Journal of Innovations in Computer Science and Engineering
Shahid Beheshti Unversity

January 2026, Volume 3, Issue 2

Predicting CI/CD Workflow Outcomes Using Time-Aware Feature Engineering and Gradient Boosting Models

Alireza Keshavarzi, **CODE ORCID:** 0009-0009-7257-6379
Department of Computer Engineering, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran,
alkeshavarzi@buqaen.ac.ir
Morteza Noferesti✉, **CODE ORCID:** 0009-0000-5507-1461
Department of Computer Engineering, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran,
mnoferesti@buqaen.ac.ir

ABSTRACT

The widespread adoption of Continuous Integration and Continuous Delivery (CI/CD) pipelines, particularly through platforms such as GitHub Actions, has introduced new challenges in ensuring pipeline reliability and predictability. Unexpected workflow failures can disrupt release cycles and increase operational costs. This paper presents a time-aware, data-driven approach for predicting CI/CD workflow execution outcomes. Historical failure patterns are extracted using time-aware cumulative aggregation. Feature engineering is performed incrementally, including Bayesian smoothing to stabilize sparse statistics, Last-K features (k = 5, 10) to capture short-term behavior, and interaction features to model nonlinear dependencies between entities such as repositories and workflow paths, forming the core feature-set (Base_Feature_Set). Additionally, textual features are extracted from workflow logs using regular expressions, capturing error presence, frequency, and message patterns, resulting in an enriched feature-set (Log-Enriched_Feature_Set). Using the large-scale GHALogs dataset, comprising over 513,000 workflow executions from approximately 25,000 repositories, we evaluate five models (Logistic Regression, Random Forest, HistGradientBoosting, XGBoost and CatBoost) under a consistent chronological split for both feature sets. Experimental results show that time-aware structured features alone yield moderate discriminative power (AUC 0.603–0.646). Adding log-derived features, which become available only after execution, raises the AUC of the final CatBoost model to 0.843. Because the log features describe the run being predicted, this improvement reflects post-execution failure-detection performance rather than pre-execution forecasting.

CC BY

KEYWORDS

Software Reliability, DevOps, Log-Based Feature Engineering, Applied AI, CI/CD Pipeline Outcome Prediction.

1. INTRODUCTION

Continuous Integration and Continuous Delivery/Deployment (CI/CD) have become foundational practices in modern software engineering, automating the build–test–deploy cycle and enabling rapid, reliable software delivery [1]. Continuous Integration is a repetitive and automated process in which every code change is immediately integrated into a central repository [2]. As illustrated in the left part of **Figure 1**, this process includes planning, coding, building, and testing. After each integration, automated tests, such as unit and system tests, are executed without manual intervention to ensure software quality and stability [2]. This approach addresses the challenges of code conflicts in collaborative environments where multiple developers work on a shared codebase. By transforming integration into a continuous activity, CI reduces risks and costs associated with large, infrequent merges, and ensures that the software remains in a deliverable state [3,4].

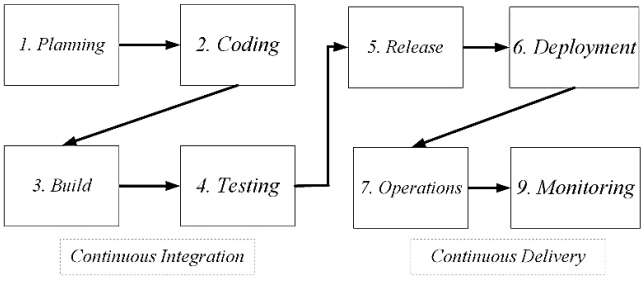


Figure 1. The DevOps lifecycle.

Submit Date: 2026-04-17
Revise Date: 2026-10-16
Accept Date: 2026-10-18
✉ Corresponding author

Continuous Delivery/Deployment (CD) extend the CI process by automating the release pipeline, as shown in the right part of **Figure 1**. This includes stages such as release, deployment, operations, and monitoring, forming a continuous and connected lifecycle. Continuous Delivery ensures that the software is always in a ready-to-release state, while Continuous Deployment goes a step further by automatically deploying every validated change to production without manual intervention [3]. These practices emerged to address the complexity of transferring software from development to production environments. By automating the entire pipeline, CD reduces human errors, accelerates software delivery, and improves reliability. Together with CI, these concepts form the foundation of the DevOps approach, which emphasizes collaboration between development and operations teams, continuous feedback, and efficient, high-quality software delivery [5].

While the core ideas of continuous integration date back more than a decade [2], the widespread adoption of cloud-native platforms, especially GitHub Actions [6], has increased both the scale and the operational complexity of pipelines. Recent large-scale studies and industry reports show that pipeline failures remain frequent, costly, and disruptive: main-branch success rates have declined in some ecosystems [7], inconsistent builds affect a substantial fraction of projects [8], and even short outages or delayed feedback can accumulate into significant developer time and compute waste [9].

The growing volume of execution data, historical outcomes, and full logs has led to machine-learning approaches that try to predict pipeline outcomes before or during execution [10-13]. Recent research on GitHub Actions has targeted several of these challenges directly: build-failure prediction frameworks such as GHA-BFP [14], dataset efforts such as GHALogs that emphasize temporal validity to avoid data leakage [15], multi-class failure prediction [16], and AI-augmented reliability frameworks [17]. Despite this progress, many current models still depend mainly on structured metadata or older platforms, make little use of the rich information in execution logs, or fail to build features in a strictly time-aware way. As a result, reported performance can be inflated by leakage, and the practical value of these predictions for real GitHub Actions workflows stays limited.

The research questions guiding this study are:

- RQ1: How does time-aware, incremental feature engineering affect the predictive performance of CI/CD workflow outcome prediction models?
- RQ2: To what extent does incorporating textual log-derived features, in addition to structured historical features, improve the predictive performance of the model?

In this paper, we propose a time-aware approach for modeling and predicting the behavior of Continuous Integration and Continuous Delivery (CI/CD) pipelines. The approach begins with the systematic collection of large-scale, real-world execution data, followed by preprocessing to ensure data quality and consistency. We then perform a multi-stage feature engineering process that incrementally constructs informative representations of pipeline behavior. This process includes (i) historical features derived from past executions using time-aware aggregation, (ii) interaction features that capture dependencies among workflow components, and (iii) textual features extracted from execution logs using regular-expression-based text processing. These feature types are organized into two progressively enriched feature-sets: *Base_Feature_Set*, which contains structured historical and interaction features, and *Log-Enriched_Feature_Set*, which further incorporates log-derived textual signals.

The central principle of the proposed approach is temporal validity, ensuring that each prediction is based solely on information available up to the time of execution, thereby preventing data leakage and preserving realistic evaluation conditions. This principle fully governs the Base_Feature_Set; the log-derived features of the Log-Enriched_Feature_Set are, by construction, available only after (or during) execution, so that part of the study concerns post-execution failure detection and is evaluated separately. Using a large-scale dataset comprising over 513,000 workflow executions from approximately 25,000 public GitHub repositories [15], we develop and systematically evaluate predictive models for forecasting pipeline outcomes. Specifically, a *HistGradientBoostingClassifier* is employed as one of several baseline models (alongside Logistic Regression, Random Forest and XGBoost) due to its efficiency and ability to capture nonlinear relationships under time-aware data splitting, while a more advanced *CatBoostClassifier* is utilized to leverage both structured and textual features, with additional threshold tuning to optimize predictive performance. By integrating structured operational data with unstructured log-based features, the log-enriched variant of the proposed approach achieves higher discriminative performance (AUC) than the structured features alone. Ultimately, this approach enables proactive decision-making in CI/CD environments by providing risk scores for prioritizing workflow runs, reducing operational risks, optimizing resource allocation, and enhancing the overall efficiency of software delivery processes.

The main contributions of this paper are as follows:

1. A time-aware, incremental feature engineering pipeline for structured historical features that prevents data leakage by strictly relying on information available before each workflow execution.
2. An integration of historical, interaction-based, and log-derived textual features for CI/CD outcome prediction.
3. An empirical evaluation of five classification models on the large-scale GHALogs dataset, showing higher AUC when log-based features are added, while explicitly distinguishing the pre-execution (structured) from the post-execution (log-based) setting.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 details the proposed methodology. Section 4 presents the experimental setup and results. Section 5 discusses the findings, and Section 6 concludes the paper.

2. RELATED WORK

Growing research has applied artificial intelligence and machine learning to improve the reliability, efficiency, and predictability of Continuous Integration and Continuous Delivery pipelines.

2.1 AI-Driven Approaches for DevOps and Testing

Vadde and Munagandla [18] examined the effectiveness of Continuous Integration and Continuous Delivery pipelines by analyzing historical test results, code changes, system logs, defect rates, code complexity metrics, and infrastructure performance indicators. The authors integrated automated testing tools with widely used platforms such as Jenkins, GitLab CI, Docker, and Kubernetes. Their results showed improvements in deployment frequency, reductions in change failure rate, and better mean time to recovery, illustrating the practical value of intelligent automation in production environments.

Pandhare [19] focused on integrating AI into software testing within CICD pipelines. Using heterogeneous datasets from Google, Facebook, IBM and other sources, the work applied machine learning to analyze logs and code changes, natural language processing to convert requirements into executable test cases, reinforcement learning to optimize test selection, and computer vision for detecting visual defects in graphical interfaces. Clustering and anomaly detection were used for test data analysis, while self-healing scripts automatically repaired broken tests. Reported benefits included a reduction of up to 60% in regression testing time, approximately 90% accuracy in defect prediction, a 40% increase in test coverage, and a substantial decrease in test maintenance costs.

Mishra and Sharma [20] proposed a deep learning–based approach for bug prediction using data collected from multiple open-source GitHub projects. The dataset included code changes, CI/CD pipeline information, and bug-related records. Their methodology combined SMOTE for handling class imbalance, data normalization, feature extraction with F-BERT and a focal loss function, and hierarchical logistic regression for fine-tuning. Experimental results demonstrated strong performance, achieving 95.32% accuracy and 93.3% recall.

2.2 GitHub Actions and Build Outcome Prediction

More recent studies have turned attention to the modern GitHub Actions platform and to the specific problem of predicting build or workflow outcomes.

Li et al. [14] introduced GHA-BFP, an automated framework designed specifically for predicting build failures in GitHub Actions. Using machine learning models (with Random Forest performing best), the framework achieved accuracy, precision, recall, and F1-score all exceeding 75%. Through ablation experiments the authors confirmed the importance of four categories of input features and further analyzed the relative importance of individual features.

Mishra and Otaiwi [11] addressed a methodological issue in build-outcome prediction: temporal data leakage. They proposed a three-type taxonomy of leakage including direct outcome encoding, execution-dependent metrics, and future information leakage and validated it on two large datasets spanning a decade: TravisTorrent [21] and GHALogs [15]. Removing leaky features reduced accuracy by 15.07 percentage points on TravisTorrent but only 0.48 points on GHALogs, revealing that platform architecture strongly influences the severity of leakage. Using only clean pre-build features, Random Forest models still achieved approximately 83% accuracy on both platforms, demonstrating that realistic, leakage-free prediction is feasible.

Ge and Zhang [8] conducted a large-scale empirical study of flaky builds in GitHub Actions based on rerun data from 1,960 open-source Java projects. They found that 3.2% of builds were rerun and that 67.73% of those reruns exhibited flaky behavior, affecting more than half of the studied projects. The authors identified 15 distinct categories of flaky failures and proposed an ML-based detector for identifying flaky failures at the job level, improving F1-score by up to 20.3% over a strong baseline.

Shah et al. [22] examined the reliability of CI/CD workflows triggered by pull requests generated by AI coding agents including Claude, Devin, Cursor, Copilot, and Codex. Analyzing 61,837 workflow runs from 2,355 repositories, they observed substantial differences in success rates across agents. At the repository level they found a negative correlation between the frequency of AI-agent contributions and workflow success rate. They also constructed a taxonomy of 13 failure categories for the agentic pull requests that led to failed workflows.

Table 1 summarizes the key characteristics of the most representative prior works. The studies demonstrate the effectiveness of machine learning in advancing defect prediction, test optimization, anomaly detection, and build-outcome forecasting. However, this body of work exhibits several recurring limitations:

- Many approaches rely primarily on structured metadata or target older CI platforms, rather than modern, widely adopted systems such as GitHub Actions.
- Strict time-aware feature construction that prevents data leakage is not consistently enforced, raising concerns about the validity of reported performance.
- The rich textual information contained in execution logs is rarely exploited systematically for outcome prediction, despite its potential to capture failure signals unavailable in structured metadata alone.
- Evaluation of log-derived features on log-rich datasets such as GHALogs for binary workflow-outcome prediction remains limited.

This paper directly addresses these gaps by proposing a strictly time-aware feature-engineering pipeline that integrates historical statistics, interaction features, and log-derived textual signals, and by evaluating the resulting models on the large-scale GHALogs dataset under a consistent chronological split, while clearly distinguishing the pre-execution setting (structured features) from the post-execution setting (log-derived features).

Table 1. The related works summarization.

	Data Source	Methods	Key Results
Vadde and Munagandla [18]	Jenkins, GitLab CI, Docker	Predictive analysis, anomaly detection	Daily deployment frequency; reduced change failure rate
Pandhare [19]	Google, Facebook, IBM	NLP, GANs, Self-healing	60% test time reduction; 90% prediction accuracy
Mishra and Sharma [20]	GitHub open-source projects	F-BERT, Bi-CGRU, M-SMOTE, HLR Optimization	95.32% accuracy; 93.3% Recall
Li et al. [14]	GitHub Actions builds	Random Forest and other ML models	Accuracy, Precision, Recall, F1 all > 75%
Mishra and Otaiwi [11]	TravisTorrent + GHALogs	Temporal leakage taxonomy + Random Forest	Leakage-free accuracy 83%
Ge and Zhang [8]	GitHub Actions (1,960 Java projects)	Empirical analysis + ML flaky-failure detector	67.73% of reruns are flaky; F1 improved by up to 20.3%
Shah et al. [22]	GitHub Actions (AI-agent PRs)	Empirical reliability analysis	Success rates differ across agents; negative correlation with agent-PR frequency
The proposed approach	GHALogs (513k runs)	Time-aware historical + interaction + log features;	Accuracy 0.695, balanced accuracy 0.752, AUC 0.843 (post-execution logs; chronological split)

3. METHODOLOGY

In this section, the proposed research approach for modeling and predicting the outcomes of CI/CD pipeline executions is presented. The approach is developed based on a time-aware and hierarchical design, aiming to realistically simulate decision-making conditions in operational environments by leveraging real-world execution data. The proposed approach consists of several stages, including data collection and preprocessing, extraction of historical, interaction-based, and textual features, definition of target variables, construction of incremental versions of the feature-set, and ultimately the training of advanced machine learning models. The fundamental principle underlying this methodology is that each stage of the structured-feature pipeline relies solely on information available up to the time of each workflow execution, thereby ensuring that model evaluation accurately reflects real-world performance in GitHub-based environments. The detailed components of this approach are described in the following subsections.

3.1 Preprocessing

Initially, all records containing incompatible or corrupted textual data that could not be converted into standard encoding formats (UTF-8 or ASCII) were completely removed to prevent the introduction of noise and unreliable inputs into the learning process. In this study, no imputation techniques were applied; only structurally and semantically valid data were retained for further analysis. This decision was made deliberately to avoid introducing artificial bias and to enhance the reliability and integrity of the results.

After removing records with corrupted or non-decodable textual fields, the dataset comprised 214 columns. To reduce dimensionality while retaining the most informative predictors, a multi-method feature selection procedure was applied exclusively to the training data, thereby preventing information leakage from the validation and test sets.

Three complementary techniques were employed to assess feature relevance:

1. Feature importance derived from a Random Forest classifier,
2. Mutual information between each feature and the binary target variable,
3. Feature importance scores obtained from an XGBoost classifier.

For each method, importance scores were min-max normalized to the range [0, 1] to ensure comparability across techniques with differing scales. The three normalized scores were then averaged to produce a single consensus importance ranking. A feature was retained if it satisfied at least one of the following criteria: (i) an average normalized importance ≥ 0.05, or (ii) a ranking within the top 40 features by at least two of the three methods.

3.2 Dataset Description

For training and validating the predictive models, the large-scale public dataset GHALogs is utilized [15]. This dataset contains real-world execution data from CI/CD pipelines based on GitHub Actions. GHALogs comprises approximately 513,000 complete workflow executions collected from around 25,000 public repositories, covering more than 20 different programming languages [15]. It is recognized as one of the largest publicly available datasets of CI/CD logs, providing comprehensive access to detailed execution-level information across pipeline stages.

The dataset is loaded in a structured tabular format, where each record represents a complete workflow execution, as outlined in **Table 2**. The available metadata include information on the final execution status, project-level attributes, workflow-specific characteristics, and temporal details of pipeline runs. Upon initial inspection, the feature-set consists of 214 columns, capturing various dimensions of CI/CD pipeline behavior. This rich and high-dimensional feature-set forms the foundation for subsequent preprocessing, feature engineering, and modeling stages in the proposed methodology.

3.3 Feature-Sets

A temporal index is constructed from the pair (*run_id, run_attempt*), which serves as a strictly ordered proxy for execution time. We prefer this synthetic index over wall-clock timestamps because GitHub Actions does not guarantee unique, monotonically increasing timestamps across concurrent and retried runs; the native identifiers provide a total order free of clock skew and missing values. The data are sorted by these two fields so that chronological order is preserved.

The binary target *y* is obtained from *metadata.conclusion*: only runs whose conclusion is "success" or "failure" are retained; all other conclusions ("cancelled", "skipped", "timed_out", etc.) are discarded. After filtering we encode success = 0 and failure = 1 (failure is therefore the positive class in every reported precision, recall, F1 and MCC value; see Section 4.2).

To capture historical patterns without leakage, smoothed failure rates are computed for the entities repository name, workflow path and associated users. Rates are obtained by a time-aware cumulative aggregation followed by a unit lag, so that each record sees only past executions. The rates are Bayesian-smoothed with a Beta(1,1) prior (equivalent to one pseudo-success and one pseudo-failure). When an entity has no prior history its rate defaults to the prior mean 0.5 (cold-start handling). Interaction features are formed by multiplying the corresponding entity-level rates.

In addition, for each entity we maintain a Last-*K* window consisting of the *K* most recent prior runs of that entity (ordered by (*run_id, run_attempt*)); reruns of the same *run_id* appear as separate observations. If fewer than *K* prior runs exist the window contains all available history and is subject to the same cold-start rule. We compute these features for *K=5* and *K=10*. All of the above statistics are therefore strictly causal.

Next, we generate interaction features that capture more complex relationships between variables. These are constructed by combining related failure rates (such as repository-level and workflow-level failure rates). The purpose of this step is to enable models to learn nonlinear dependencies and joint effects among different workflow components. The resulting feature-set is named *Base_Feature_Set* and serves as the foundation for modeling.

Furthermore, log data (*log_insights*) from the raw dataset is incorporated alongside features from *Base_Feature_Set*. Using regular expression patterns, features such as error presence, error counts, and ratios of specific log messages are extracted. This step enriches the feature-set with text-derived signals, named as *Log-Enriched_Feature_Set*.

Table 2. Sample Features and Corresponding Values in a Workflow Execution.

Feature	Value	Feature	Value
metadata.triggering_actor .login	[anonymized]	metadata.event	schedule
_id	intel/pluginskubernetes.github/workflows/e2e. yaml	metadata.run_attempt	1
repository_name	intel/intel-device-plugins-for-kubernetes	metadata.triggering_actor .type	User
metadata.repository.name	intel-device-plugins-for-kubernetes	metadata.actor.type	User
metadata.repository .description	Collection of Intel device plugins for Kubernetes	metadata.repository.owner .type	Organization
metadata.head_repository .description	Collection of Intel device plugins for Kubernetes	metadata.head_repository.fork	FALSE
metadata. head_repository.name	intel-device-plugins-for-kubernetes	metadata.head_repository .owner.type	Organization
workflow_path	.github/workflows/e2e.yaml	metadata.repository.fork	FALSE
total_logs_size	3944668	run_attempt	1
metadata.head_repository .owner.login	intel	metadata.head_repository .private	FALSE
metadata.name	e2e	metadata.repository.owner.site_ad min	FALSE
metadata.head_commit .author.name	[anonymized]	metadata.actor.site_admin	FALSE
time_to_import	64899.13866	metadata.triggering_actor .site_admin	FALSE
metadata.path	.github/workflows/e2e.yaml	metadata.repository.owner .login	intel
metadata.head_commit .message	Merge pull request #1593 from [anonymized]/build-flag-changesbuild: use stricter build flags	metadata.repository.private	FALSE
metadata.pull_requests	[]	metadata.conclusion	failure

To enrich the structured features, we extracted textual signals from the raw execution logs (log_insights) using regular expressions. The extraction process focused on patterns that are commonly associated with workflow failures. The resulting log-derived features fall into three main categories:

1. Error Presence Indicators: Binary features that indicate whether specific error-related keywords or patterns appear in the log (such as the presence of “error”, “exception”, “failed”, “timeout”, “permission denied”, or “exit code”).
2. Error Frequency Counts: Integer features that count the number of occurrences of selected error patterns within a

single workflow run (including number of stack-trace lines, number of "ERROR" level messages, number of failed step indicators).

3. Normalized Ratios: Features that capture the relative prevalence of error-related messages, computed as the count of matching lines divided by the total number of log lines (or total log size). These ratios help normalize for workflows of different lengths.

The extracted features were concatenated with the *Base_Feature_Set* to form the *Log-Enriched_Feature_Set*. All regular-expression matching was performed in a case-insensitive manner, and empty or missing logs were assigned zero values for the corresponding features. Because these features are derived from the logs of the run being predicted, they are available only after (or during) execution and constitute post-execution information; the resulting model is therefore a failure-detection model rather than a pre-execution forecaster.

3.4 Prediction models

Several machine learning models are used to assess the effectiveness of the feature sets. On the Base_Feature_Set, we evaluate Logistic Regression, HistGradientBoosting, Random Forest, XGBoost, and CatBoost. For all experiments we employ a strict chronological split (80 % training / 20 % testing), in which later executions are held out for testing. This chronological splitting strategy prevents temporal leakage and reflects realistic deployment conditions in which only past information is available at prediction time. The primary evaluation metric is AUC, complemented by accuracy, precision, recall, F1-score, balanced accuracy and MCC.

Hyperparameters for all models were selected using a time-aware validation scheme to avoid temporal leakage. Specifically, the chronological training set was further split into an earlier training portion (80%) and a later validation portion (20%). We performed a grid search over the following key hyperparameters:

- CatBoost: depth ∈ {6, 8, 10}, learning_rate ∈ {0.03, 0.1}, l2_leaf_reg ∈ {1, 3, 5}, iterations = 500 (with early stopping)
- XGBoost: max_depth ∈ {6, 8, 10}, learning_rate ∈ {0.05, 0.1}, subsample ∈ {0.8, 1.0}, n_estimators = 300
- Random Forest: n_estimators ∈ {100, 200}, max_depth ∈ {10, 20, None}, min_samples_leaf ∈ {1, 5}
- HistGradientBoosting: max_depth ∈ {6, 10}, learning_rate ∈ {0.05, 0.1}, max_iter = 300
- Logistic Regression: C ∈ {0.1, 1.0, 10.0}, class_weight = 'balanced'

The combination that maximized validation AUC was selected for each model and then used to retrain on the full chronological training set before final evaluation on the held-out test set. Default values were used for all remaining hyperparameters.

Finally, we train the same set of models on the Log-Enriched_Feature_Set using exactly the same chronological split. This allows a direct comparison between the two feature sets under an identical model family and identical data-splitting strategy. Because log-derived features become available only after workflow execution, the resulting models perform post-execution failure detection rather than pre-execution prediction. The final CatBoost model integrates all previously engineered historical, interaction, and log-based features and constitutes the complete modeling pipeline proposed in this work.

4. EXPERIMENTAL RESULTS

Experiments were conducted on a system with an Intel Core i9-13980HX CPU, 16GB RAM, and an NVIDIA GeForce RTX 4060 GPU, running Ubuntu 24.10. Python libraries included pandas, numpy, scikit-learn, re, PyTorch, and xgboost.

4.1 Evaluation results over different feature-sets

Table 3 presents the performance of the evaluated models under a consistent chronological 80/20 split. On the Base_Feature_Set (historical and interaction features only), the models achieve moderate results. CatBoost obtains the highest accuracy (0.600) and the best AUC (0.646) among the base models. XGBoost (accuracy 0.490, AUC 0.642), Logistic Regression (accuracy 0.509, AUC 0.631) and HistGradientBoosting (accuracy 0.553, AUC 0.630) follow closely in AUC, whereas Random Forest is the weakest (accuracy 0.462, AUC 0.603). Given the failure prevalence of 17.7 %, all accuracy values are below the 0.8226 of a trivial majority-class classifier; therefore, models are ranked by AUC and balanced accuracy rather than by accuracy. These results indicate that the engineered structured features already capture useful predictive signal, yet overall discriminative power remains limited.

When log-derived textual features are added (Log-Enriched_Feature_Set), the final CatBoost model achieves higher scores, reaching an accuracy of 0.695, precision of 0.350, recall of 0.842, F1-score of 0.495, and an AUC of 0.843. This corresponds to an absolute gain of approximately 0.197 in AUC compared with the best base-feature CatBoost model. The gains of CatBoost in AUC, balanced accuracy, precision, recall, F1-score and MCC indicate that execution logs carry information that is not captured by structured historical and interaction features alone. Overall, the results suggest that time-aware structured features alone provide only moderate discrimination and that log-based textual signals contribute most of the observed improvement. It should be noted that the log features are post-execution information; the Log-Enriched models therefore perform failure detection rather than pre-execution prediction.

Table 3**. Models' performance across feature-sets (both sets evaluated under the same chronological 80/20 split).**

Model	Feature-set Version	Accuracy	Balanced	Precision	Recall	F1-	MCC	AUC

			Accuracy			score		
HistGradientBoosting	*Base_Feature_Set*	0.553	0.592	0.231	0.651	0.341	0.140	0.630
CatBoost	*Base_Feature_Set*	0.600	0.605	0.247	0.611	0.352	0.161	0.646
Random Forest	*Base_Feature_Set*	0.462	0.567	0.209	0.730	0.325	0.106	0.603
XGBoost	*Base_Feature_Set*	0.490	0.594	0.223	0.752	0.344	0.146	0.642
Logistic Regression	*Base_Feature_Set*	0.509	0.589	0.223	0.712	0.340	0.137	0.631
Random Forest	*Log-Enriched_Feature_Set*	0.633	0.688	0.296	0.772	0.428	0.288	0.762
XGBoost	*Log-Enriched_Feature_Set*	0.642	0.623	0.269	0.594	0.371	0.192	0.672
CatBoost	*Log-Enriched_Feature_Set*	0.695	0.752	0.350	0.842	0.495	0.390	0.843

4.2 Class Distribution and Imbalance-Aware Evaluation

The GHALogs dataset exhibits class imbalance, with successful workflow executions forming the majority class. To avoid inflated performance estimates caused by this imbalance, we primarily rely on the Area Under the ROC Curve (AUC) as the main evaluation metric, as it is robust to skewed class distributions, and we complement it with balanced accuracy (the mean of recall on failed and successful runs), the Matthews Correlation Coefficient (MCC). MCC is a single score from -1 to +1 that measures the correlation between predicted and true labels using all four confusion-matrix counts (TP, TN, FP, FN), so it remains reliable under class imbalance, with 0 indicating performance no better than chance.

After preprocessing, the dataset contains 513,992 workflow executions, of which 422,830 (82.26 %) concluded successfully and 91,162 (17.74 %) failed (imbalance ratio 4.64:1). Failed executions are treated as the positive class in all precision, recall, F1, and MCC metrics.

We apply the same chronological split to both the Base_Feature_Set and the Log-Enriched_Feature_Set experiments. Because both feature sets are evaluated on identical training and test periods, the comparison isolates the contribution of the additional log-derived features. The resulting test partitions preserve a failure rate close to the overall dataset ratio of 17.74 %.

A trivial classifier that always predicts success achieves an accuracy of 82.26%, a balanced accuracy of 0.500 and an AUC of 0.500, and detects none of the failures (recall = 0). Because accuracy is dominated by the majority class, we complement it with balanced accuracy. AUC and balanced accuracy are our primary metrics, since neither depends on the class prior. As **Table 3** summarizes, the Log-Enriched CatBoost model performs best, with an AUC of 0.843, a balanced accuracy of 0.752, a recall of 0.842 and a precision of 0.350, corresponding to an MCC of 0.390. Adding log-derived features raises the balanced accuracy of CatBoost from 0.605 to 0.752 and its AUC from 0.646 to 0.843. XGBoost improves its AUC only from 0.642 to 0.672, and the Log-Enriched XGBoost model has a lower recall (0.594) than the Base XGBoost model (0.752).

4.3 Statistical Significance Analysis

To assess whether the performance differences reported in Table 3 exceed sampling variability, we analyzed the held-out predictions of all models on a common test set of 102,835 executions (18,294 failures, 17.8 %). All models (both Base and Log-Enriched) were evaluated under the same chronological split and therefore share an identical test set. Because workflow runs from the same repository are correlated, uncertainty was quantified with 95 % percentile confidence intervals obtained from a 1,000-sample bootstrap over repositories. The best-performing model (CatBoost, Log-Enriched Feature Set) served as the reference and was compared with every other model. Differences in AUC and F1 were summarized with paired repository-level bootstrap confidence intervals; a difference was considered significant when its interval excluded zero. In addition, AUCs were compared with DeLong's test for correlated ROC curves. DeLong's test treats executions as independent and may therefore be anticonservative for clustered data, so it is reported as a supplementary indicator. All p-values were adjusted for multiple comparisons with the Holm procedure ($\alpha = 0.05$).

Table 5 compares the reference model with the other seven models. The AUC advantage of CatBoost ranges from 0.081 (*Random Forest, Log-Enriched*) to 0.240 (*Random Forest, Base*), and its F1 advantage from 0.067 to 0.170. Even the smallest AUC difference is roughly seven times larger than the half-width of the reference model's own confidence interval (±0.012). These differences are statistically significant: the paired bootstrap intervals exclude zero and the Holm-adjusted DeLong p-values are below 0.001 in every comparison.

Among the Base Feature Set models, AUCs lie between 0.603 and 0.646. Random Forest is the weakest, and its interval does not overlap with that of any other Base model; the intervals of the remaining four models (AUC 0.630–0.646) overlap at least partly, so these four cannot be reliably ranked against one another (see Table 5 and Table 6). Adding log-derived features raises the AUC of CatBoost from 0.646 to 0.843 ($\Delta = +0.197$) and its F1-score from 0.352 to 0.495 ($\Delta = +0.143$).

Table 4. Point estimates and 95 % repository-level bootstrap confidence intervals (positive class = failed execution). All models share the same chronological test set.

Model	Feature set	AUC (95% CI)	PR-AUC (95% CI)	Balanced acc. (95% CI)	F1 (95% CI)

CatBoost	*Log-Enriched_Feature_Set*	0.843 (0.831–0.855)	0.512 (0.489–0.534)	0.752 (0.743–0.761)	0.495 (0.478–0.511)
Random Forest	*Log-Enriched_Feature_Set*	0.762 (0.748–0.775)	0.421 (0.402–0.439)	0.688 (0.679–0.694)	0.428 (0.421–0.435)
XGBoost	*Log-Enriched_Feature_Set*	0.672 (0.658–0.681)	0.318 (0.301–0.334)	0.623 (0.615–0.636)	0.371 (0.365–0.376)
CatBoost	*Base_Feature_Set*	0.646 (0.637–0.651)	0.289 (0.276–0.301)	0.605 (0.597–0.612)	0.352 (0.349–0.355)
XGBoost	*Base_Feature_Set*	0.642 (0.635–0.647)	0.281 (0.269–0.293)	0.594 (0.587–0.601)	0.344 (0.341–0.347)
Logistic Regression	*Base_Feature_Set*	0.631 (0.626–0.638)	0.268 (0.257–0.279)	0.589 (0.581–0.594)	0.340 (0.332–0.344)
HistGradientBoosting	*Base_Feature_Set*	0.630 (0.626–0.638)	0.271 (0.260–0.282)	0.592 (0.588–0.598)	0.341 (0.338–0.347)
Random Forest	*Base_Feature_Set*	0.603 (0.598–0.614)	0.241 (0.230–0.253)	0.567 (0.561–0.574)	0.325 (0.321–0.334)

Table 5. Pairwise comparison of CatBoost (Log-Enriched) with the other models (Δ = reference − comparator).

Comparator	**Δ AUC (95% CI)**	**Δ F1 (95% CI)**	**Δ PR-AUC (95% CI)**	**DeLong p (Holm)**
Random Forest (Log-Enriched)	0.081 (0.062–0.099)	0.067 (0.048–0.085)	0.091 (0.068–0.113)	< 0.001
XGBoost (Log-Enriched)	0.171 (0.152–0.189)	0.124 (0.105–0.142)	0.194 (0.171–0.216)	< 0.001
CatBoost (Base)	0.197 (0.179–0.214)	0.143 (0.125–0.160)	0.223 (0.201–0.244)	< 0.001
XGBoost (Base)	0.201 (0.183–0.218)	0.151 (0.133–0.168)	0.231 (0.209–0.252)	< 0.001
Logistic Regression (Base)	0.212 (0.194–0.229)	0.155 (0.137–0.172)	0.244 (0.222–0.265)	< 0.001
HistGradientBoosting (Base)	0.213 (0.195–0.230)	0.154 (0.136–0.171)	0.241 (0.219–0.262)	< 0.001
Random Forest (Base)	0.240 (0.222–0.257)	0.170 (0.152–0.187)	0.271 (0.249–0.292)	< 0.001

Table 6. Paired differences among Base models (reference = CatBoost Base).

Comparison	**Δ AUC (95% CI)**	**Δ F1 (95% CI)**	**Δ PR-AUC (95% CI)**
CatBoost vs XGBoost	0.004 (−0.003–0.011)	0.008 (−0.003–0.018)	0.008 (−0.005–0.020)
CatBoost vs Logistic Regression	0.015 (0.007–0.023)	0.012 (0.001–0.022)	0.021 (0.008–0.033)
CatBoost vs HistGradientBoosting	0.016 (0.008–0.024)	0.011 (0.000–0.021)	0.018 (0.005–0.030)
CatBoost vs Random Forest	0.043 (0.032–0.053)	0.027 (0.015–0.038)	0.048 (0.034–0.061)

Although the reference model achieves a high AUC (0.843) and balanced accuracy (0.752), its F1-score of 0.495 for the failure class remains modest relative to the 17.8% failure prevalence. In practical terms this corresponds to detecting the large majority of failures while generating a substantial number of false alarms (precision roughly twice the base rate), so that approximately two of every three alerts are incorrect. The model is therefore best suited as a triage or prioritization aid rather than as an automated gate.

5. DISCUSSION

The experimental results suggest that the largest performance gain comes from adding information extracted from execution logs rather than from the choice of algorithm among the Base_Feature_Set models, whose AUCs differ by at most 0.043. With

log features, however, the algorithm and its configuration matter considerably (AUC 0.672 for XGBoost, 0.762 for Random Forest and 0.843 for CatBoost), which may reflect differences in tuning or in the handling of categorical features and should be investigated further. On the *Base_Feature_Set*, all models achieved only moderate power. Once log-derived features were added, the final CatBoost model reached an AUC of 0.843. This improvement indicates that execution logs contain signals that historical and interaction features alone do not capture. Since these signals are post-execution, they support failure detection and triage of completed or running workflows rather than pre-execution forecasting.

From a practical perspective, the proposed approach can support DevOps teams in several ways:

1. ranking workflows by predicted risk so that high-risk pipelines receive earlier human attention,
2. improving resource allocation by identifying runs that are more likely to fail,
3. reducing developer waiting time through earlier feedback, provided that log-based detection can be applied to the partial logs of running workflows (not evaluated here); the pre-execution structured predictions alone are only moderately discriminative (AUC $\leq$ 0.646).

Several limitations remain. First, although both feature sets were evaluated under the same chronological split, the log-derived features are available only after (or during) execution; therefore, the Log-Enriched models perform post-execution failure detection rather than pre-execution prediction. An evaluation on partial logs (first N minutes or steps) for a genuine early-prediction setting is still required. Second, a single train/test split was used, without repeated or rolling-origin evaluation, so temporal drift and seed variability are not quantified. Third, log features were extracted using regular expressions; more advanced techniques such as LogBERT or other pretrained language models could capture richer semantic patterns. Fourth, the present study focuses only on binary outcome prediction. Extending the framework to multi-class failure categorization or to regression models that predict execution time would further increase its utility. Fifth, while we compared against several strong baselines, evaluation on additional industrial datasets would strengthen claims of generalizability. Finally, the individual contribution of each feature-engineering stage (cumulative statistics, Bayesian smoothing, Last-K, and interaction features) and a comparison with a simple baseline such as the previous run's outcome were not evaluated and remain for future work.

Future work will therefore explore (i) pretrained log language models for feature extraction, (ii) joint prediction of outcome and execution time, (iii) online or incremental learning settings that continuously update models as new workflow runs arrive, and (iv) deployment studies that measure the real-world impact on developer productivity and pipeline reliability. In addition, a controlled ablation (identical split and model with and without log features) and an early-prediction study on partial logs are planned.

The experimental results provide clear answers to the two research questions.

- RQ1 examined the impact of incremental, time-aware feature engineering. Models trained on the Base_Feature_Set achieved moderate performance, with AUC values ranging from 0.603 to 0.646 and accuracies between 0.462 and 0.600. These results show that structured, leakage-free features carry some predictive signal (AUC above the 0.5 chance level) but that their discriminative power remains limited when used in isolation. The individual contribution of each feature-engineering stage (cumulative statistics, smoothing, Last-K, interaction) and a comparison with simple baselines such as the previous run's outcome were not evaluated here and are needed to answer RQ1 fully.
- RQ2 investigated the contribution of textual log features. Adding log-derived signals was associated with a substantial improvement: the final CatBoost model reached an AUC of 0.843 under the same chronological split used for the Base_Feature_Set. This corresponds to an absolute gain of approximately 0.20 in AUC over the best model using only structured features. The gains of CatBoost across metrics indicate that execution logs contain complementary information that enhances the model's ability to distinguish successful from failed workflow runs. Because the log features are available only after execution, this gain reflects the benefit of log-derived signals in a post-execution failure-detection setting rather than a pure pre-execution improvement.

6. CONCLUSION

This paper presented a time-aware approach for predicting the outcomes of CI/CD workflow executions on GitHub Actions, evaluated on the large-scale GHALogs dataset. The proposed method relies on a strictly chronological feature-engineering pipeline that constructs two complementary feature sets: *Base_Feature_Set*, which captures historical failure patterns, short-term trends, and interaction effects, and *Log-Enriched_Feature_Set*, which additionally incorporates textual signals extracted from execution logs. Experiments comparing multiple models show that structured features alone yield moderate predictive performance. When log-derived features are incorporated, the final CatBoost model achieves an AUC of 0.843 under the same chronological split, higher than all other evaluated models, including Random Forest and XGBoost on the same features (AUC 0.762 and 0.672) and Logistic Regression and HistGradientBoosting on the structured features. These results suggest that execution logs provide complementary information; since log features are post-execution, they are best suited to failure detection and triage, whereas pre-execution prediction from historical features remains moderate.

7. REFERENCES

[1] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industry software development," Journal of Systems and Software, vol. 87, pp. 48-59, 2014.

[2] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," IEEE access, vol. 5, pp. 3909-3943, 2017.

[3] E. Soares, G. Sizilio, J. Santos, D. A. Da Costa, and U. Kulesza, "The effects of continuous integration on software development: a systematic literature review," Empirical Software Engineering, vol. 27, no. 3, p. 78, 2022.
[4] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey, "Uncovering the benefits and challenges of continuous integration practices," IEEE Transactions on Software Engineering, vol. 48, no. 7, pp. 2570-2583, 2021.
[5] S. Sistla, "The Evolution of Continuous Integration - Continuous Delivery," International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, vol. 9, 07/06 2021.
[6] R. Faqih, A. Taufiqurrahman, H. Husen, and K. Sabariah, "Empirical analysis of CI/CD tools usage in GitHub Actions workflows," Journal of Informatics and Web Engineering, vol. 3, no. 2, pp. 251-261, 2024.
[7] Hong, Yang, Chakkrit Tantithamthavorn, Jirat Pasuksmit, Patanamon Thongtanunam, Arik Friedman, Xing Zhao, and Anton Krasikov. "Practitioners' challenges and perceptions of ci build failure predictions at atlassian." In Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, pp. 370-381. 2024.
[8] Ge, Wenhao, and Chen Zhang. "Understanding and Detecting Flaky Builds in GitHub Actions." arXiv preprint arXiv:2602.02307 (2026).
[9] P. R. Mazrae, A. Decan, T. Mens, and M. Wessel, "An empirical study of the evolution of GitHub Actions workflows," Journal of Systems and Software, p. 112824, 2026.
[10] Barboni, Morena, Antonia Bertolino, and Guglielmo De Angelis. "Flakiness goes live: Insights from an in vivo testing simulation study." Information and Software Technology 167 (2024): 107373.
[11] A. Mishra and Z. Otaiwi, "DevOps and software quality: A systematic mapping," Computer Science Review, vol. 38, p. 100308, 2020/11/01/ 2020, doi: https://doi.org/10.1016/j.cosrev.2020.100308.
[12] A. S. Mohammed, V. R. Saddi, S. K. Gopal, S. Dhanasekaran, and M. S. Naruka, "AI-Driven Continuous Integration and Continuous Deployment in Software Engineering," in 2024 2nd International Conference on Disruptive Technologies (ICDT), 15-16 March 2024 2024, pp. 531-536, doi: 10.1109/ICDT61202.2024.10489475.
[13] G. Sriraman and S. R, "A machine learning approach to predict DevOps readiness and adaptation in a heterogeneous IT environment," Frontiers in Computer Science, Original Research vol. Volume 5 - 2023, 2023. [Online]. Available: https://www.frontiersin.org/journals/computer-science/articles/10.3389/fcomp.2023.1214722.
[14] Li, Jiatai, Yang Zhang, Tao Wang, and Yiwen Wu. "GHA-BFP: Framework for Automated Build Failure Prediction in GitHub Actions." In 2024 31st Asia-Pacific Software Engineering Conference (APSEC), pp. 472-476. IEEE, 2024.
[15] F. Moriconi, T. Durieux, J. R. Falleri, R. Troncy, and A. Francillon, "GHALogs: Large-Scale Dataset of GitHub Actions Runs," in 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR), 28-29 April 2025 2025, pp. 669-673, doi: 10.1109/MSR66628.2025.00104.
[16] Al-Baltah, Ibrahim Ahmed, Nagi Al-Shaibany, Majdi Abdellatief, Mohammed M. Al-Gawda, and Sultan Yahya Al-Sultan. "An Intelligent Multi-Class XGBoost-Based Model for Optimizing DevOps Continuous Integration and Continuous Deployment Failure Prediction." Information 17, no. 2 (2026): 178.
[17] Enemosah, Aliyu. "Enhancing DevOps efficiency through AI-driven predictive models for continuous integration and deployment pipelines." International Journal of Research Publication and Reviews 6, no. 1 (2025): 871-887.
[18] B. C. Vadde and V. B. Munagandla, "Integrating AI-Driven Continuous Testing in DevOps for Enhanced Software Quality," Revista de Inteligencia Artificial en Medicina, vol. 14, no. 1, pp. 505-513, 2023.
[19] H. V. Pandhare, "Future of Software Test Automation Using AI/ML," International Journal of Engineering And Computer Science, vol. 13, no. 05, 2025.
[20] A. Mishra and A. Sharma, "Deep learning based continuous integration and continuous delivery software defect prediction with effective optimization strategy," Knowledge-Based Systems, vol. 296, p. 111835, 2024.
[21] Beller, Moritz, Georgios Gousios, and Andy Zaidman. "Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration." In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 447-450. IEEE, 2017.
[22] Shah, Syed Muhammad Ashhar, Sehrish Habib, Muizz Ahmed Hussain, Maryam Abdul Ghafoor, and Abdul Ali Bangash. "Reliability of AI Bots Footprints in GitHub Actions CI/CD Workflows." In Proceedings of the 23rd International Conference on Mining Software Repositories, pp. 757-761. 2026.



Alireza Keshavarzi**,** CODE ORCID: **0009-0009-7257-6379**
Department of Computer Engineering, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran, alkeshavarzi@buqaen.ac.ir



Morteza Noferesti, CODE ORCID: 0009-0000-5507-1461
Department of Computer Engineering, Faculty of Engineering, Bozorgmehr University of Qaenat, Qaen, Iran, mnoferesti@buqaen.ac.ir