

An Intelligent Machine Learning Framework for Early Detection of CI/CD Pipeline Failures

Sana Fatima *, Muazzam Khan and Shayan Faiz

Department of Software Engineering, NED University of Engineering and Technology, Karachi, 75270, Pakistan

* Corresponding Author: Sana Fatima (Email: sanafatima@cloud.neduet.edu.pk)

Received: 17/08/2026, Revised: 09/10/2026, Accepted: 12/10/2026

Abstract—Continuous integration and continuous delivery (CI/CD) pipelines automate software build, testing, and deployment workflows, but failures can delay releases and consume engineering resources. This study investigates early failure detection using an adaptive ensemble of XGBoost, bidirectional long short-term memory (Bi-LSTM), and a BERT encoder, combined through a gating multilayer perceptron. The evaluation covers 194,400 pipeline executions from Jenkins, GitHub Actions, GitLab CI/CD, Azure DevOps, and CircleCI. A candidate space of 167 features captures build metrics, log semantics, resource utilization, test outcomes, code changes, and dependencies. Nine models are compared using precision, recall, F1-score, AUC-ROC, and training time. The ensemble achieves an F1-score of 0.940 and an AUC-ROC of 0.974, compared with 0.924 and 0.961 for BERT. At checkpoints corresponding to 10%, 25%, and 50% of estimated pipeline duration, failure-detection rates are 54.2%, 78.6%, and 91.3%, with mean warning lead times of 14.1, 8.7, and 4.3 minutes, respectively. The results indicate a trade-off between earlier warnings and detection coverage. Build-duration deviation, error-keyword frequency, and CPU spikes receive the highest feature-importance scores. Interpretation is limited by unequal model feature sets, the absence of leave-one-platform-out results, and incomplete end-to-end training-cost accounting. Future evaluation should examine threshold calibration, temporal drift, and false-alarm costs before operational deployment.

Index Terms—CI/CD pipelines, DevOps, ensemble methods, failure prediction, log analysis, machine learning

I. INTRODUCTION

Continuous integration and continuous delivery (CI/CD) pipelines automate software build, testing, and deployment activities. A systematic review by Shahin et al. describes continuous practices, their supporting tools, and implementation challenges [1]. Research on software delivery also links development and operational practices to organizational performance [2]. Pipeline failures can interrupt these workflows and require developers to investigate build output, test results, dependencies, and infrastructure conditions. Flaky tests are a documented source of inconsistent test outcomes [3].

Early failure detection aims to provide a warning while a pipeline is still running. Potential inputs include build metadata, resource measurements, code changes, dependency information, and log messages. These signals differ in structure and become available at different stages of execution. Tree-based models such as XGBoost support structured predictive inputs [4], while log-analysis methods such as LogRobust and LogBERT examine sequential or contextual information in system logs [5], [6]. Combining these information sources offers a way to examine failure risk throughout execution.

This study investigates an adaptive ensemble combining XGBoost, Bi-LSTM, and a BERT encoder through a gating multilayer perceptron. Its objective is to combine complementary telemetry and log information for checkpoint-based failure prediction. The evaluation considers classification performance, warning lead time, and the relative importance of monitoring signals. Feature availability, class imbalance, platform transfer, and false-alarm costs are central to interpreting the results.

The paper makes four contributions:

- 1) A synthesis of related approaches to log analysis, operational anomaly detection, and continuous integration, identifying their relevance to early pipeline failure detection.
- 2) An analysis of 194,400 pipeline execution records across five CI/CD platforms, using a candidate space of 167 features.
- 3) A comparison of nine models using classification metrics and training times, together with an assessment of detection coverage and warning lead time at three execution checkpoints.
- 4) An adaptive ensemble architecture that combines structured, sequential, and contextual representations for failure prediction.

Section II discusses related literature. Section III describes the dataset, feature engineering, model architectures, and evaluation protocol. Section IV presents the results. Section V discusses limitations, and Section VI concludes the paper.

II. LITERATURE REVIEW

A. Continuous Integration Data and Build Behavior

TravisTorrent links Travis CI build information with GitHub project data to support empirical research on continuous integration [7]. Beller et al. also examine testing and build behavior in Travis CI projects [8]. These studies provide relevant context for analyzing build logs and outcomes. Their platform coverage differs from the five-platform evaluation in this study, and their results should not be interpreted as evidence of transfer between CI/CD systems.

B. Log Analysis and Failure-Related Signals

Log parsing converts free-text messages into structured events suitable for subsequent analysis [9]. LogRobust addresses anomaly detection when log data change, using semantic representations and an attention-based Bi-LSTM [5]. LogBERT applies self-supervised BERT-based learning to log anomaly detection [6]; BERT itself was introduced as a



language-representation method [10]. LogAnomaly examines sequential and quantitative anomalies in unstructured logs [11]. These approaches motivate the use of log representations alongside structured pipeline metrics. Their datasets and prediction targets differ, however, so their performance measures cannot be treated as a common CI/CD benchmark.

Drain uses a fixed-depth tree to parse streaming log messages into templates [12]. DeepLog models normal log sequences with LSTM networks and detects deviations from learned patterns [13]. Loghub provides system-log datasets for log-analytics research [14]. These resources help distinguish parsing, anomaly modeling, and benchmark construction. They are related work, not sources of the pipeline executions evaluated here.

C. Operational Diagnosis and Continuous Integration

Related operational studies address complementary prediction and diagnosis tasks. Microsoft's incident-management study discusses challenges in service dependencies and resource-health assessment [15]. SCWarn examines bad software changes using multimodal anomaly detection [16], while Groot uses event graphs for root-cause analysis [17]. These studies illustrate the relevance of multiple operational signals, although detecting a bad change or diagnosing an incident differs from predicting the outcome of an unfinished pipeline.

Other work addresses test scheduling and verification of machine-learning updates. Spieker et al. apply reinforcement learning to test prioritization and selection in continuous integration [18]. The ease.ml/ci framework supports continuous integration of machine-learning models [19]. These tasks concern the allocation of testing effort and assessment of model updates, respectively. SMOTE provides a general approach to minority-class oversampling [20], which is relevant when failed runs are less frequent than successful runs.

D. Scope and Research Focus

This focused review covers structured prediction, log anomaly detection, incident diagnosis, and test selection. Differences in datasets, prediction targets, and evaluation protocols prevent direct comparison of reported performance. Table I summarizes the tasks addressed by these approaches and their relevance to early CI/CD pipeline failure detection.

The present study focuses on three connected questions: how complementary inputs relate to failure discrimination, how detection coverage changes at earlier execution checkpoints, and which monitored signals receive the highest importance scores. Generalization to an unseen platform remains a separate question from evaluation on a pooled multi-platform dataset.

III. METHODOLOGY

A. Research Design and Dataset

The workflow comprises data collection and preprocessing, feature extraction and selection, model training, and evaluation. Pipeline execution logs cover January 2021–December 2024 and originate from APIs and webhook integrations for five platforms: enterprise Jenkins clusters, open-source GitHub Actions repositories, organizational GitLab CI/CD namespaces,

commercial Azure DevOps projects, and containerized CircleCI projects. Table II summarizes the dataset.

Records are deduplicated using pipeline run identifiers and content hashes. Records with more than 15% missing feature values are excluded, remaining missing values are handled using k-nearest-neighbor imputation, and timestamps are normalized to UTC. The dataset contains 194,400 executions. The rounded platform-specific rates imply an overall failure rate of approximately 21.9%. SMOTE [20] is used for class-imbalance handling during model fitting.

B. Feature Engineering

The candidate feature space comprises six categories totaling 167 features:

- 1) Build metrics (31): build and stage durations, deviations from historical durations, and queue time.
- 2) Log semantics (42): TF-IDF-based representations and error-keyword information. Log processing uses a vocabulary cap of 5,000 tokens, the first 500 log messages, and a set of 87 error keywords.
- 3) Resource utilization (28): CPU percentiles, memory headroom, disk-I/O measurements, and GPU memory allocation where available.
- 4) Test coverage and quality (24): test duration, failure counts, historical flakiness, and coverage changes.
- 5) Code changes (22): lines added or deleted, complexity changes, modified modules, and file churn.
- 6) Dependencies and environment (20): resolution time, version mismatches, image-download failures, and environment overrides.

The platform-specific feature counts range from 124 to 167; the 167-dimensional candidate space should therefore be distinguished from the number of directly available fields on each platform. Platform-specific availability is a limitation when comparing models across heterogeneous sources.

Feature selection applies a mutual-information threshold of 0.01, reducing 167 candidates to 143, followed by Random-Forest-based recursive feature elimination with cross-validation (RFECV) to select 98 features. Classical models use the 98 selected features, while the deep-model configurations use the 167-feature input space. Consequently, the experiment compares model configurations with different inputs rather than isolating architecture alone.

C. Model Architectures

The nine evaluated models are Logistic Regression, Decision Tree, Random Forest, XGBoost, an RBF-kernel SVM, LSTM, Bi-LSTM, BERT, and the ensemble. Random Forest combines randomized decision trees [21]. XGBoost supplies the tree-boosting component [4]. Its hyperparameter search uses 200 Bayesian-optimization iterations over the following ranges:

$$\begin{aligned} \text{max_depth} &\in [3, 10], \\ \text{learning_rate} &\in [0.01, 0.3], \\ \text{n_estimators} &\in [100, 1000], \\ \text{subsample} &\in [0.6, 1.0]. \end{aligned}$$

The deep-learning configuration incorporates numeric features and log-token sequences with a maximum length of 512.

TABLE I
RELATED APPROACHES AND THEIR SCOPE

Source	Documented focus	Boundary of relevance
XGBoost [4]	Tree boosting	Structured predictive modeling.
LogRobust [5]	Anomaly detection on changing logs	Sequential and semantic log representations.
LogBERT [6]	Self-supervised log anomaly detection	Separate datasets and task definitions.
Log parsing [9]	Structured events from log text	Preprocessing for downstream log analysis.
LogAnomaly [11]	Sequential and quantitative anomalies	Related log-analysis method.
Incident management [15]	Operational incident investigation	Service dependencies and resource-health assessment.
SCWarn [16]	Bad software-change detection	Related multimodal operational task.
Groot [17]	Event-graph root-cause analysis	Diagnosis rather than CI/CD checkpoint prediction.
Test prioritization [18]	Reinforcement learning for test selection	Scheduling rather than pipeline failure classification.
ease.ml/ci [19]	Integration of machine-learning updates	Assessment of machine-learning model updates.

TABLE II
DATASET COMPOSITION

Dataset Source	Pipeline Type	Total Samples	Failure Rate (%)	Features
Jenkins Enterprise	Build & Test	48,500	23.4	142
GitHub Actions	CI/CD Full Stack	62,300	18.7	167
GitLab CI/CD	Multi-stage	35,200	27.1	153
Azure DevOps	Release Pipelines	29,800	21.3	138
CircleCI	Containerized	18,600	19.8	124

The LSTM architecture builds on gated recurrent memory [22] and uses two layers with 256 hidden units and dropout of 0.3. The Bi-LSTM uses bidirectional processing, following the bidirectional recurrent-network approach [23]. The transformer branch uses the attention-based Transformer architecture [24] through BERT-base-uncased, with 12 layers and a hidden dimension of 768, followed by a classification head [10].

The ensemble combines XGBoost probability outputs, a pooled Bi-LSTM state, and the BERT classification-token embedding. Its fusion network includes an MLP with dimensions 512→128→2, GELU activations, and layer normalization. Softmax gating is conditioned on a 16-dimensional context vector representing platform and project information. Figure 1 provides an overview of the architecture.

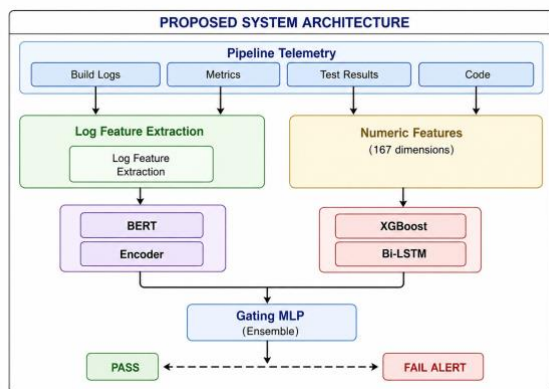


Fig. 1. Adaptive Ensemble Architecture for CI/CD Failure Detection

D. Early Detection Protocol

Predictions are evaluated at 10%, 25%, and 50% of estimated pipeline duration. An early-detection protocol requires both the duration estimate and the input features to be available

at prediction time; using eventual run duration or future test outcomes would leak information about the target. Feature availability varies across stages, particularly for test-related signals.

Failure probabilities are calibrated using isotonic regression and compared with an alert threshold. Isotonic calibration is studied by Niculescu-Mizil and Caruana [25]; Guo et al. examine calibration of modern neural networks [26]. Calibration and threshold selection serve different purposes: calibration relates model scores to failure probabilities, while the threshold determines when an alert is issued. Alert lead time is the elapsed time between the alert timestamp and the recorded failure timestamp. Evaluation considers failure-detection coverage and mean lead time at each checkpoint. A survey of 48 DevOps engineers provides practitioner feedback on warning lead times, including a two-minute actionability criterion.

E. Experimental Setup and Evaluation

Experiments use two NVIDIA A100 40GB GPUs, 256 GB RAM, and 64 AMD EPYC cores. The software environment comprises Python 3.11, PyTorch 2.2, Scikit-learn 1.4, and Hugging Face Transformers 4.39. Evaluation uses stratified five-fold cross-validation and reports precision, recall, F1-score, AUC-ROC, and training time.

Model comparisons are interpreted descriptively. Five-fold cross-validation provides limited evidence for inferential comparisons, and the folds share training observations. Accordingly, differences in the reported metrics are not treated as statistically significant improvements.

IV. RESULTS AND DISCUSSION

A. Dataset Composition

Table II summarizes 194,400 pipeline records across five platforms. GitHub Actions contributes the largest share, with

62,300 records (approximately 32.0%), and has the lowest failure rate at 18.7%. GitLab CI/CD has the highest failure rate at 27.1%. The weighted rate implied by the rounded platform percentages is approximately 21.9%. These differences describe the sampled executions and do not establish a causal effect of platform choice.

B. Model Performance Comparison

Table III presents the performance of nine models. The ensemble achieves the highest F1-score (0.940) and AUC-ROC (0.974) among the evaluated configurations. Relative to BERT, the absolute differences are 0.016 in F1-score and 0.013 in AUC-ROC. Figure 2 summarizes AUC-ROC across all models. These differences are descriptive; statistical significance is not established.

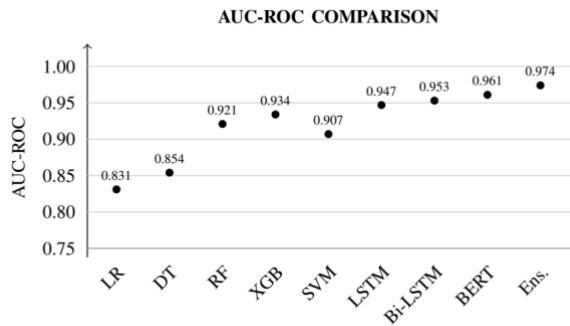


Fig. 2. AUC-ROC Comparison Across the Evaluated Models

Logistic Regression and Decision Tree have the lowest F1-scores, at 0.772 and 0.797. XGBoost achieves an F1-score of 0.887, while LSTM and Bi-LSTM achieve 0.899 and 0.911. Deep-model configurations have higher recall than the classical configurations in this evaluation. Because their input feature sets differ, the comparison does not isolate the effect of architecture. Likewise, the 0.012 F1-score difference between LSTM and Bi-LSTM cannot be attributed solely to bidirectional context without a controlled ablation.

BERT's training time is 891.4 seconds, approximately 14.0 times the 63.5 seconds recorded for XGBoost. The ensemble entry is 512.7 seconds. These entries do not establish an end-to-end efficiency advantage for the ensemble: the accounting boundaries for component training and fusion must be comparable before such an inference is justified. Computational cost is therefore treated separately from predictive performance.

C. Feature Importance Analysis

Table IV presents eight feature-importance scores. Build-duration deviation has the highest score (0.187), followed by error-keyword frequency (0.163) and CPU spike ratio (0.141). Test failure rate and cyclomatic-complexity change follow at 0.129 and 0.112. The ranking indicates that build behavior, log patterns, and resource conditions are useful monitoring targets in this evaluation. Feature-importance scores describe associations within the fitted model and do not establish causal relationships. The ranking identifies candidate signals for monitoring, but determining whether a smaller feature set preserves predictive

performance requires controlled feature-ablation experiments on held-out executions.

D. Early Detection Performance

Figure 3 shows failure-detection rates of 54.2%, 78.6%, and 91.3% at the 10%, 25%, and 50% checkpoints, respectively. The corresponding mean lead times are 14.1, 8.7, and 4.3 minutes. Earlier checkpoints offer longer warning times, whereas later checkpoints detect a larger proportion of failures. The 4.3-minute mean therefore applies specifically to the 50% checkpoint.

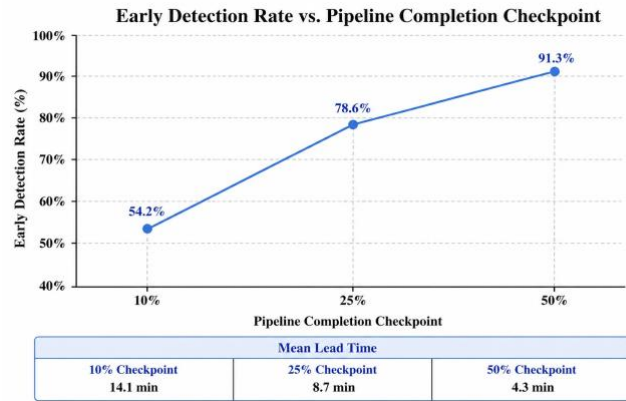


Fig. 3. Failure-Detection Rates and Mean Lead Times at Execution Checkpoints

Lead-time standard deviations are 2.4 minutes at the 25% checkpoint and 1.8 minutes at the 50% checkpoint. Detection rate among failing runs is distinct from overall classification accuracy and does not quantify the false-alarm burden on successful runs. The checkpoint results demonstrate a coverage-lead-time trade-off, while operational usefulness also depends on alert precision, investigation time, and the cost of intervention.

V. LIMITATIONS

A. Generalization and Evaluation Design

Stratified evaluation on a pooled multi-platform dataset does not establish generalization to an unseen platform. Leave-one-platform-out evaluation is needed to assess that question. Repository-grouped and chronological holdouts would also help distinguish transferable patterns from project overlap and temporal dependence.

The comparison uses 98 selected features for classical models and 167 features for deep-model configurations. Thus, differences may reflect both representation and architecture. Equal-input comparisons and feature-set ablations are needed to separate these effects. The validity of the cross-validation results depends on separating training operations from held-out evaluation. Imputation, vocabulary construction, feature selection, resampling, hyper-parameter tuning, and calibration must be confined to their designated training or validation partitions. Leakage across these boundaries can produce optimistic performance estimates. Model-selection bias can affect performance estimates when selection and evaluation are not

TABLE III
MODEL PERFORMANCE COMPARISON

Model	Precision	Recall	F1-Score	AUC-ROC	Train Time (s)
Logistic Regression	0.781	0.764	0.772	0.831	12.4
Decision Tree	0.803	0.791	0.797	0.854	8.7
Random Forest	0.876	0.869	0.872	0.921	47.2
XGBoost	0.891	0.883	0.887	0.934	63.5
SVM (RBF Kernel)	0.858	0.841	0.849	0.907	124.8
LSTM	0.903	0.896	0.899	0.947	312.6
Bi-LSTM	0.914	0.908	0.911	0.953	478.3
Transformer (BERT)	0.927	0.921	0.924	0.961	891.4
Ensemble (Proposed)	0.943	0.937	0.940	0.974	512.7

TABLE IV
FEATURE IMPORTANCE SCORES

Feature Category	Feature Name	Importance Score	Rank
Build Metrics	Build Duration Deviation	0.187	1
Log Patterns	Error Keyword Frequency	0.163	2
Resource Usage	CPU Spike Ratio	0.141	3
Test Results	Test Failure Rate	0.129	4
Code Metrics	Cyclomatic Complexity Delta	0.112	5
Dependency	Dependency Resolution Time	0.098	6
Network	Artifact Download Failure	0.087	7
Environment	Container Resource OOM	0.083	8

appropriately separated [27]. A reproducible evaluation must fit these operations using only the appropriate training partitions and leave held-out observations unaffected by resampling.

B. Threshold Calibration and False Alarms

Calibration and alert-threshold selection require data separated from final evaluation. Calibration may vary across checkpoints because feature availability and score distributions change during execution. Isotonic regression can be unstable with small calibration samples. Checkpoint-specific calibration error, precision–recall behavior, false-positive rates, and alerts per run would provide a fuller assessment than detection coverage alone. Investigation effort, unnecessary job interruption, and incorrect rollback must be weighed against missed-failure costs. The present evaluation does not quantify these operational costs. Precision–recall analysis is particularly relevant when class frequencies are unequal [28].

C. Concept Drift and Feature Availability

Changes in repositories, tools, dependencies, infrastructure, and failure prevalence can affect discrimination and calibration. Concept-drift research distinguishes changes in the data-generating process and methods for adapting to them [29]. Longitudinal evaluation is needed to determine when retraining or recalibration is appropriate. Checkpoints must use duration estimates available at prediction time, and features must exclude future log entries and final test outcomes. Platform-specific feature availability and the mapping to a common input space also affect transferability. Feature availability varies across platforms and execution stages, which may limit the transferability of a common input representation.

D. Implementation Detail and Computational Cost

The training times in III do not establish the ensemble’s end-to-end computational cost relative to the baseline models. Computational efficiency therefore remains unresolved. Consequently, no ensemble training-efficiency advantage is inferred. Independent replication requires implementation details and dataset-access arrangements. Sharing enterprise logs is subject to organizational permissions and confidentiality requirements.

VI. CONCLUSION

This study investigates early CI/CD pipeline failure detection using an adaptive ensemble of XGBoost, Bi-LSTM, and BERT with context-dependent gating. Evaluation across 194,400 executions from five platforms yields an ensemble F1-score of 0.940 and an AUC-ROC of 0.974, the highest values among the nine configurations compared. These results support further investigation of complementary structured and log-based representations, while unequal feature sets prevent attributing the differences solely to architecture.

Checkpoint evaluation shows a trade-off between warning time and detection coverage. At 10% of estimated duration, the framework detects 54.2% of failures with a mean lead time of 14.1 minutes. At 50%, detection reaches 91.3% with a mean lead time of 4.3 minutes. Build-duration deviation, error-keyword frequency, and CPU spikes receive the highest importance scores and identify candidate signals for focused monitoring.

The results provide descriptive evidence of the ensemble’s predictive performance on the pooled dataset. Generalization to unseen platforms, robustness to temporal changes, and end-to-end computational efficiency remain unestablished.

Future evaluation should use platform and chronological holdouts, compare models under matched feature sets, and assess checkpoint-specific calibration and false-alarm costs to determine suitability for operational deployment.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest to report regarding the present study.

AUTHOR CONTRIBUTIONS

Conceptualization, S.F., M.K., and S.F.; methodology, S.F. and M.K.; software, S.F., and S.F.; validation, M.K.; formal analysis, S.F., M.K., and S.F.; investigation, S.F. and M.K.; resources, M.K.; data curation, S.F.; writing—original draft preparation, S.F., and M.K.; writing—review and editing, S.F.; supervision, M.K.; project administration, S.F. and M.K.

FUNDING STATEMENT

This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT

Not applicable.

INFORMED CONSENT STATEMENT

Not applicable.

DATA AVAILABILITY STATEMENT

Data is available on reasonable request.

CONSENT FOR PUBLICATION

All authors have read and approved the final version of the manuscript and consent to its publication.

DECLARATION OF GENERATIVE AI IN SCIENTIFIC WRITING

For preparing this manuscript, the authors have only made use of generative artificial intelligence systems to enhance language proficiency, grammar, and readability.

REFERENCES

- [1] M. Shahin et al., “Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices,” *IEEE Access*, vol. 5, pp. 3909–3943, 2017.
- [2] N. Forsgren et al., *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution, 2018.
- [3] Q. Luo et al., “An empirical analysis of flaky tests,” in *Proc. FSE*, 2014, pp. 643–653.
- [4] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. ACM SIGKDD*, 2016, pp. 785–794.
- [5] X. Zhang et al., “Robust log-based anomaly detection on unstable log data,” in *Proc. ESEC/FSE*, 2019, pp. 807–817.
- [6] H. Guo et al., “LogBERT: Log anomaly detection via BERT,” in *Proc. IJCNN*, 2021, pp. 1–8.
- [7] M. Beller et al., “TravisTorrent: Synthesizing Travis CI and GitHub for full-stack research on continuous integration,” *Proc. MSR*, pp. 447–450, 2017.
- [8] M. Beller et al., “Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub,” in *Proc. MSR*, 2017.
- [9] P. He et al., “Towards automated log parsing for large-scale log data analysis,” *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 931–944, 2018.
- [10] J. Devlin et al., “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.
- [11] W. Meng et al., “LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs,” in *Proc. IJCAI*, 2019, pp. 4739–4745.
- [12] P. He et al., “Drain: An online log parsing approach with fixed depth tree,” in *Proc. IEEE ICWS*, 2017.
- [13] M. Du et al., “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. ACM CCS*, 2017.
- [14] J. Zhu et al., “Loghub: A large collection of system log datasets for AI-driven log analytics,” in *Proc. IEEE ISSRE*, 2023.
- [15] Z. Chen et al., “Towards intelligent incident management: Why we need it and how we make it,” in *Proc. ESEC/FSE*, 2020, pp. 1487–1497.
- [16] N. Zhao et al., “Identifying bad software changes via multimodal anomaly detection for online service systems,” in *Proc. ESEC/FSE*, 2021, pp. 527–539.
- [17] H. Wang et al., “Groot: An event-graph-based approach for root cause analysis in industrial settings,” in *Proc. IEEE/ACM ASE*, 2021, pp. 419–429.
- [18] H. Spieker et al., “Reinforcement learning for automatic test case prioritization and selection in continuous integration,” in *Proc. ISSTA*, 2017, pp. 12–22.
- [19] C. Renggli et al., “Continuous integration of machine learning models with ease.ml/ci: Towards a rigorous yet practical treatment,” in *Proc. MLSys*, 2019.
- [20] N. V. Chawla et al., “SMOTE: Synthetic minority over-sampling technique,” *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.
- [21] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, 2001.
- [22] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [23] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [24] A. Vaswani et al., “Attention is all you need,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.
- [25] A. Niculescu-Mizil and R. Caruana, “Predicting good probabilities with supervised learning,” in *Proc. ICML*, 2005.
- [26] C. Guo et al., “On calibration of modern neural networks,” in *Proc. ICML*, vol. 70, pp. 1321–1330, 2017.
- [27] G. C. Cawley and N. L. C. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *J. Mach. Learn. Res.*, vol. 11, pp. 2079–2107, 2010.
- [28] T. Saito and M. Rehmsmeier, “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets,” *PLOS ONE*, vol. 10, no. 3, Art. no. e0118432, 2015.
- [29] J. Gama et al., “A survey on concept drift adaptation,” *ACM Computing Surveys*, vol. 46, no. 4, Art. no. 44, 2014.