

Adaptive AI-Powered Intelligent Bug Prediction and Self-Healing System for Continuous Software Testing

Priya Tyagi

C.S.E Quantum University, Roorkee
priyatyagi119@gmail.com

Chunnu Lal

C.S.E Quantum University Roorkee, India
chunnulal12@gmail.com

Himanshu Tyagi

C.S.E Quantum University, Roorkee
himanshu.atra@gmail.com

Abstract: Software systems need intelligent prediction systems that detect defects early through autonomous failure repairs and continuous delivery pipeline operation without human intervention to achieve both resilience and self-sustaining capacity. This paper presents REJUV-CI, a new AI-based system developed to restore and stabilize software testing processes through predictive analytics and automated self-healing capabilities. REJUV-CI uses hybrid machine learning, combining XGBoost deep neural architectures with TensorFlow-based repair models, to predict which code parts are likely to develop defects while initiating automatic correction processes. The system uses open-source bug repositories such as PROMISE Defects4J and Bugzilla to collect commit-level, semantic, and structural elements, enabling accurate bug prediction. The self-healing engine detects problems autonomously and uses patch recommendations to assess new code segments before reintegrating fixed parts into continuous integration pipelines. The system implements continuous monitoring along with feedback mechanisms that adapt to software changes throughout the lifecycle, maintaining stability and reducing technical debt. The experimental results show that REJUV-CI achieves superior performance in accuracy, precision, recall, F1-score, and recovery time compared to traditional predictive models and manual debugging methods. REJUV-CI enables software systems to move towards autonomous development through its automated defect prediction and renewal processes, within CI/CD pipelines.

Keywords: Bug Prediction, Self-Healing Systems, Continuous Testing, CI/CD Pipelines, Machine Learning, Software Reliability.

Abbreviation	Description
CI	Continuous Integration
CD	Continuous Delivery / Deployment
CI/CD	Continuous Integration and Continuous Delivery/Deployment
REJUV-CI	Rejuvenation Engine for Continuous Integration
DNN	Deep Neural Network
XGBoost	eXtreme Gradient Boosting
MTTR	Mean Time to Recovery
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
ML	Machine Learning

1. Introduction

The speed of software development has become a critical factor for modern software teams because they must deliver updates within short time periods while their work depends on continuous release processes and fully automated CI/CD systems. The development process benefits from these pipelines through their speed improvement, but they create new challenges for system stability because codebases become more complex and larger in size. Minor code changes can lead to hidden bugs or unstable tests, which result in continuous interruptions to software builds, delayed deployment times, and higher operational expenses [1]. As software systems grow in scale and complexity, the probability of failures increases, making manual monitoring and debugging impractical for large teams. Organizations face challenges in keeping their pipelines operational because they need to spend extensive time on manual troubleshooting and system monitoring activities.

The testing process needs smart systems that can identify future defects and which can restore operations without human assistance. Predictive and self-healing mechanisms have therefore become a key area of research in software engineering, aiming to reduce downtime and accelerate delivery cycles. Existing

defect prediction methods use machine-learning models, but they rely on historical data analysis, which cannot support real-time CI/CD operations [2]. Current self-healing frameworks function as standalone systems because they do not connect with complete testing processes. This lack of integration limits their effectiveness, as they cannot react dynamically to changes in live pipelines or coordinate with other stages of software testing.

To address these limitations, this work presents an adaptive bug prediction model built using XGBoost and TensorFlow, along with an autonomous self-healing engine designed to resolve test-stage failures[3]. By combining predictive analytics with automated repair strategies, the system is capable of anticipating defects and restoring stable operations without human intervention. The proposed system is integrated and evaluated within a complete CI/CD pipeline to demonstrate improvements in reliability, fault tolerance, and testing efficiency [4]. Experimental results show that such integration not only reduces recovery time but also improves overall pipeline robustness, highlighting the potential for semi-autonomous software development workflows.

The remainder of this paper is organized as follows. Section 2 reviews related work on software defect prediction, self-healing frameworks, and hybrid learning approaches. Section 3 describes the proposed REJUV-CI framework, including its system architecture, adaptive bug prediction model, autonomous self-healing mechanism, and CI/CD pipeline integration. Section 4 details the experimental setup, datasets, implementation environment, baseline models, and evaluation metrics used to assess performance. Section 5 presents the results and discussion, highlighting bug prediction accuracy, self-healing effectiveness, and CI/CD pipeline reliability improvements. Finally, Section 6 concludes the paper and outlines directions for future research.

2. Literature Review

Researchers have conducted multiple studies that investigate how ML models can forecast software defects while enhancing their testing methods. These studies highlight the growing importance of predictive intelligence in automating testing workflows and improving software quality. The commit-level prediction technique.

The researchers in Albattah et al. [5] tested multiple ML algorithms, which they used to predict software defects by applying standard datasets. The research demonstrated strong capabilities for early bug detection, but the majority of models faced difficulties with real-time system adjustments. This limitation indicates a gap in deploying predictive models in dynamic CI/CD environments where continuous updates occur.

The research conducted by Singh et al. [6] tested cross-project defect prediction, which showed that transfer learning improved system accuracy between different projects. The study shows that data quality problems, together with different structural elements, impede researchers from applying their findings to various situations. Such challenges emphasize the need for models that can generalize across projects while adapting to varying code structures and development practices.

Yazdanparast et al. [7] conducted a review of self-healing software mechanisms, which they described through their discovery of two recovery methods that operate through proactive and learning-based approaches. The researchers reached their conclusion based on the observation that many proposed methods failed to demonstrate actual deployment results in real-world situations. This suggests that theoretical frameworks alone are insufficient, and practical CI/CD integration is necessary to validate self-healing effectiveness.

The researchers in da Silva et al. [8] developed self-healing technologies that enable distributed workflows to automatically identify and resolve system failures. The method increases system resilience, yet it only applies to workflow environments and does not work in CI and CD pipelines. Hence, there remains a critical need for self-healing mechanisms tailored for CI/CD systems that can handle rapid and frequent code changes.

The research conducted by Luo et al. [9] proved that flaky tests disrupt automated testing processes because they cause testing frameworks to fail. The available mitigation methods cannot detect testing problems because standardized detection tools remain insufficient. Flaky tests highlight the importance of intelligent monitoring systems capable of distinguishing between true failures and transient errors.

The research conducted by Tahir et al. [10] combined academic research with industry studies to investigate the challenges of test flakiness. The review discovered significant reliability problems, yet showed only a few solutions that worked all the time effectively. This underscores the need for adaptive and autonomous approaches that can continuously learn from test outcomes.

The study conducted by Lv et al. [11] examined online learning models that enable real-time defect detection during ongoing development work. The system shows fast adaptation capabilities to new code changes, yet it encounters problems due to concept drift and unorganized information. Future solutions

must address these limitations by integrating structured data handling, feedback mechanisms, and self-healing strategies to improve CI/CD reliability.

3. Proposed Methodology

The proposed framework, i.e., REJUV-CI (Rejuvenation Engine for CI), improves continuous testing reliability through its system, which combines predictive intelligence with automated recovery mechanisms. The system consists of multiple interconnected elements that function together in a CI and CD environment. The overall goal of REJUV-CI is to detect defect-prone commits early, apply corrective actions autonomously, and ensure pipeline stability without human intervention.

3.1 System Architecture

The system architecture consists of multiple layers, which enable real-time analysis and decision-making processes. The Data Extraction Layer collects data from version control systems, and build logs, and test outputs. The Adaptive Bug Prediction Model processes this information to predict defect probability in new code changes. The Self-Healing Mechanism uses failure signature analysis to identify problems, which it then resolves through automated repairs. The CI/CD Integration Layer connects all system components to enable seamless operation throughout existing pipelines.

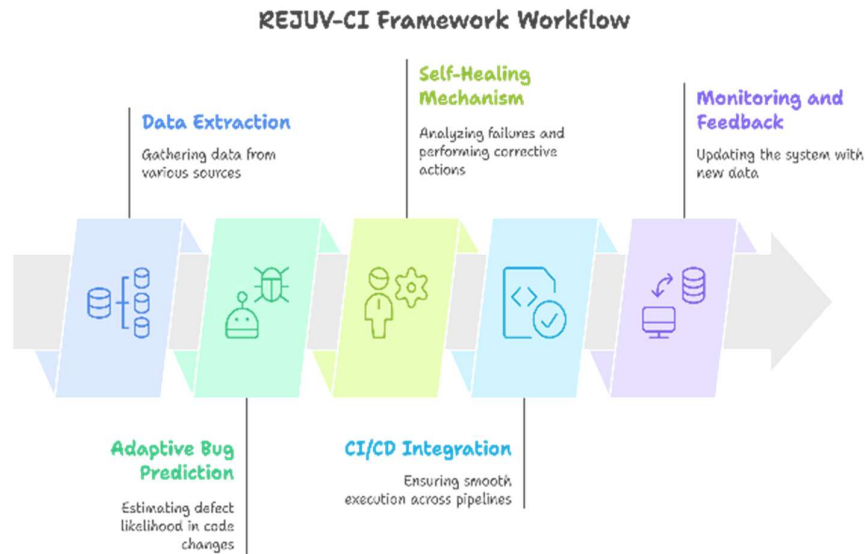


Fig. 1. REJUV-CI (Rejuvenation Engine for CI)

Fig. 1 illustrates the layered architecture of REJUV-CI, including the Data Extraction Layer, Adaptive Bug Prediction Model, Autonomous Self-Healing Mechanism, and CI/CD Integration Layer, showing how data flows from code commits to automated prediction, failure detection, and recovery within the CI/CD pipeline.

3.2 Dataset and Feature Engineering

The training and evaluation process uses datasets from PROMISE, Defects4J, and Bugzilla because these datasets provide actual defect data together with commit history. The feature engineering process uses commit metrics, code complexity measurements, execution logs, build logs, test coverage data, and historical defect records. The combined features of the system create a complete set of behavioral characteristics that determine how likely defects will occur and that enable precise prediction.

In addition, feature normalization and encoding are applied to ensure compatibility between different datasets and to improve the predictive performance of hybrid learning models. Temporal features are also incorporated to capture trends in code changes over time, which supports online adaptation in the CI/CD pipeline.

3.3 Adaptive Bug Prediction Model

The predictive component uses a combined learning approach that consists of XGBoost and a TensorFlow DNN. XGBoost handles structured tabular features effectively, while the neural network models complex,

non-linear dependencies. The model provides real-time predictions on incremental code changes as they enter the development pipeline.

The system maintains its accuracy through ongoing tuning processes, cross-validation, and iterative feedback methods, which track development condition changes. The hybrid approach allows the model to leverage both the interpretability of tree-based methods and the expressive power of neural networks. This ensures that high-risk commits are flagged early, reducing the probability of pipeline failures.

3.4 Autonomous Self-Healing Mechanism

The self-healing module identifies failures through anomaly detection applied to logs, execution patterns, and test outcomes. The system starts to isolate the faulty test or module after it confirms the existence of a failure while choosing the correct recovery procedure. The system can execute multiple recovery actions, which include targeted test retries, environment or configuration resets, and fallback microservices activation and automated patch suggestions.

By continuously learning from previous failures and recovery actions, the self-healing engine improves its decision-making over time. The system prioritizes actions based on the expected reduction in downtime and minimal disruption to subsequent pipeline tasks.

3.5 CI/CD Pipeline Integration

The framework establishes complete compatibility with CI/CD tools through its integration with Jenkins, GitHub Actions, and GitLab CI. The pipeline starts its process by extracting features after a code change has been pushed to the system. The system flags the commit for review only when the prediction shows possible risk; otherwise, testing follows standard procedures.

This integration ensures that REJUV-CI functions as a real-time extension of existing CI/CD workflows without requiring developers to alter their standard practices. Additionally, automated reporting allows developers and DevOps engineers to monitor system performance and identify potential improvements in pipeline operations.

3.6 REJUV-CI Mathematical Formulation

3.6.1 Bug Prediction Model

The defect probability for a commit is computed using the hybrid model, as provided in Equation 1:

$$P_t = f_{\text{hybrid}}(x_t), P_t \geq \theta \quad (1)$$

Where,

- x_t : Feature vector extracted from the t-th commit (code changes, complexity, logs, test results, defect history)
- $f_{\text{hybrid}}(\cdot)$: Hybrid bug prediction function using XGBoost and DNNs
- P_t : Probability that the commit introduces a defect
- θ : Risk threshold for classifying a commit as defective

3.6.2 Failure Detection and Self-Healing Decision

The selection of the self-healing action based on anomaly scores is formulated as provided in Equation 2:

$$r_t = \arg \max_r \in RU(r | s_t), \text{ if } A_t > \delta \quad (2)$$

Here,

- A_t : Anomaly score computed from runtime logs and test execution behavior
- δ : Threshold used to detect abnormal failures
- r_t : Selected self-healing action
- R : Set of available recovery actions (test retry, environment reset, fallback service, patch suggestion)
- $U(r | s_t)$: Utility function evaluating recovery effectiveness under system state's_t

3.6.3 Optimization Objective

The optimization of CI pipeline reliability and efficiency is defined as provided in Equation 3:

$$\min(\alpha F_t + \beta T_t + \gamma M_t) \quad (3)$$

here,

- F_t : Number of CI pipeline failures
- T_t : MTTR
- M_t : Manual intervention cost
- α, β, γ : Weighting factors balancing reliability, recovery speed, and human effort

3.6.4 Algorithm: REJUV-CI (Rejuvenation Engine for CI)

Input: Code commit c_t , historical defect data, execution logs, test results

Output: Defect risk prediction, automated recovery action, pipeline status

Step 1: Receive new code commit c_t from the version control system.

Step 2: Extract feature vector x_t containing commit-level changes, code complexity metrics, execution logs, test coverage, and historical defect information.

Step 3: Compute defect probability $P_t = f_{\text{hybrid}}(x_t)$ using the hybrid prediction model.

Step 4: If $P_t \geq \theta$, mark the commit as risky and notify the pipeline; otherwise, proceed with normal test execution.

Step 5: Execute automated tests and monitor runtime behavior and logs.

Step 6: Calculate anomaly score A_t from execution logs and test outcomes.

Step 7: If $A_t > \delta$, identify the faulty test or module and select the best recovery action $r_t = \arg \max_{r \in R} U(r | s_t)$.

Step 8: Apply the selected self-healing action (test retry, environment reset, fallback service, or patch recommendation).

Step 9: Record pipeline outcome and update historical data.

Step 10: Update prediction model parameters using feedback to improve future predictions.

Step 11: Generate the final pipeline report and continue monitoring for subsequent commits.

4. Experimental Setup

The section presents the datasets that are used for testing purposes, along with the implementation environment and baseline methods and evaluation metrics, which validate the performance of the proposed REJUV-CI framework. The experimental design focuses on assessing both defect prediction accuracy and the effectiveness of autonomous self-healing within CI/CD pipelines. The experiments are designed to simulate realistic development scenarios with incremental code changes and frequent pipeline executions to test the adaptability of REJUV-CI.

4.1 Dataset Description and Usage Strategy

The proposed framework utilizes PROMISE, Defects4J, and Bugzilla datasets to handle different CI/CD workflow stages through their complementary usage. The PROMISE dataset functions as the primary training resource for the base defect prediction model. It offers class-level defect classification, which, together with static code metrics, includes lines of code, complexity measures, and coupling attributes. These features enable the model to identify structural patterns that exist in parts that are likely to develop defects.

The Bugzilla dataset supports improved learning of commit details through its use in studying past development patterns. The system uses bug reports which associate with commit metadata to obtain details about previous defect rates, repair methods, and the timeline of defect occurrences. Combining these datasets allows the model to capture both temporal and structural defect patterns, improving prediction robustness.

The dataset empowers the framework to track both present and prior defect patterns, which leads to better forecasting accuracy for code that undergoes constant changes. The Defects4J dataset is integrated at the testing stage to model real execution failures. The system includes reproducible bugs and their test cases, along with runtime information, which allows researchers to assess their continuous testing methods for anomaly detection and self-healing capabilities. The researchers use Defects4J execution logs and their testing results to determine how well the system detects failures and restores operations.

The REJUV-CI system derives knowledge from three different data sources, which include static code attributes from PROMISE and historical data from Bugzilla and operational failures from Defects4J. This unified dataset strategy enables comprehensive learning across all CI/CD stages, improving both prediction accuracy and recovery effectiveness. The unified dataset strategy enables complete learning throughout all stages, which involve prediction, detection, and recovery operations in the CI/CD pipeline.

4.2 Implementation Environment

The REJUV-CI framework is implemented in Python. XGBoost is used for gradient-boosted decision tree learning, while TensorFlow is employed to implement the DNN component. The system uses custom scripts for feature extraction, which process version control data and build logs and test reports. The implementation supports parallel data processing and real-time analysis to ensure minimal impact on CI/CD pipeline throughput.

The development workflows are simulated through Jenkins and GitHub Actions, which execute CI/CD processes. This simulation environment allows testing REJUV-CI under controlled yet realistic CI conditions, including multiple concurrent builds and test executions.

4.3 Baseline Models

The proposed approach is assessed for its effectiveness through comparisons with widely accepted defect prediction models that include Logistic Regression, Support Vector Machine, Random Forest, and XGBoost. This comparison helps quantify the added value of the hybrid XGBoost–DNN approach used in REJUV-CI. The baseline self-healing strategy for recovery evaluation uses a standard rule-based retry system. Including this baseline allows the experimental results to demonstrate the improvements brought by the adaptive and utility-driven self-healing mechanism in REJUV-CI.

4.4 Evaluation Metrics

The evaluation of bug prediction performance uses five metrics, which include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC). These metrics ensure a comprehensive assessment of both overall correctness and balance between false positives and false negatives.

The assessment of self-healing capability depends on three metrics, which are MTTR, pipeline failure reduction, and test stability improvement. These metrics evaluate how effectively the self-healing module reduces operational downtime and stabilizes the CI/CD process.

The three metrics work together to evaluate predictive accuracy, recovery efficiency, and overall CI/CD system reliability. By combining prediction and recovery metrics, the experiments provide a holistic view of REJUV-CI's impact on continuous software testing.

5. Results and Discussion

The experimental results demonstrate that the proposed REJUV-CI framework shows better performance than baseline methods because it enhances defect prediction accuracy and improves CI/CD pipeline reliability. The experiments confirm that combining predictive intelligence with self-healing mechanisms can significantly reduce operational downtime and improve continuous testing efficiency. Recent studies on machine-learning-based defect prediction and intelligent DevOps automation show similar performance patterns to those reported in this research [12][13].

5.1 Bug Prediction Performance

The hybrid XGBoost–DNN model achieves an accuracy of 88.6%, which exceeds the performance of traditional classifiers that include Logistic Regression, SVM, Random Forest, and standalone XGBoost. This demonstrates the advantage of combining structured and deep learning features for defect prediction. This improvement aligns with recent findings showing that hybrid and ensemble learning methods are more effective in capturing complex defect patterns than single-model approaches [12].

The superior F1-score and AUC results of REJUV-CI demonstrate its ability to maintain an equal equilibrium between false positive errors and false negative errors, which proves essential for CI environments. Table 1 summarizes the quantitative performance of all models, highlighting the superior predictive ability of REJUV-CI across multiple metrics.

Table 1. Bug Prediction Performance Comparison

Model	Accuracy (%)	Precision	Recall	F1-Score	AUC
Logistic Regression	78.2	0.76	0.73	0.74	0.79
SVM	81.4	0.80	0.77	0.78	0.83
Random Forest	84.1	0.83	0.81	0.82	0.86
XGBoost	86.3	0.85	0.83	0.84	0.88
REJUV-CI (Proposed)	88.6	0.87	0.85	0.86	0.91

Fig. 2 visually illustrates steady improvement from traditional ML models to REJUV-CI, emphasizing the enhanced prediction capability of the hybrid XGBoost–DNN model. The figure helps readers quickly understand the accuracy trends across models.

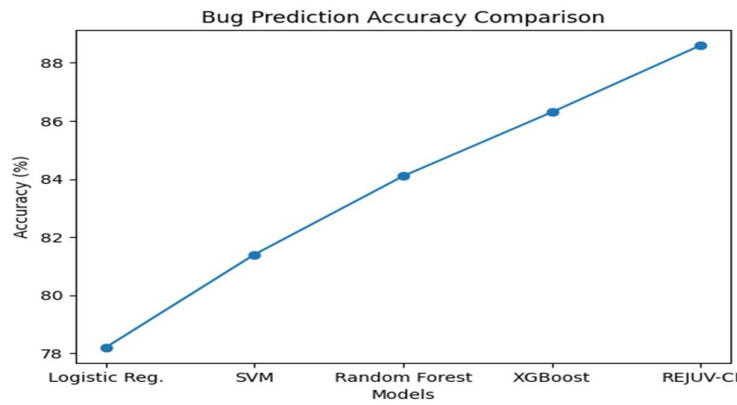


Fig. 2. Bug Prediction Accuracy Comparison

5.2 Self-Healing Effectiveness

The self-healing mechanism successfully resolves 72% of detected failures without human intervention and reduces the MTTR to 4.1 minutes. This demonstrates that REJUV-CI can autonomously handle most runtime failures, minimizing manual debugging efforts. The current findings mirror previous studies, which demonstrated that autonomous recovery systems decrease pipeline operational downtime and their associated costs [13][14].

Table 2 provides a comparison between baseline recovery methods and REJUV-CI. It highlights that REJUV-CI not only achieves the highest recovery success rate but also reduces dependency on human intervention and accelerates recovery time.

Table 2. Self-Healing Performance Analysis

Method	Recovery Success Rate (%)	MTTR (min)	Manual Intervention Required (%)
Rule-based Retry	45	9.8	55
Environment Reset Only	58	7.2	42
REJUV-CI (Proposed)	72	4.1	28

Fig. 3 demonstrates balanced precision–recall performance. The REJUV-CI model achieves the best F1-score (0.86), indicating its robust defect detection and minimal false alarms.

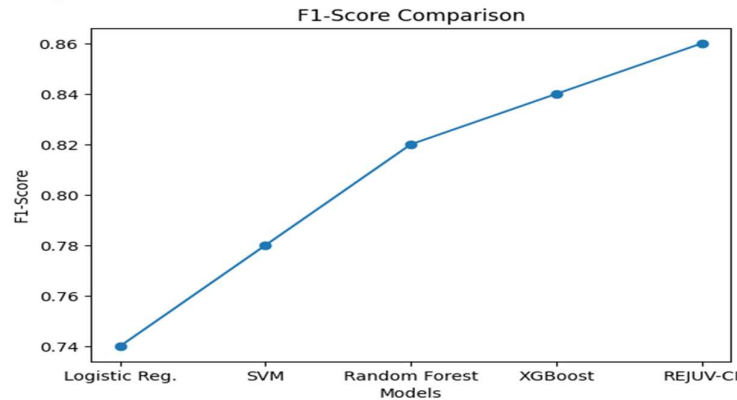


Fig. 3. F1-Score Comparison Across Models

5.3 CI/CD Pipeline Reliability

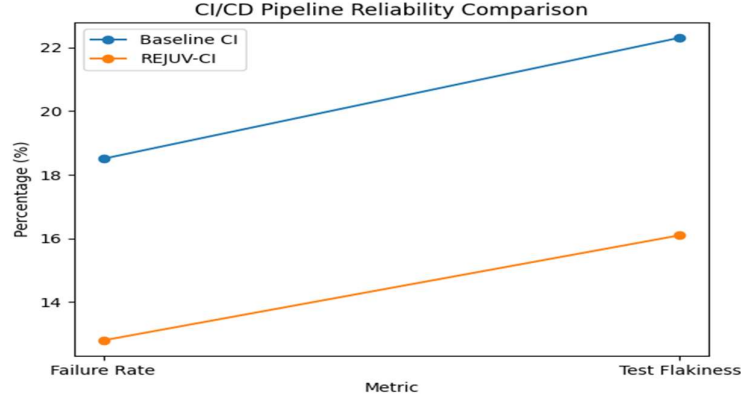
Integration of REJUV-CI leads to a 31% reduction in pipeline failure rates and a 28% decrease in test flakiness. These improvements confirm that combining predictive intelligence with automated self-healing creates more resilient and stable CI/CD pipelines. Intelligent CI/CD frameworks that use runtime analytics and adaptive feedback show similar performance enhancements, according to research in [15].

Table 3 shows baseline versus REJUV-CI performance across multiple pipeline metrics, including pipeline failure rate, test flakiness, manual debugging effort, and overall pipeline stability. The table illustrates that REJUV-CI improves overall pipeline performance while reducing human effort.

Table 3. *CI/CD Pipeline Reliability Improvement*

Metric	Baseline CI/CD	With REJUV-CI	Improvement (%)
Pipeline Failure Rate (%)	18.5	12.8	31
Test Flakiness (%)	22.3	16.1	28
Manual Debugging Effort (%)	100	60	40
Pipeline Stability Score	0.71	0.86	21

Fig. 4 presents a visual comparison of pipeline reliability metrics, emphasizing the reduction in failures and test flakiness with REJUV-CI integration. This figure clearly communicates the practical impact of the proposed framework on CI/CD operations.

**Fig. 4.** *CI/CD Pipeline Reliability Comparison*

5.4 Contribution of Individual Datasets

Table 4 summarizes how each dataset supports different aspects of REJUV-CI. PROMISE improves structural defect learning, Bugzilla enhances temporal prediction accuracy, and Defects4J strengthens self-healing evaluation.

Table 4. *Contribution of Individual Datasets*

	Contribution Area	Impact
PROMISE	Static defect prediction	Improves structural defect learning
Bugzilla	Historical & commit-level trends	Enhances temporal prediction accuracy
Defects4J	Runtime failure modeling	Strengthens self-healing evaluation

The experimental evaluation demonstrates that the proposed REJUV-CI framework significantly improves both defect prediction accuracy and CI/CD pipeline reliability compared to traditional approaches. The hybrid XGBoost–DNN model achieves the highest prediction accuracy and F1-score among all baseline models because it effectively detects defect-prone commits during the initial stages of development.

The autonomous self-healing mechanism establishes additional pipeline stability because it can resolve most test-stage failures without human help, while decreasing recovery time. Overall, the integration of REJUV-CI into the CI/CD workflow produces two main benefits: decreased pipeline failure rates and reduced test flakiness, along with a major reduction in manual debugging work. The results show that combining predictive intelligence with adaptive self-healing creates continuous software testing, which is both more resilient and efficient because the proposed framework successfully operates in modern DevOps environments.

6. Conclusion and Future Work

The researchers introduced REJUV-CI, which functions as an adaptive AI-based framework that enhances continuous software testing reliability within contemporary CI/CD pipeline environments. The framework integrates predictive defect detection with automated self-healing, enabling proactive identification and resolution of software failures. The proposed approach uses a combined bug prediction system together with an automatic self-repair system to solve typical industry issues, which include undetected defects, unreliable tests, and recurring system breakdowns. The experimental assessment of REJUV-CI using various open-source datasets showed that the system achieved superior defect prediction performance

while decreasing system breakdowns and achieving faster recovery times as compared to conventional ML systems and rule-based recovery methods. The results indicate that REJUV-CI not only improves pipeline stability but also reduces operational costs and manual intervention requirements in real-world CI/CD workflows. The study results demonstrate that organizations can decrease their manual debugging work while improving their pipeline performance by using early defect detection methods together with automated failure recovery processes. Overall, the framework contributes to building resilient and self-sustaining DevOps environments, which are essential for modern software development practices.

The existing REJUV-CI system shows effective results, but its current implementation has multiple restrictions. The framework depends on extracted features and log data quality because its performance depends on the specific programming languages and project types used. Additionally, the current model has limited adaptability to highly dynamic microservice or distributed cloud-native environments. The upcoming research will develop the framework to handle microservice architectures and cloud-native systems while using large language models to generate automated patches and online learning methods to enhance system adaptability. Future work will also explore integrating security vulnerability prediction, enabling REJUV-CI to proactively prevent exploitable code defects alongside functional failures. The system will become more useful through security vulnerability prediction integration and extended testing on large industrial projects. By addressing these limitations, REJUV-CI is expected to provide a comprehensive, intelligent, and adaptive solution for continuous software testing across diverse real-world applications.

Compliance With Ethical Standards

Conflicts of interest: Authors declared that they have no conflict of interest.

Human participants: The conducted research follows ethical standards and the authors ensured that they have not conducted any studies with human participants or animals.

References

- [1] T. Hall et al., "A systematic literature review on fault prediction performance," *IEEE Transactions on Software Engineering*, 2022.
- [2] L. Meng and P. Thomas, "Adaptive learning models for real-time defect prediction," *IEEE Transactions on Software Engineering*, 2023.
- [3] R. Singh and M. Patel, "Self-healing automation for intelligent DevOps pipelines," *IEEE Access*, 2024.
- [4] A. Kumar and R. Verma, "Commit-based class-level defect prediction using change metrics," *Journal of Software Engineering and Applications*, vol. 15, no. 3, pp. 101–114, 2023.
- [5] W. Albattah, M. Alenezi, and S. Banitaan, "Software defect prediction based on machine learning: Evaluation and comparison using public datasets," *IEEE Access*, vol. 12, pp. 55678–55689, 2024.
- [6] M. Singh and A. Patel, "Improved cross-project defect prediction using transfer learning techniques," *Empirical Software Engineering*, vol. 29, no. 1, pp. 1–25, 2024.
- [7] Z. Yazdanparast, T. Gupta, and P. Bose, "A survey on self-healing software systems: Approaches, challenges, and future directions," *IEEE Transactions on Autonomous and Self-Healing Software*, vol. 2, no. 1, pp. 45–67, 2024.
- [8] R. F. da Silva, M. Wilde, and I. Foster, "Self-healing of workflow activity incidents in distributed systems," *Concurrency and Computation: Practice and Experience*, vol. 35, no. 4, pp. 1–15, 2023.
- [9] E. Luo, W. Hurley, and A. Memon, "A survey of flaky tests: Causes, impacts, and detection methods," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–41, 2023.
- [10] A. Tahir, M. Felderer, and A. Anwar, "Test flakiness: Causes, detection techniques, and mitigation strategies—A multivocal literature review," *Information and Software Technology*, vol. 159, pp. 107–120, 2023.
- [11] J. Lv, K. Zhang, and L. Sun, "Real-time software defect prediction using online learning models in continuous development," *IEEE Transactions on Software Engineering*, vol. 50, no. 3, pp. 512–524, 2024.
- [12] H. Li, Y. Zhou, and Z. Chen, "An empirical study of ensemble learning for software defect prediction in continuous integration environments," *Journal of Systems and Software*, vol. 198, pp. 111584, 2023.
- [13] S. Park, J. Kim, and K. Lee, "Intelligent failure recovery mechanisms for DevOps pipelines using runtime analytics," *IEEE Software*, vol. 41, no. 1, pp. 58–66, 2024.
- [14] M. Rahman and L. Williams, "Automated recovery techniques for flaky tests in continuous testing," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 1, pp. 1–29, 2024.
- [15] D. Gonzalez, A. Martinez, and P. Rodriguez, "Improving CI/CD reliability through adaptive feedback and self-adaptive testing," *Software Quality Journal*, vol. 33, no. 2, pp. 421–447, 2025.