International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 6, Issue 4, 24-31, 2025
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V6I4P104

Original Article

Reinforcement Learning Driven Adaptive Software Testing with Continuous Fault Anticipation and Self-Healing Feedback Loops in SAP

Akhilesh Kumar Aleti
Operations IT Data Manager, The Dow Chemical Company, Dallas, Texas, USA.

Received On: 19/08/2025 ***Revised On: 23/09/2025*** ***Accepted On: 30/09/2025*** ***Published On: 19/10/2025***

Abstract - *With the changing dynamics in software development comes the need for a smart testing framework that can adapt to changes in the environment. Our approach brings a new way of viewing reinforcement learning (RL) and adaptive software testing, that combines them to constantly build a fault anticipation and self-healing mechanisms. The ultimate goal is to create some autonomous testing tool which understands and learns the defect patterns from history, and through feedback loops optimizes the evolution of the test case generation. We utilize Q-learning algorithms via Markov Decision Processes to create adaptive testing agents that learn from the results of fault detections. The hybrid model that is built-in to LSTM enables predictive fault prediction, and the self-healing mechanisms automatically change testing parameters. Over a half-a-year span of testing around 15,000 test cycles across five open-source Google projects of differing complexity, an illuminating study was performed. The results show an increase of 47.3% in fault detection rates and decrease of 38.6% in testing time in comparison with traditional approaches. NOTE: Statistical analysis presents strong associations between adaptation driven by RL & defect prediction. Our proposed framework gets 92.4% precision concerning vital faults to predict before deployment. This study adds value to autonomous software quality assurance by creating intelligent self-optimizing testing ecosystems.*

Keywords - *Reinforcement Learning, Adaptive Testing, Fault Anticipation, Self-Healing Systems, Continuous Feedback.*

1. Introduction

It is one of the most important phases of the software development lifecycle and is responsible for taking up to 40–50% of the overall development resources (Gunda et al., 2025). However, traditional testing methodologies are done with pre-specified test suites that are not able to adapt to changing software architectures and new fault patterns. The complexity of modern software is doubling every two years (Katz, 2023) and now that agile and DevOps practices have made their way into every large organization, deployment cycles are at an all-time rapid pace which compresses the time a testing/tasking need to be positive [8] and pushes for more intelligent testing frameworks capable of self-learning and optimal testing strategy selection (Chen et al., 2023). Automated testing tools today are more about automating the execution of tests rather than making intelligent choices about what, when and how to test. Reinforcement Learning (RL) is a promising approach to build adaptive systems that learn how to behave through interaction with their environment (Sutton & Barto, 2018). For example, in software testing, RL agents can leverage historical defect data and real-time feedback to learn to prioritize test cases, identify high-risk modules, and optimize resource allocation (Pan et al., 2022). Combining RL with software testing overcomes two major weaknesses of static testing techniques as it can continuously learn from testing results and faulty discoveries.

Machine learning has shown immense promise in the domains of software defect prediction and vulnerability detection (Gunda, 2024a; Gunda, 2025b). But these methods are usually pre-deployment prediction based, and they do not adapt progressively during test times. Self-healing systems have become a trend in autonomous computing [3], where automatic detection, diagnosis, and recovery of failures happen without any human intervention. The integration of RL driven adaptive testing with self-healing creates a full spectrum framework for autonomous SQA. On the other hand, fault prediction boils down to a non-reactive software testing, meaning that it supply feedback on potential defects that are likely to occur in production environments (Gunda, 2024c). Code smell prediction models that also support the dynamic nature of software systems are mostly missing, as traditional fault prediction models are static and mainly rely on historical data. Integrating Continuous Feedback Loops with real-time results, testing frameworks can adjust in strategy, creating cycles of improvement that lead to higher efficiency and effectiveness.

The framework proposed in the paper deals with some of the most daunting problems faced in modern software testing: traditional testing methods are inflexible and incapable of adapting to evolving software environments. It suffers from inefficient resource utilization when executing test cases, lacks in providing immediate feedback (i.e., it delays the revelation of faults), thus increasing the defect-remediation costs, and it is unable to predict early (or, at least, earlier) on whether new problems will arise, etc. Enabled by RL algorithms, the framework builds smart testing agents that learn the best testing policies by exploration and exploitation strategies and adapting policies based on accumulated knowledge. A consolidated architecture for Q-learning algorithms and Long Short-Term Memory (LSTM) networks to allow adaptive test generation and predictive fault prediction is introduced in this research. Self-healing element: Automatically alters testing parameters, rebalances the priority of the test case, and optimizes resource allocation according to feedback got from the result of fault detection. These never ending feedback loops guarantees that the system evolves with the software under test which keeps it relevant and effective throughout the development lifecycle.

2. Literature Review

Machine learning and software testing have crossed paths for a decade now, and the intersection has gained considerable maturity over this time-period. Numerous research studies explained traditional software testing methodologies including systematic approaches on test case generation, test suite optimization, and defect prediction Myers et al. (2011). However, the aforementioned approach in modern software development, as a form of dynamic nature, lacks some formulation type. Applying Reinforcement Learning to software engineering has been studied extensively. Mahmud et al. We have seen examples, such as RL-based test case prioritization in (2021), where the authors showed that by using Q-learning, they could train agents to become better than conventional test case prioritization heuristics by 23%. While their work laid down some of the fundamental principles for using RL to perform testing optimally, it did not consider continuous adaptation or self-healing capabilities. Similarly, Spieker et al. Similarly, RETECS (2017) is an RL-based test case selection framework that is able to substantially increase fault detection rates via dynamic prioritization strategies. Machine learning techniques have taken the idea of software defect prediction to a new dimension. Gunda et al. Based on such data, Hussain et al. (2025) showed that scalable machine learning models can help to improve software reliability in agile environments by early predicting faults and achieving moderate improvements on sprint planning efficiency. As observed in (Gunda, 2024a), their classification ensemble-based approach that utilized both boosting and voting methods performed better than classifiers used separately, as well. In addition, it has been shown that there are strong relations between static characteristics of the code and defect proneness (Gunda, 2025b), which suggests that it might be a possible indicator in the context of vulnerability detection with the help of code metrics and feature extraction.

The outcomes of comparative studies on machine learning models for software fault prediction indicate the ineffectiveness of one model over the other could be dataset- and context-dependent (Gunda, 2024c; Gunda, 2024d). The articles turned the focus to three different aspects; feature selection, model selection and characteristics of the dataset for accurate predictions. But prior approaches mostly tackle a static prediction model without adaptation of software traits over time. Self-healing systems is a concept, first introduced with the autonomous computing research. Motivation Self-adapting frameworks that could automatically detect and respond to certain system anomalies were proposed by Psaier and Dustdar (2011). Although their work pointed out the importance of feedback loops in keeping the system healthy, it was not related to a software testing context. Abstract: Self-healing architectures have recently shown promise in achieving automatic fault recovery in systems operating in production environments. Microservices have created new challenges as well as opportunities for adaptive testing, especially in CI/CD environments. Elbaum et al. In a similar vein, (2014) focused on regression test selection in continuous integration environments, emphasizing the challenge of intelligently prioritizing test cases as the need to consider the (time-bounded) trade-off between testing depth and breadth increases. They highlight the importance of adaptive approaches that can consider code changes over time, as well as fault history.

The success of deep learning applications in software testing has emerged at a rapid pace. White et al. The work by Yu et al. (2019) used recurrent neural networks to learn the patterns of generating test cases, resulting in state-of-the-art neural network capabilities to model the relations between snippets of code and the correct test cases. And LSTM networks are very useful to model sequential dependencies for software evolution and fault occurrences patterns (Wang et al., 2018). Specific to the test case selection problem, there are a subset of RL techniques known as multi-armed bandit algorithms, which have been applied in the literature. Spieker et al. In (2017), they demonstrated that bandit strategies are effective in balancing exploration of untested scenarios with exploitation of high-value tests. While their work explored the test architecture of reward function design that would facilitate testing optimization, it did not include predictive components of fault anticipation. Integration of machine learning has been beneficial to fault localization and diagnosis. Wong et al. Thus, various spectrum-based fault localization techniques are explored by (2016), which exhibits the potential of boosting debugging processes based on program execution profiles. Combining RL with faulty localization is a new research direction, in which agents learn how to navigate codebases and quickly find likely defect sources. The main contribution of our work is that we propose a single framework that seamlessly

integrates RL-based adaptation, continuous fault prediction and self-healing. Although individual components have been explored, comprehensive systems synthesizing these and more to fully autonomous testing ecosystems have not yet been developed. This study aims to bridge this gap through an integrated architecture that combines the synergy of adaptive learning, predictive anticipation, and automated remediation.

3. Objectives

The primary objectives of this research are as follows:

- To develop an RL-based adaptive testing framework using Q-learning for optimized fault detection and resource efficiency.
- To integrate LSTM-based predictive models for proactive fault anticipation and risk mitigation.
- To implement self-healing feedback mechanisms for dynamic adjustment of testing parameters and resource allocation.
- To validate the framework through empirical evaluation on open-source projects, assessing improvements in fault detection, efficiency, and predictive accuracy.

4. Methodology

We used the experimental quantitative research method in the integration of behavior coupling (BC)-based RL and adaptive software testing frameworks in this study. This methodology involved developing system architecture, implementing algorithms, empirical evaluation, and statistical analysis across five open-source software projects from different domains including web applications, databases, and utilities. Over a period of 6 months, 15,000 testing cycles were conducted on project implementations (between 5,000–50,000 lines of code), across three phases (baseline testing, RL-driven implementation, or comparison). This framework architecture included three main components, that of an rl agent, a long-short term memory (LSTM) based fault prediction module and a feedback controller for self-healing. Test case optimization for defect detection was learned by the RL agent using Q-learning with an epsilon-greedy strategy (learning rate = 0.1, discount factor = 0.9) so that the test cases are selected and prioritized according to the efficiency to detect defect and resource consumption. The LSTM module was trained to predict high-risk modules based on code metrics, temporal patterns, and historical defects over three layers with 128 hidden units each. With periodic performance monitoring, the self-healing controller adjusted parameters and resource allocation in sub 100 ms intervals to keep the system efficient. Execution time, defect detections, coverage metrics and resource utilization were captured through automated data collection. Paired t-tests, correlation, and regression modeling, as appropriate, were conducted at the 0.05 level to evaluate improvement in fault detection, predictive accuracy, and efficiency. Proposed RL-driven adaptive testing framework robustness and generalizability were ensured via cross-validation and transfer learning.

5. Results

We conducted extensive experiments to acquire performance data along multiple dimensions for the proposed RL-driven adaptive testing framework. Fault Detection Effectiveness, Testing Efficiency Improvements, Predictive Accuracy and Self-Healing Mechanism Performance Statistical Analysis This section provides the results of an extensive statistical analysis of all aspects for fault detection effectiveness, testing efficiency improvements, predictive accuracy, and self-healing mechanism performance.

Table 1: Comparative Performance Metrics Across Projects

Project	Traditional FDR (%)	RL-Driven FDR (%)	Improvement (%)	Testing Time Reduction (%)	Resource Efficiency Gain (%)
Project A	64.2	94.8	47.7	42.3	36.8
Project B	68.5	96.3	40.6	38.2	41.2
Project C	61.8	91.7	48.4	35.7	34.5
Project D	66.3	93.2	40.6	39.8	38.9
Project E	62.9	92.1	46.4	37.4	35.7
Average	64.7	93.6	44.7	38.7	37.4

As shown in Table 1, all five experimental projects have shown great enhancement in FDR. The RL-driven adaptive framework secured an overall fault detection rate of 93.6%, outperforming the standard testing approaches with an average of 64.7% by 44.7%. Projects A (+47.7%) showed the most notable improvement, followed by Projects B and D (+40.6%) with the lowest but still considerable gains. The overall reduction in testing time was 38.7% across all the projects, and the maximum reduction of 42.3% (project a) The average gain in resource efficiency, defined as the fractional use of computational resources for each fault detected, from the baselines controlled with D4C was 37.4% a strong positive sign that D4C optimizes the allocation of testing resources. The robustness of the RL-driven approach across different projects was confirmed by statistical analyses using paired t-tests, showing all improvements significant at $p < 0.001$ level.

Table 2: Fault Anticipation Accuracy Metrics

Metric	Precision (%)	Recall (%)	F1-Score (%)	False Positive Rate (%)	Anticipation Lead Time (hours)
Critical Faults	92.4	88.7	90.5	4.8	48.3
Major Faults	87.6	85.3	86.4	7.2	36.7
Minor Faults	81.3	79.8	80.5	11.4	24.6
Overall	87.1	84.6	85.8	7.8	36.5

The performance of the LSTM-based fault anticipation module over different fault severity categories is shown in Table 2. The system thus achieved an excellent precision of 92.4% on killer bugs and proved that deadly defects can be predicted with high precision before manifestation. Recall rates were high as well (88.7%) for critical faults corresponding to actual coverage of a critical defect. The F1-scores (harmonic means of precision and recall) were 80.5% for minor faults, 87.5% for major faults, 90.5% for critical faults, and an overall score of 85.8%. The false positive rates were keeping very low, especially for critical faults, at 4.8%, so it is preventing wasting efforts in testing on previously predicted defects that are not true. Critical faults had the highest anticipation lead time with an average of 48.3 hours between prediction and actual occurrence, allowing sufficient time for proactive testing remediation. These metrics as a whole provide a strong indicator of how well the predictive component is performing in allowing testers to shift right.

Table 3: Self-Healing Mechanism Performance

Adaptation Trigger	Occurrence Frequency	Average Response Time (ms)	Success Rate (%)	Performance Recovery (%)	Intervention Accuracy (%)
Coverage Decline	342	87.3	94.7	89.3	91.8
False Positive Spike	256	76.4	96.2	92.1	94.3
Resource Bottleneck	198	94.8	91.4	86.7	88.9
Detection Rate Drop	287	82.1	95.3	90.8	92.6
Overall	1083	85.2	94.4	89.7	91.9

The performance of self-healing feedback controller is summarized in Table 3 and categorized into different types of adaptation triggers that received a self-healing feedback controller during the execution time. In total, 1,083 adaptation events were identified, the most common cause of adaptations was coverage decline with 342 occurrences, i.e. when test coverage metrics get worse and require some intervention. The average response time for all trigger types was 85.2 milliseconds, suggesting capability for very close to instantaneous adaptation. The average success rate (i.e. whether the self-healing intervention could fix the condition that triggered it) was 94.4%: with the highest being at 96.2% with false positive spikes. Using recovery metrics, it was estimated that 89.7% of testing capability was restored to pre-degradation levels with interventions. The intervention accuracy, the average score (91.9%) of whether the applied adjustments are appropriate for the triggering condition, shows that the controller is able to make an intelligent decision. This confirms that the self-healing mechanism is able to continuously adapt to perform testing optimally, as designed.

Table 4: RL Agent Learning Progression

Testing Cycle Range	Average Q-Value Convergence	Exploration Rate (%)	Optimal Policy Achievement (%)	Cumulative Reward	Average Episode Length
1-3000	0.342	45.8	62.3	2847.6	156.3
3001-6000	0.568	32.4	78.6	4926.8	142.7
6001-9000	0.721	21.7	87.4	6834.2	128.4
9001-12000	0.843	15.3	92.8	8247.9	118.6
12001-15000	0.891	12.6	95.7	9163.5	112.3

The table 4 is an example of the performance of the RL agent over 15,000 test cycles, split into five sections of 3,000 cycles each. The Q-value convergence metric indicating stability of learned state-action values (bottom right) improved from 0.342 to 0.891 between the first and last phase indicating significant learning and policy convergence. The exploration rate defined by the epsilon-greedy strategy was systematically decreased from 45.8% to 12.6%, indicating the transition of the agent behavior from exploration-dominant to exploitation of learned optimal strategies. We saw a stark improvement in optimal policy achievement (the percentage of decisions taken that are aligned with theoretically optimal actions) from an average of 62.3% to 95.7% which is a good indication that we are learning. In all phases, cumulative rewards increased, confirming the agent's increasing ability to simultaneously detect faults and optimize resources. Number of testing actions per cycle averaged over all testing iterations: Start: 156.3; End: 112.3 — reduction of 28% shows that with the learning cycle in progress, the test cases chosen became more efficient. These trends confirm that at the end of the experiment, the RL

agent was capable of adapting and improving consistently over iterations.

Table 5: Feature Importance in Fault Anticipation

Feature Category	Importance Score	Contribution to Prediction (%)	Correlation with Actual Faults	Ranking
Cyclomatic Complexity	0.284	28.4	0.762	1
Code Change Frequency	0.247	24.7	0.698	2
Lines of Code Modified	0.189	18.9	0.643	3
Historical Defect Density	0.156	15.6	0.587	4
Module Coupling	0.124	12.4	0.521	5

Feature importance analysis for the LSTM-based fault anticipation module, as shown in Table 5, indicates which code metrics had the largest influence on predictive accuracy. Of all these, cyclomatic complexity proved to be the most influential feature with an importance measure of 0.284 — contributing 28.4% of the overall predictive capability. With the correlation of 0.762 with the occurrence of faults, we assert that this metric plays a role in indicating defect-prone code areas. The second-most influential measure with a contribution of 24.7% was code change frequency, mirroring the empirical foundation between code volatility and the introduction of defects [8]. The lines of code modified were directly responsible for 18.9% with historical defect density and module coupling responsible for 15.6% and 12.4% respectively. The high correlation coefficients >0.5 for each feature confirm that all features are meaningful features for the ground truth fault occurrence. These results guide in feature selection for fault prediction approaches and serve as an observation for defect prone software components.

Table 6: Cross-Project Transfer Learning Performance

Source Project	Target Project	Initial Transfer FDR (%)	Post-Adaptation FDR (%)	Adaptation Cycles Required	Knowledge Retention (%)
Project A	Project B	78.4	93.8	847	83.6
Project A	Project C	74.2	91.3	1023	79.4
Project B	Project D	76.8	92.7	892	81.7
Project C	Project E	72.9	89.6	1154	76.3
Average	-	75.6	91.9	979	80.3

Table 6 explores the framework's ability to transfer learning, measuring the success of RL agents trained on one project when moving to different projects. In support of transfer fault detection, agents was able to maintain a reasonable level of knowledge – with an average initial transfer fault detection rate of 75.6% across contexts and no project-specific retraining. Following adaptation periods averaging 979 cycles, corresponding transferred agents attained an average FDR of 91.9%, close to the performance of agents trained directly on target projects. The number of training cycles needed for adaptation depended on the determined similarity of the projects considered with transfers between more similar projects requiring less training sessions. Knowledge retention scores, which track the percentage of learned policies that can be applied to other projects, yielded an average of 80.3%, reflecting a high level of generalization of those learned testing strategies across projects. Project A|B—847 cycles (fastest adaption time); Project C|E—1,154 cycles (most difficult transfer) These findings confirm the framework's versatility and indicate that transfer learning methods could enable faster deployment.

6. Discussion

The results reveal the high synergistic added-value brought by the combined heuristic (i.e., adaptive working), thereby suggesting that the reinforcement learning with adaptive software testing components together achieves significant gains on several performance aspects. The scaling-up and empirical validation of state-of-the-art testing techniques scores 44.7% on average of relative improvement over the baseline testing body of evidence in its ability to discover faults, which is transformative by addressing the essential testing problem of maximally efficient defect discovery (Gunda et al. 2025). It happens because the RL agent is able to learn good strategies for prioritizing test cases, so that resources are concentrated on software modules with the highest risk, instead of following a pre-established order of test cases. For critical defects, the fault anticipation component delivered especially stellar numbers (precision = 92.4%), allowing QA teams to proactively test before high-severity faults appear. With this ability to anticipate, the focus of testing is no longer on reactive defect discovery and is instead on proactive risk mitigation, in line with the modern principles of shift-left testing. Critical faults have a long (48.3 hours on average) anticipation lead time, which gives development teams much more time to deal with potential issues before they affect production systems – potentially resulting in a

remediation effort that is an order of magnitude lower in cost than post-deployment fixes. The fact that 94.4% of the performance degradations were addressed by the self-healing feedback mechanism evidences the potential of test optimization in an autonomous way. With static testing strategies, if there is a performance degradation or if the testing strategy becomes sub optimal, manual intervention is required, slowing down the process and needing an expert eye on (Elbaum et al., 2014). Its automated adaptation ability with under 100 milliseconds latency guarantees that the strategy will be optimal at any point in time and no human is required to step in to optimize, making it a highly suitable framework for CI/CD environments with quick feedback loops.

The progression of RL agent's learning is unique to the adaptive testing system and gives us many insights on how these systems tend to convergence. While the systematic improvements in Q-value convergence and optimal policy achievement across 15 000 cycles broadly demonstrate that sufficient exploration of the testing state-action space is essential for effective learning, they also underscore the fact that not all initial testing conditions are created equal. The test results were poorer in the very early phases when the agent was exploring bad strategies, however the exponential reward accumulation in the later phases indicates that the learning investment was worthwhile in the long term. Organizations that choose to adopt such frameworks can expect some initial learning phases before receiving the full suite of benefits. The feature importance analysis demonstrates that many traditional software metrics still play an important role in predictive modeling. Following decades of software engineering research confirming this coupling with defects proneness (McCabe, 1976), cyclomatic complexity is the most heavily influential of all cognitive complexity measures when it comes to fault prediction (p. 10). Yet, this finding highlights the substantial role of temporal data, such as code change frequency, and the need for dynamic cross-sectional metrics that can capture software evolution characteristics. The LSTM network utilizes static code metrics with temporal features, which implementations together comprise a complete fault anticipation model that is superior to using either category independently. This makes the transfer learning results especially relevant to deploying a framework in the real world. An average of 80.3% of knowledge is retained between projects. This implies many learned testing strategies generalize between software, so new projects don’t need training on crucial parts. While not a requirement, organizations may find that RL can enable them to train master agents on representative projects to be transferred onto new developments, allowing adaptive testing capabilities to be deployed quicker. Still, somehow the differences in requirements of the adaptation cycle proof that fine-tuning on a project basis is not redundant, especially in software with prescribed architectural features.

A comparison of these results with prior literature highlights a number of areas of improvement. While Spieker et al. (2017) proposed RL applications to test case selection and although they used purposively selection, they did not include consideration of predictive anticipation or self-heal feature. This integrated framework combines several of the adaptive capabilities described previously, opening the possibility for a larger autonomous testing ecosystem. Likewise, machine learning models showed high levels of baseline performance in previous defect prediction studies (Gunda, 2024a; Gunda, 2024c), but to the best of our knowledge, utilizing predictions in continuous feedback cycles for adapting testing in real-time poses as a new contribution. The average reduction of testing time is 38.7%, so the efficiency problem is critical for software development, because testing is often the bottleneck of deployment pipeline (Chen et al., 2023). This efficiency gain is enabled by an intelligent selection of high-value tests and (thus) deprioritization of redundant or low-value executions. Such framework is economically attractive from the perspective of adoption, which is primarily driven by testing infrastructure, people efficiency improvements of at least 37.

This study has some limitations which should be taken into account. The experimental projects are not the single focus; although they are diverse, they only cover open-source software domains. Industrial software developed on proprietary codebases and using unique practices may have different manifestations affecting framework performance. The second argument is that six months is long enough to observe significant learning and performance, but it may not be long enough to capture dynamics, or stability in performance, over the multi-year scenarios that are common in operational deployments. Third, the fault classification was based on existing bug tracking systems, which might be an incomplete or inconsistent representation of actual defects. Quite frankly, the rates of false positives were acceptably low, meaning we need to improve again on the accuracy of anticipating faults. An overall false positive rate of 7.8% means that resources are sometimes spent testing for predicted faults that do not actually occur, which is waste. Future improvements might add uncertainty quantification mechanisms that respond with intervals for predictions, allowing the user to make better prioritization decisions based on their confidence on the prediction. Because tests perform differently across the projects, it appears that some characteristics of the software being tested impact the effectiveness of adaptive testing. The improvement of 47.7% (R-value of 0.477 significant at 0.001 level) over Project C's 48.4% improvement suggests that potential for adaptation may not be a function of how mature an initial test suite is especially given the wide range of baseline fault detection rates. Future work could determine which software characteristics provide the strongest relationship with the benefits from an adaptive test, thereby helping organizations select projects that are most suited to RL-based approaches.

7. Conclusion

The work demonstrates that reinforcement learning is capable of acquiring adaptive testing policies and provides strong evidence that adaptive software testing can lead to autonomous, self-optimizing testing frameworks. Our proposed architecture which integrates Q-learning agents, LSTM-based fault prediction and self-healing feedback mechanisms shows significant advancements from generic testing approaches in terms of fault detection rates, testing efficiencies and resource utilization. This validation through five projects strengthens the generality and practical applicability of the framework for a wide variety of software development contexts. By identifying high-severity issues boring at 92.4% prediction accuracy ahead of time, the fault prediction part supports proactive testing techniques that catch mission-critical defects before they are deployed to production, thus fundamentally improving the software QA processes. Due to the autonomous adaptation capabilities of the components of the self-healing mechanism where manual testing optimization is not needed, this framework can be beneficial in continuous integration and deployment environments [37]. Directions for future research involve generalizing the framework by adapting other machine learning paradigms, using long-term experience in real-world industrial systems to validate performance, and developing transfer learning optimization techniques with the goal of faster deployment of the framework in different projects. The testing effectiveness and efficiency improvements shown in this work set the idea of RL-driven adaptive testing as a novel paradigm for what self-adaptive software (SAS) could do for autonomous software quality assurance.

References

[1] Chen, T. Y., Kuo, F. C., Merkel, R. G., & Tse, T. H. (2023). Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1), 60-66. https://doi.org/10.1016/j.jss.2009.02.022

[2] S. K. Gunda, "Software Defect Prediction Using Advanced Ensemble Techniques: A Focus on Boosting and Voting Method," 2024 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2024, pp. 157-161, https://doi.org/10.1109/ICESIC61777.2024.10846550

[3] Elbaum, S., Rothermel, G., & Penix, J. (2014). Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 235-245). ACM. https://doi.org/10.1145/2635868.2635910

[4] S. R. Gudi, "Ensuring Secure and Compliant Fax Communication: Anomaly Detection and Encryption Strategies for Data in Transit," 2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 786-791, https://doi.org/10.1109/ICIMIA67127.2025.11200537.

[5] Mahmud, S., Iqbal, M. Z., & Ullah, Z. (2021). Reinforcement learning based adaptive test case prioritization for continuous integration. *IEEE Access*, 9, 117208-117223. https://doi.org/10.1109/ACCESS.2021.3106715

[6] Gunda, S.K. (2026). A Hybrid Deep Learning Model for Software Fault Prediction Using CNN, LSTM, and Dense Layers. In: Bakaev, M., et al. Internet and Modern Society. IMS 2025. Communications in Computer and Information Science, vol 2672. Springer, Cham. https://doi.org/10.1007/978-3-032-05144-8_21

[7] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), 308-320. https://doi.org/10.1109/TSE.1976.233837

[8] Srikanth Reddy Gudi. (2025). A Comparative Analysis of Pivotal Cloud Foundry and OpenShift Cloud Platforms. The American Journal of Applied Sciences, 7(07), 20–29. https://doi.org/10.37547/tajas/Volume07Issue07-03

[9] I. Manga, "Unified Data Engineering for Smart Mobility: Real-Time Integration of Traffic, Public Transport, and Environmental Data," 2025 5th International Conference on Soft Computing for Security Applications (ICSCSA), Salem, India, 2025, pp. 1348-1353, doi: 10.1109/ICSCSA66339.2025.11170800.

[10] S. K. Gunda, "Automatic Software Vulnerabilty Detection Using Code Metrics and Feature Extraction," 2025 2nd International Conference On Multidisciplinary Research and Innovations in Engineering (MRIE), Gurugram, India, 2025, pp. 115-120, https://doi.org/10.1109/MRIE66930.2025.11156601.

[11] Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.

[12] Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: A systematic literature review. *Empirical Software Engineering*, 27(2), 29. https://doi.org/10.1007/s10664-021-10066-6

[13] S. K. Gunda, "Enhancing Software Fault Prediction with Machine Learning: A Comparative Study on the PC1 Dataset," 2024 Global Conference on Communications and Information Technologies (GCCIT), BANGALORE, India, 2024, pp. 1-4, https://doi.org/10.1109/GCCIT63234.2024.10862351

[14] Psaier, H., & Dustdar, S. (2011). A survey on self-healing systems: Approaches and systems. *Computing*, 91(1), 43-73. https://doi.org/10.1007/s00607-010-0107-y

[15] I. Manga, "AutoML for All: Democratizing Machine Learning Model Building with Minimal Code Interfaces," *2025 3rd International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, Erode, India, 2025, pp. 347-352, doi: 10.1109/ICSCDS65426.2025.11167529.

[16] Gunda, S. K., Yalamati, S., Gudi, S. R., Manga, I., & Aleti, A. K. (2025). Scalable and adaptive machine learning models for early software fault prediction in agile development: Enhancing software reliability and sprint

planning efficiency. International Journal of Applied Mathematics, 38(2s). https://doi.org/10.12732/ijam.v38i2s.74

[17] Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 12-22). ACM. https://doi.org/10.1145/3092703.3092709

[18] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[19] S. K. Gunda, "Comparative Analysis of Machine Learning Models for Software Defect Prediction," 2024 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), Chennai, India, 2024, pp. 1-6, https://doi.org/10.1109/ICPECTS62210.2024.10780167.

[20] Wang, S., Liu, T., & Tan, L. (2018). Automatically learning semantic features for defect prediction. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 297-308). ACM. https://doi.org/10.1145/2884781.2884804

[21] S. R. Gudi, "Monitoring and Deployment Optimization in Cloud-Native Systems: A Comparative Study Using OpenShift and Helm," 2025 4th International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Tirupur, India, 2025, pp. 792-797, https://doi.org/10.1109/ICIMIA67127.2025.11200594.

[22] White, M., Tufano, M., Vendome, C., & Poshyvanyk, D. (2019). Deep learning code fragments for code clone detection. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (pp. 87-98). ACM. https://doi.org/10.1145/2970276.2970326

[23] S. K. Gunda, "Analyzing Machine Learning Techniques for Software Defect Prediction: A Comprehensive Performance Comparison," 2024 Asian Conference on Intelligent Technologies (ACOIT), KOLAR, India, 2024, pp. 1-5, https://doi.org/10.1109/ACOIT62457.2024.10939610

[24] Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2016). A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8), 707-740. https://doi.org/10.1109/TSE.2016.2521368

[25] S. R. Gudi, "Deconstructing Monoliths: A Fault-Aware Transition to Microservices with Gateway Optimization using Spring Cloud," 2025 6th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2025, pp. 815-820, https://doi.org/10.1109/ICESC65114.2025.11212326

[26] I. Manga, "Federated Learning at Scale: A Privacy-Preserving Framework for Decentralized AI Training," *2025 5th International Conference on Soft Computing for Security Applications (ICSCSA)*, Salem, India, 2025, pp. 110-115, doi: 10.1109/ICSCSA66339.2025.11170780.