximate semantic fidelity but are imperfect proxies for developer utility—high textual similarity does not necessarily translate to faster debugging. We do not directly measure operational energy consumption or emissions; sustainability implications are inferred from token reductions under proportional energy–token assumptions used in prior Green AI models. GPTScore, as an LLM-based judge, may also introduce model-specific bias; triangulating with lexical metrics and exact-match accuracy partly mitigates this risk. Baseline configuration choices (e.g., *LogZip* parameters, random seeds) may also influence outcomes. We used default settings to reflect typical usage and report distributional summaries (median, min, max) to reduce interpretation bias.

Conclusion Validity. Our evaluation included 20 repositories, sufficient for an initial feasibility study but not for broad statistical generalization. While effect sizes are large, variance remains. Future iterations will include confidence intervals, bootstraps, and per-repository analyses. We also plan paired significance tests (e.g., McNemar) for classification agreement and random-seed robustness checks to validate reproducibility.

Reproducibility and Evolution Threats. LLM services evolve (e.g., model snapshots, safety filters, tokenizers), which may alter outputs over time. We fixed prompts, temperature, and tokenizer (`tiktoken`) and will release all artifacts—prompts, code, and labels—for replication. We recommend pinning model versions, logging random seeds, and exporting raw responses to mitigate drift. For automation experiments, training code and cross-validation splits will be released to support long-term reproducibility.

Ethical and Privacy Considerations. We used public open-source CI logs, which may still contain incidental identifiers. We avoided storing PII and will provide redaction guidelines in our replication package. For private CI environments, organizations should apply privacy filters and enforce access control. By releasing our code, labels, and prompts, we aim to support open, transparent, and ethical research in automated CI analytics.

7 Conclusion

We studied CI logs from 20 open-source Android projects on GitHub Actions (failed runs) and introduced *LogSieve*, a RCA-aware, semantics-preserving log reduction technique that filters low-information lines while retaining content relevant to downstream LLM-based analysis. Across repositories, *LogSieve* removed on average 42% of lines and 40% of tokens (RQ1). Despite these reductions, semantic fidelity remained high for explanation responses (Cosine = 0.93, GPTScore = 0.93), and failure categorization matched full-log labels in 80% of cases (RQ2). Embedding-based classifiers automated relevance labeling with near-human accuracy (up to 97%; RQ3). Compared with structure-first compression approaches such as *LogZip*, *LogSieve* preserved substantially more RCA-relevant semantics for both explanation and categorization (e.g., Cosine 0.93 vs. 0.70; exact-match 80% vs. 20%; RQ4). By reducing token budgets before inference, *LogSieve* advances sustainable, interpretable CI automation without modifying model weights or prompts.

Future Work. Future research will extend *LogSieve* in several directions. We plan to benchmark it against additional reduction and parsing techniques (e.g., *Drain* [18], *Spell* [8], *LoFI* [23]) on shared datasets and standardized downstream tasks to better contextualize trade-offs between structure-first and task-aware approaches. We also aim to automate relevance detection using heuristics and weak supervision, support adaptive reduction based on token budgets and failure types, and generalize to other log domains such as server or telemetry data. Beyond method extensions, we will

evaluate *LogSieve* with additional LLMs to assess model-agnostic robustness and quantify end-to-end energy and carbon savings in real deployments. We plan to explore hybrid pipelines that integrate *LogSieve* with structural parsers (e.g., *Drain*) to balance interpretability, scalability, and sustainability. Finally, we intend to complement LLM-based evaluation with a practitioner survey to assess whether the reduced logs are also more meaningful and interpretable for humans. This will help establish whether the task-aware reduction principles demonstrated here generalize beyond root-cause analysis to other CI and software-maintenance tasks, uniting human and model-centric perspectives on sustainable automation.

Artifacts and Replicability. We provide a publicly available replication package supporting this study. The artifacts are hosted on Figshare and can be accessed via https://doi.org/10.6084/m9.figshare.30811382. The package enables replication of the analyses reported in this paper and supports further investigation of task-aware CI log reduction, including RCA-focused analyses.

Acknowledgments

Funding: We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC): [RGPIN-2021-03969] and [RGPIN-2025-05897].

References

[1] Edward Abrokwah and Taher A Ghaleb. 2025. An Empirical Study of Complexity, Heterogeneity, and Compliance of GitHub Actions Workflows. *arXiv preprint arXiv:2507.18062* (2025). https://arxiv.org/abs/2507.18062

[2] Marcus Emmanuel Barnes, Taher A. Ghaleb, and Safwat Hassan. 2025. Task-Aware Reduction for Scalable LLM-Database Systems. In *2025 IEEE International Conference on Collaborative Advances in Software and COmputiNg (CASCON)*. IEEE Computer Society, Los Alamitos, CA, USA, 631–635. doi:10.1109/CASCON66301.2025.00114

[3] Marcus Emmanuel Barnes, Taher A. Ghaleb, and Safwat Hassan. 2026. Replication Package for "LogSieve: Task-Aware CI Log Reduction for Sustainable LLM-Based Analysis". doi:10.6084/m9.figshare.30811382

[4] Carolin E Brandt, Annibale Panichella, Andy Zaidman, and Moritz Beller. 2020. LogChunks: A data set for build log analysis. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 583–587. doi:10.1145/3379597.3387485

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794. doi:10.1145/2939672.2939785

[6] Łukasz Chomątek, Jakub Papuga, Przemyslaw Nowak, and Aneta Poniszewska-Marańda. 2025. Decoding CI/CD Practices in Open-Source Projects with LLM Insights. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*. 1638–1644.

[7] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46. doi:10.1177/001316446002000104

[8] Min Du and Feifei Li. 2016. Spell: Streaming Parsing of System Event Logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 859–864. doi:10.1109/ICDM.2016.0103

[9] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2024. 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)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 6556–6576. doi:10.18653/v1/2024.naacl-long.365

[10] Taher Ghaleb, Osamah Abduljalil, and Safwat Hassan. 2025. CI/CD Configuration Practices in Open-Source Android Apps: An Empirical Study. *ACM Transactions on Software Engineering and Methodology* (May 2025). doi:10.1145/3736758

[11] Taher Ahmed Ghaleb, Daniel Alencar da Costa, Ying Zou, and Ahmed E. Hassan. 2021. Studying the Impact of Noises in Build Breakage Data. *IEEE Transactions on Software Engineering* 47, 9 (2021), 1998–2011. doi:10.1109/TSE.2019.2941880

[12] Taher A Ghaleb, Safwat Hassan, and Ying Zou. 2023. Studying the interplay between the durations and breakages of continuous integration builds. *IEEE Transactions on Software Engineering* 49, 4 (2023), 2476–2497. doi:10.1109/TSE.2022.3222160

[13] Taher A Ghaleb and Dulina Rathnayake. 2025. Can LLMs Write CI? A Study on Automatic Generation of GitHub Actions Configurations. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 767–772.

[14] Sina Gholamian and Paul AS Ward. 2021. A comprehensive survey of logging in software: From logging statements automation to log mining and analysis. *arXiv preprint arXiv:2110.12489* (2021).

[15] GitHub, Inc. 2025. Usage Limits, Billing, and Administration. http://docs.github.com/en/actions/concepts/overview/usage-limits-billing-and-administration#artifact-and-log-retention-policy. GitHub Documentation (accessed July 10, 2025).

[16] Julian Harty, Haonan Zhang, Lili Wei, Luca Pascarella, Mauricio Aniche, and Weiyi Shang. 2021. Logging practices with mobile analytics: An empirical study on firebase. In *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. IEEE, 56–60. doi:10.1109/MobileSoft52590.2021.00013

[17] Kazi Amit Hasan, Jerin Yasmin, Huizi Hao, Yuan Tian, Safwat Hassan, and Steven H. H. Ding. 2025. Understanding Abandonment and Slowdown Dynamics in the Maven Ecosystem. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*. 354–358. doi:10.1109/MSR66628.2025.00065

[18] Pinjia He, Jieming Zhu, Zibin He, Jian Li, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40. doi:10.1109/ICWS.2017.13

[19] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37. doi:10.1145/3460345

[20] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research* 21, 248 (2020), 1–43.

[21] Md Nazmul Hossain and Taher A Ghaleb. 2025. CIgrate: Automating CI Service Migration with Large Language Models. *arXiv preprint arXiv:2507.20402* (2025).

[22] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79. doi:10.1145/3695988

[23] Junjie Huang, Zhihan Jiang, Jinyang Liu, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Cong Feng, Hui Dong, Zengyin Yang, and Michael R. Lyu. 2024. Demystifying and Extracting Fault-Indicating Information from Logs for Failure Diagnosis. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 511–522. doi:10.1109/ISSRE62328.2024.00055

[24] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[25] Christoph König, Daniel J. Lang, and Ina Schaefer. 2025. Sustainable Software Engineering: Concepts, Challenges, and Vision. *ACM Transactions on Software Engineering and Methodology* 34, 5 (2025), 1–28. doi:10.1145/3709352

[26] Xiaoyun Li, Hongyu Zhang, Van-Hoang Le, and Pengfei Chen. 2024. LogShrink: Effective log compression by leveraging commonality and variability of log data. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12. doi:10.1145/3597503.3608129

[27] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013/

[28] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R. Lyu. 2019. Logzip: Extracting Hidden Structures via Iterative Clustering for Log Compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 863–873. doi:10.1109/ASE.2019.00085

[29] Fadel M Megahed, Ying-Ju Chen, L Allison Jones-Farmer, Steven E Rigdon, Martin Krzywinski, and Naomi Altman. 2024. Comparing classifier performance with baselines. *Nature Methods* 21, 4 (2024), 546–548. doi:10.1038/s41592-024-02234-5

[30] Stefan Naumann, Daniel Schmidt, Marcel Dick, Jakob Kern, and Judith M. Müller. 2011. The GREENSOFT Model: A Reference Model for Green and Sustainable Software and Its Engineering. *Sustainable Computing: Informatics and Systems* 1, 4 (2011), 294–304. doi:10.1016/j.suscom.2011.06.004

[31] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318. https://aclanthology.org/P02-1040/

[32] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R So, Maud Texier, and Jeff Dean. 2022. The carbon footprint of machine learning training will plateau, then shrink. *Computer* 55, 7 (2022), 18–28. doi:10.1109/MC.2022.3148714

[33] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350* (2021).

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss,

Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[35] Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol J. Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. 2023. 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)*. IEEE, 273–280. doi:10.1109/HPCC-DSS-SMARTCITY-DEPENDSYS60770.2023.00045

[36] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. 2024. Exploring LLM-based agents for root cause analysis. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 208–219. doi:10.1145/3663529.3663841

[37] Gerard Salton and Michael J. McGill. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY.

[38] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (2020), 54–63. doi:10.1145/3381831

[39] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2020. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13693–13696.

[40] Jiabo Wang, Guojun Chu, Jingyu Wang, Haifeng Sun, Qi Qi, Yuanyi Wang, Ji Qi, and Jianxin Liao. 2024. LogExpert: Log-based recommended resolutions generation using large language model. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 42–46. doi:10.1145/3639476.3639773

[41] Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. 2024. RCAgent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4966–4974. doi:10.1145/3627673.3680016

[42] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. 2022. Sustainable AI: Environmental implications, challenges and opportunities. *Proceedings of machine learning and systems* 4 (2022), 795–813.

[43] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. 2009. Detecting Large-Scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. 117–132. doi:10.1145/1629575.1629587

[44] Kundi Yao, Mohammed Sayagh, Weiyi Shang, and Ahmed E. Hassan. 2022. Improving State-of-the-Art Compression Techniques for Log Management Tools. *IEEE Transactions on Software Engineering* 48, 8 (2022), 2748–2760. doi:10.1109/TSE.2021.3069958

[45] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. BERTscore: Evaluating text generation with BERT. *arXiv preprint arXiv:1904.09675* (2019).

[46] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130. doi:10.1109/ICSE-SEIP.2019.00021