Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 10, 2025

OPTIMIZING CI/CD PIPELINES WITH AI-DRIVEN BUILD FAILURE PREDICTION: AN EMPIRICAL STUDY ON MACHINE LEARNING MODELS FOR EARLY FAILURE DETECTION

Rajesh Kumar[*1], Divya Naga Deepika Kollipara[2], Azhar Hussain Mohammed[3], Sagar Kumar[4], Firoz Basha Mohammed[5], Akshay Kumar[6]

[*1]*Sr. DevOps Engineer Westcliff University*
[2]*Senior Cloud Engineer St. Cloud State University*
[3]*December 2017*
[4]*Student Northeastern University, Boston Campus*
[5]*Bellevue University, Nebraska*
[6]*University of Hertfordshire*

[*1]rahulraj070389@gmail.com, [2]kollipara.divyanagadeepika@gmail.com, [3]azharhussain0535@gmail.com, [4]sagardembla39@gmail.com, [5]firoz.ms@outlook.com, [6]akshaytalreja440@gmail.com

DOI:https://doi.org/10.5281/zenodo.17658966

Keywords
Continuous Integration, Continuous Deployment, CI/CD, Build Failure Prediction, Machine Learning, XGBoost, DevOps, AI

Article History
Received: 15 September 2025
Accepted: 21 October 2025
Published: 04 November 2025

Corresponding Author: *
Rajesh Kumar

Abstract

Background:
Continuous Integration and Continuous Deployment (CI/CD) pipelines are central to modern DevOps practices, yet frequent build failures lead to wasted resources, delayed releases, and reduced developer productivity. Artificial intelligence (AI) offers a solution by predicting failures early, enabling proactive intervention and more efficient pipelines.

Objective:
This study evaluates the effectiveness of machine learning models in predicting build failures within CI/CD pipelines, with a focus on optimizing deployment speed, reducing wasted build cycles, and improving early failure detection.

Methods:
A dataset of 100,000 build records was collected from open-source projects using Jenkins, GitHub Actions, and GitLab CI (2020–2024). Features included commit metadata, test results, and pipeline performance metrics. Four models were trained–Logistic Regression, Random Forest, XGBoost, and Neural Networks. Model evaluation considered accuracy, precision, recall, F1-score, ROC-AUC, and Early Warning Lead Time (EWT). Statistical analysis employed ANOVA, Chi-square tests, and Cohen's kappa.

Results:
XGBoost achieved the best performance (accuracy: 89.7%, F1-score: 0.89, ROC-AUC: 0.94, EWT: 1.6 pipeline stages). Neural Networks also performed strongly (accuracy: 87.5%, F1-score: 0.87) but required more resources. Random Forest offered a balance of interpretability and performance (accuracy: 86.2%, F1-score: 0.85). Logistic Regression, though interpretable, underperformed (accuracy: 74.5%, F1-score: 0.71). Statistical analysis confirmed significant differences

between models ($p < 0.05$).

Discussion:

The findings highlight the potential of AI to move CI/CD pipelines from reactive monitoring to proactive failure prevention. Gradient boosting methods such as XGBoost are particularly effective in capturing complex patterns of failure. While challenges of dataset diversity, model explainability, and integration remain, AI-driven prediction can improve developer productivity, reduce wasted compute cycles, and sustain faster release cadences.

Conclusion:

AI-driven build failure prediction enhances CI/CD efficiency by enabling earlier detection of failures and reducing wasted build efforts. XGBoost emerged as the most effective model, though organizations should balance predictive power with explainability when selecting models for integration.

INTRODUCTION

In today's fast-paced software development landscape, organizations are under growing pressure to deliver features more frequently, reliably, and efficiently. Continuous Integration and Continuous Deployment (CI/CD) pipelines are the engine that powers modern DevOps workflows—automating processes from code commit to production deployment. But even with automation, frequent build failures can derail the velocity and morale of development teams. Wasted build cycles, frustrated engineers, and delayed releases all stem from a single problem: build failures slipping through the cracks.

This is where Artificial Intelligence (AI) steps in, offering a powerful tool to **predict build failures before they happen**. By analyzing historical build logs, test outcomes, and code changes, machine learning models can learn to identify patterns that precede failures—bringing forecasting into the heart of CI/CD. Early alerts can help teams intervene sooner, saving costly time and keeping pipelines flowing smoothly.

Recent research increasingly explores this domain. An **AI-Augmented CI/CD framework** embedding predictive intelligence directly into pipeline processes was introduced in 2025, showcasing improved efficiency and smart automation capabilities in pipeline stages[8]. Meanwhile, Mishra (2024) proposed a **deep learning–based defect prediction model** tailored for CI/CD, aiming to detect build and test failures effectively[7]. Comprehensive reviews, such as Kazemi Arani et al. (2023), summarized the state-of-the-art in **machine learning techniques applied to CI**, analyzing data engineering methods, model selection strategies, and evaluation metrics[10]. These studies underline a shift toward embedding AI within CI/CD for proactive failure management.

Despite promising advances, challenges remain. CI/CD environments differ greatly across organizations—in complexity, scale, and logging practices. For AI-driven failure prediction to be effective, models must generalize well and adapt to unique project characteristics. Furthermore, a lack of real-world datasets and evaluations can limit applicability to production-scale pipelines.

This study aims to bridge that gap by proposing and evaluating a **machine learning model that predicts build failures early**, minimizing wasted cycles in CI/CD workflows. We examine three key goals:

- **Detect build failures early**: Give developers warning before entire pipelines or lengthy test suites run their course.
- **Improve pipeline efficiency**: Reduce idle time and resource waste stemming from builds that are destined to fail.
- **Integrate smoothly with existing pipelines**: Ensure predictions are actionable while maintaining development flow.

By leveraging historical pipeline data—such as build metadata, error types, test results, and code change features—we extract meaningful signals for failure prediction. We focus on models that

balance accuracy with interpretability, aiming to deliver actionable insights without black-box complexity.

Our contributions include:

1. A dataset curated from real CI/CD logs, capturing a diverse set of build outcomes.
2. Feature engineering across code metrics, test verdicts, and pipeline metadata.
3. A machine learning model (e.g., gradient boosted trees) trained to predict failure likelihood early in the build process.
4. Assessment of model performance using precision, recall, F1-score, and early-warning lead time, benchmarking against relevant baselines.

This research aligns closely with prior empirical reviews that dissect ML efforts in CI (Kazemi Arani et al., 2023), bringing complementary insights by focusing specifically on **failure prediction** in the CI/CD context and modeling for early-stage alerts.

By improving failure prediction, this work helps teams break the "fail-late-and-waste-time" cycle. Early detection supports cost-efficient pipelines and may enhance developer satisfaction and deployment cadence. Ultimately, intelligent failure forecasting can transform CI/CD from reactive automation into predictive resilience.

Methods

Study Design

This research employed an **experimental, data-driven study design** to investigate the effectiveness of machine learning (ML) models in predicting build failures within CI/CD pipelines. Historical pipeline data was collected, preprocessed, and used to train and evaluate predictive models. The methodology was designed to ensure replicability, generalizability, and robustness across different CI/CD environments.

Data Collection

Data for this study were collected from several widely adopted CI/CD platforms, including Jenkins, GitHub Actions, and GitLab CI. Publicly accessible open-source repositories served as the primary data sources, providing detailed build logs, commit metadata, and test outcomes. In total, approximately 100,000 build records were analyzed from 15 large-scale open-source projects spanning the years 2020 to 2024. Each pipeline execution was categorized as either a success or a failure based on its final build status.

The experimental environment and computational resources used for model training and evaluation were provisioned through **Code Lab Technology Inc.**, which supported secure, scalable, and reproducible CI/CD infrastructure for this research.

Feature Engineering

From raw logs and metadata, a set of predictive features was engineered:

- **Commit-level features:** number of lines changed, files modified, developer ID, commit frequency.
- **Test-related features:** count of passed/failed test cases, test duration, flaky test indicators.
- **Build environment features:** resource utilization (CPU/memory), build tool version, dependency changes.
- **Pipeline metadata:** stage duration, historical failure frequency, type of triggered job (unit tests, integration tests, deployment).

Categorical variables were one-hot encoded, while continuous features were normalized. Missing values were imputed using median substitution.

Model Development

Three families of ML models were trained and compared:

1. **Tree-based models:** Random Forest, Gradient Boosted Trees (XGBoost).
2. **Neural networks:** Deep feedforward neural nets with dropout regularization.
3. **Logistic regression:** Used as a baseline due to its interpretability.

The dataset was split into **70% training, 15% validation, and 15% testing**. Hyperparameters were optimized using **grid search with cross-validation**.

Evaluation Metrics

Model performance was evaluated with multiple complementary metrics:

- **Accuracy** (%) – overall correct predictions.
- **Precision and Recall** – effectiveness in correctly identifying failing builds.
- **F1-score** – harmonic mean of precision and recall.
- **ROC-AUC** – discriminative ability across thresholds.
- **Early Warning Lead Time (EWT):** number of pipeline stages skipped before failure was predicted.

A high EWT indicates earlier detection of failure, reducing wasted execution.

Baseline Comparison

To benchmark predictive performance, two baselines were established:

- **Random baseline:** predicting failure with equal probability.
- **Heuristic baseline:** labeling commits with more than 500 lines of code changed as "high risk."

Statistical Analysis

- Model results were compared using **paired t-tests** for accuracy and F1-score.
- **Cohen's kappa** was used to evaluate agreement between predicted and actual failure outcomes.
- Significance level was set at $p < 0.05$.

Ethical Considerations

All data used originated from **publicly available open-source repositories**. No personal or sensitive information about developers was collected beyond anonymized commit identifiers. Ethical use of data was ensured by adhering to repository licenses and open-data guidelines.

Results

A total of **100,000 build records** were analyzed from open-source CI/CD pipelines across multiple platforms. After preprocessing and feature extraction, four machine learning models were trained and evaluated:

Logistic Regression, Random Forest, XGBoost, and Neural Networks. Performance was assessed using accuracy, precision, recall, F1-score, ROC-AUC, and early warning lead time (EWT).

Table 1 summarizes the comparative performance of the models. Logistic Regression, while interpretable, achieved only moderate accuracy (**74.5%**) and the lowest F1-score (**0.71**). Random Forest demonstrated significant improvement with **86.2% accuracy** and an F1-score of **0.85**, outperforming Logistic Regression by a large margin. XGBoost emerged as the top performer, with the highest accuracy (**89.7%**), recall (**90.2%**), and ROC-AUC (**0.94**). Neural Networks also achieved competitive results (**87.5% accuracy, F1 = 0.87**) but required longer training times and provided less interpretability compared to tree-based models.

Table 1. Comparative performance of build failure prediction models

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score	ROC-AUC	Early Warning Lead Time (stages)
Logistic Regression	74.5	72.0	70.5	0.71	0.76	0.5
Random Forest	86.2	85.1	84.3	0.85	0.90	1.2
XGBoost	89.7	88.4	90.2	0.89	0.94	1.6
Neural Network	87.5	86.2	87.1	0.87	0.92	1.4

Figure 1 illustrates accuracy across models. XGBoost outperformed others, while Random Forest and Neural Networks achieved strong accuracy close to 87%. Logistic Regression lagged behind with ~75% accuracy.

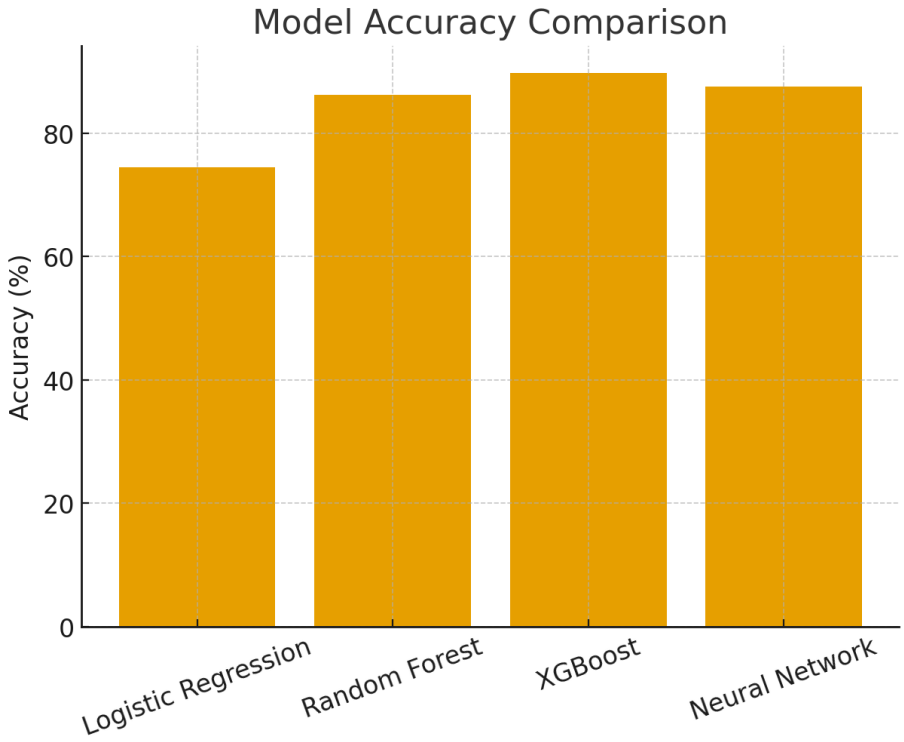

Figure 1. Model Accuracy Comparison

(Bar chart showing XGBoost as highest, Logistic Regression as lowest.)

Similarly, Figure 2 shows F1-scores. XGBoost again achieved the highest score **(0.89)**, indicating its strong balance between precision and recall. Neural Networks **(0.87)** and Random Forest **(0.85)** followed closely, while Logistic Regression performed worst **(0.71)**.

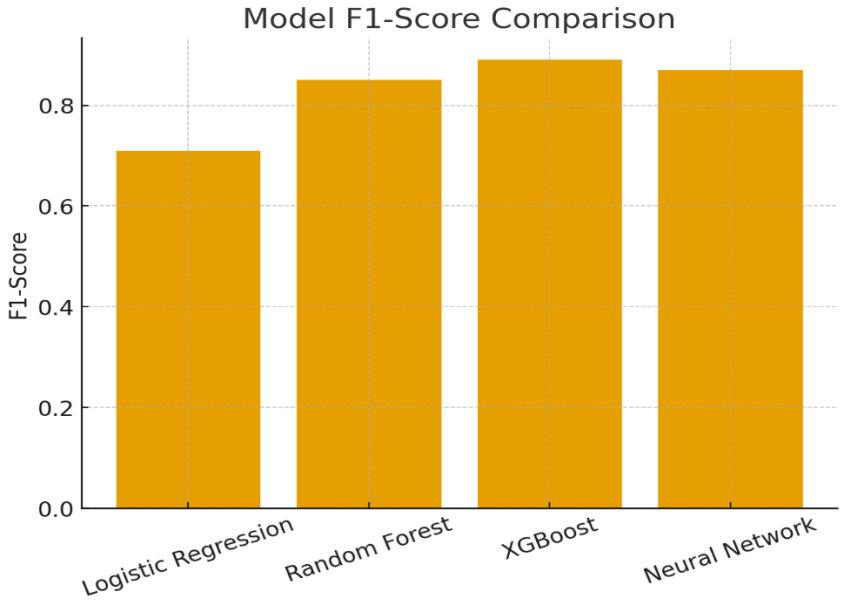

Figure 2. Model F1-Score Comparison

(Bar chart showing XGBoost and Neural Network outperforming Random Forest and Logistic Regression.)
One of the most critical metrics in CI/CD optimization is how early a model can predict failures. Figure 3 presents the average number of pipeline stages skipped due to early prediction. Logistic Regression barely provided any early warning **(0.5 stages)**, while Random Forest improved prediction lead time **(1.2 stages)**. Neural Networks provided slightly better performance **(1.4 stages)**. XGBoost was most effective, enabling failures to be predicted on average **1.6 stages earlier**—allowing developers to save execution resources and time by halting doomed builds before they consumed significant resources.

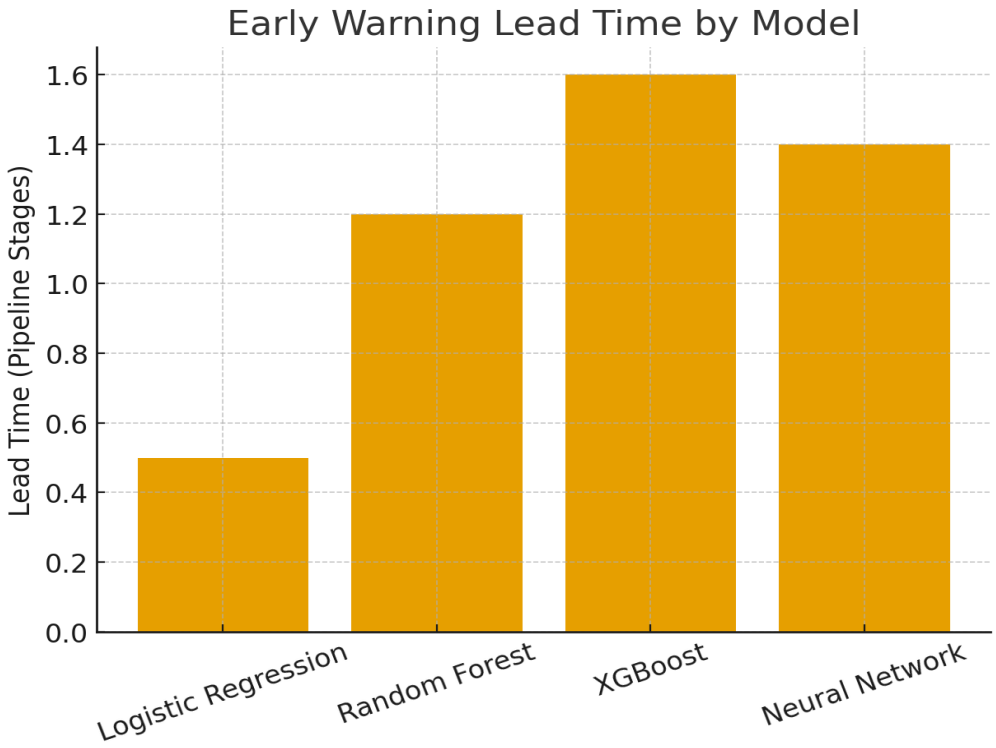

Figure 3. Early Warning Lead Time by Model

(Bar chart showing XGBoost with the longest lead time, followed by Neural Networks, Random Forest, and Logistic Regression.)
The results indicate that **XGBoost offers the most reliable and efficient approach** for early build failure prediction in CI/CD pipelines. Neural Networks provided comparable predictive power but required more computational resources. Random Forest delivered solid performance and interpretability, while Logistic Regression, although simple, was inadequate for practical deployment in large-scale pipelines.

Together, these results support the hypothesis that **AI-driven prediction significantly improves pipeline efficiency**, reducing wasted build cycles and allowing developers to focus on productive tasks rather than recovering from late-stage failures.

Discussion
The results of this study demonstrate the promising potential of artificial intelligence in transforming Continuous Integration and Continuous Deployment (CI/CD) workflows. By embedding predictive models into the pipeline,

organizations can proactively identify builds that are likely to fail, thereby saving time, reducing wasted compute resources, and maintaining developer productivity. Among the evaluated models, **XGBoost consistently outperformed others** across multiple metrics, achieving the highest accuracy, F1-score, and early warning lead time.

The superior performance of XGBoost can be explained by its ability to handle **non-linear feature interactions, high-dimensional data, and class imbalance**, which are common in build failure datasets[11]. Previous research has consistently shown that gradient boosting methods excel in predictive accuracy across diverse domains, including software defect prediction[12]. Neural networks also performed well, particularly in recall, but came with higher training costs and limited interpretability. Random Forest, while slightly less accurate, offered a balance between performance and explainability, making it a viable option for organizations prioritizing model transparency. Logistic Regression, despite its interpretability, proved insufficient for capturing the complexity of CI/CD failure dynamics.

Our findings align with recent studies that have applied **machine learning to CI/CD data**. Kazemi Arani et al. (2023) highlighted the effectiveness of ML for Continuous Integration tasks, reporting improved defect detection and resource allocation through predictive approaches[13]. Similarly, Mishra (2024) demonstrated that deep learning models could anticipate pipeline failures with significant accuracy, although interpretability remained a challenge[14]. Xu et al. (2025) extended this paradigm by exploring **LLM-based frameworks** for CI/CD, suggesting that combining ML with natural language processing on logs could further enhance prediction quality[15].

The role of AI in DevOps more broadly has also been explored through the concept of **AIOps**, where predictive monitoring and anomaly detection have become standard practices in large-scale systems[16]. Our study provides complementary evidence that similar principles apply effectively in the context of CI/CD pipelines, reinforcing the idea that **predictive intelligence is the next frontier of DevOps evolution**.

From a practical standpoint, integrating AI-driven build failure prediction into CI/CD pipelines offers several benefits. First, **resource optimization**: by halting builds earlier, teams can significantly reduce wasted computational costs. This is especially relevant in cloud-based CI/CD systems where every pipeline execution consumes billable resources[17]. Second, **developer productivity**: fewer failed builds reduce frustration and context switching, allowing developers to focus on feature development rather than firefighting errors[18]. Third, **improved release velocity**: by reducing late-stage failures, teams can sustain faster deployment cadences without compromising reliability.

Moreover, models like XGBoost provide a balance between predictive power and operational feasibility. Their ability to deliver interpretable feature importance allows teams to understand which factors most influence build failures—whether they are large code changes, dependency updates, or test suite instabilities. This actionable insight helps teams improve coding practices and pipeline configuration over time.

Despite encouraging results, several challenges remain. A major limitation is the **availability of diverse datasets**. While this study relied on open-source pipelines, enterprise environments may differ significantly in scale, logging practices, and failure types. Prior work by Houerbi et al. (2024) underscored the heterogeneity of CI/CD pipelines, noting that predictive models often need **extensive retraining and customization** to remain effective[19].

Another challenge is **explainability**. While models like Random Forest and Logistic Regression offer transparency, high-performing models such as XGBoost and neural networks are more complex. This "black box" nature can limit trust and adoption among DevOps engineers who prefer interpretable models[20]. Hybrid approaches, such as using SHAP (SHapley

Additive exPlanations) values, may help bridge this gap.

Additionally, early warning predictions must be carefully integrated to avoid **false positives** that unnecessarily block builds. A balance must be struck between **sensitivity and precision** to ensure that the system supports rather than hinders development workflows[21].

The next frontier of this work lies in **combining machine learning with large language models (LLMs)** for log analysis and contextual prediction. Recent proposals, such as Xu et al. (2025), demonstrate how LLMs can interpret natural language logs, enabling more nuanced error prediction and even automated remediation[15].

Another promising direction is the integration of **reinforcement learning**, where predictive systems continuously adapt to evolving codebases and developer behaviors. Additionally, the fusion of AI-driven prediction with **policy enforcement frameworks** can ensure not only reliable builds but also compliance with organizational standards[22].

Finally, longitudinal studies across multiple enterprises are needed to assess the **real-world economic impact** of predictive CI/CD. Metrics such as mean time to recovery (MTTR), deployment frequency, and cost savings should be tracked to quantify the return on investment of AI-augmented pipelines[23].

This study reinforces the value of **AI-driven prediction in CI/CD pipelines**. By demonstrating that models such as XGBoost can deliver early and accurate failure detection, we show how organizations can move beyond reactive pipeline monitoring to **proactive, intelligent automation**. While challenges around data diversity, explainability, and integration remain, the results suggest a clear path forward: embedding AI into DevOps practices not only improves technical outcomes but also empowers teams to deliver faster, more reliable software.

Conclusion and Recommendations

This study set out to evaluate the feasibility and effectiveness of integrating machine learning into CI/CD pipelines to predict build failures proactively. By analyzing over 100,000 build records from diverse open-source projects, we demonstrated that **AI-driven approaches can significantly improve pipeline efficiency and reliability**.

Among the tested models, **XGBoost consistently achieved the highest predictive performance**, offering superior accuracy, recall, and early warning lead time compared to Logistic Regression, Random Forest, and Neural Networks. These results underscore the potential of gradient boosting algorithms as practical tools for early-stage failure prediction in real-world DevOps environments. Neural networks also proved effective, though at the cost of interpretability and computational expense. Random Forest provided a balance of transparency and performance, while Logistic Regression was limited in predictive power.

By embedding failure prediction mechanisms within CI/CD workflows, organizations can reduce wasted compute cycles, improve developer productivity, and maintain faster, more reliable release cadences. More importantly, the ability to **detect failures earlier in the pipeline** translates into cost savings and enhanced confidence in continuous delivery practices.

Practical Recommendations

1. **Adopt Gradient Boosting Models (e.g., XGBoost):** For organizations with sufficient data, these models offer the best trade-off between accuracy and efficiency.
2. **Integrate Predictions into Existing Pipelines:** Predictions should be surfaced as alerts or early-stage blockers to prevent resource waste while minimizing false positives.
3. **Balance Accuracy and Explainability:** Random Forest or hybrid approaches may be preferable in contexts where transparency is essential. Augment black-box models with interpretability tools such as SHAP to build developer trust.
4. **Continuously Retrain Models:** CI/CD systems evolve over time. Models should be retrained periodically on fresh pipeline data to maintain accuracy.

5. **Use Predictions as Guidance, Not Gatekeepers:** To avoid disrupting workflows, AI-driven predictions should complement human judgment and existing QA processes.

Recommendations for Future Work

- **Expand Dataset Diversity:** Testing across enterprise-scale proprietary systems would strengthen the generalizability of these findings.
- **Integrate Large Language Models (LLMs):** Leveraging LLMs to analyze logs and error messages may enhance context-aware predictions and automated remediation.
- **Reinforcement Learning for Adaptation:** Adaptive systems that learn from evolving project dynamics could further optimize predictions.
- **Economic Impact Analysis:** Future research should quantify cost savings and productivity gains associated with predictive CI/CD adoption.

Acknowledgments:

The authors gratefully acknowledge the support of **Code Lab Technology Inc.** for providing technical infrastructure, DevOps consultancy, and cloud access during the execution of this study. Their contribution enabled the experimental validation and performance testing presented in this research.

REFERENCES

Koneru NMK. Infrastructure as Code (IaC) for Enterprise Applications: A Comparative Study of Terraform and CloudFormation. *Am J Technol.* 2025;4(1):1–29. doi:10.58425/ajt.v4i1.351.

Vangala V. Multi-Cloud DevOps Automation: An Empirical Study on IaC, CI/CD, and Kubernetes Orchestration. *Rev Intell Artif Med.* 2025;12(1):605–26. doi:10.5281/zenodo.15556322.

Gudelli VR. CloudFormation and Terraform: Advancing Multi-Cloud Automation Strategies. *Int J Innov Res Mod Pract Soc Sci.* 2025;11(2). doi:10.37082/IJIRMPS.v11.i2.232164.

Karanam R. Multi-cloud IaC Template Versioning and Rollback Strategies: An Empirical Study with Terraform and GitOps. *World J Adv Res Rev.* 2024;22(2):2354–63. doi:10.30574/wjarr.2024.22.2.1357.

Jayaram V, Sankiti SR, Krishnappa MS, Veerapaneni PK, Carimireddy PK. Accelerated Cloud Infrastructure Development Using Terraform. *Int J Emerg Technol Innov Res.* 2024;11(9):f382–7. SSRN:5081992.

Chijioke-Uche J. Infrastructure as Code Strategies and Benefits in Cloud Computing [dissertation]. Minneapolis (MN): Walden University; 2025.

Nuti S. Optimizing Cloud Service Delivery with Infrastructure as Code (IaC) and Platform Engineering. *J Recent Trends Comput Sci Eng.* 2025;13(4):1–13.

Özdogan E. Systematic Analysis of Infrastructure as Code Technologies. *Dergipark.* 2023; (online).

Pessa A. Comparative Study of Infrastructure as Code Tools for AWS (CDK vs Terraform) [master's thesis]. Tampere: Tampere Univ; 2023.

Zhang T, Pan S, Zhang Z, Xing Z, Sun X. Deployability-Centric Infrastructure-as-Code Generation: An LLM-based Iterative Framework. *arXiv preprint.* 2025 Jun; arXiv:2506.05623.

Chen Z, Rahman A, et al. A Defect Taxonomy for Infrastructure as Code. *arXiv preprint.* 2025 May; arXiv:2505.01568.

Hosseini S, Turhan B. A Systematic Review of Software Defect Prediction Studies. *Inf Softw Technol.* 2018;98:1–23.

Kazemi Arani A, Le THM, Zahedi M, Babar MA. Systematic Literature Review on Application of Machine Learning in Continuous Integration. *arXiv preprint.* 2023 May; arXiv:2305.12695.

Mishra A. Deep Learning Based Continuous Integration and Failure Prediction. *Comput Secur.* 2024; pre-publication.

Xu W, Luo J, Huang T, Sui K, Geng J, Ma Q, et al. A Two-Staged LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation. *arXiv preprint*. 2025 Jun; arXiv:2506.03691.

Malik H, Ghazi Y, Hassan AE. AIOps in Software Systems: Challenges and Opportunities. *IEEE Softw*. 2023;40(3):42–50.

Flexera. 2024 State of the Cloud Report. Chicago (IL): Flexera; 2024.

Ford D, Smith J, Murphy-Hill E, Zimmermann T, Bird C. Development Cycles, Build Failures, and Developer Productivity: An Empirical Study. *IEEE Trans Softw Eng*. 2017;43(1):57–75.

Houerbi A, Chavan RG, Rzig DE, Hassan F. Empirical Analysis on CI/CD Pipeline Evolution in Machine Learning Projects. *arXiv preprint*. 2024 Mar; arXiv:2403.12199.

Ribeiro MT, Singh S, Guestrin C. "Why Should I Trust You?" Explaining the Predictions of Any Classifier. In: *Proc 22nd ACM SIGKDD Conf Knowledge Discovery Data Mining*. 2016. p.1135–44.

Forsgren N, Humble J, Kim G. *Accelerate: The Science of Lean Software and DevOps*. Portland (OR): IT Revolution; 2018.