A HYBRID MODEL USING CLUSTERING AND REINFORCEMENT LEARNING FOR TEST CASE PRIORITIZATION IN AGILE ENVIRONMENTS

M. S. Nadeem[a,*] M. A. Farooq[b,*] and S. Afzal[c,*]

[a]Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan [b]Institute of Data Science, University of Engineering and Technology, Lahore, Pakistan
[c]Department of Computer Science, University of Engineering and Technology, Lahore, Pakistan *Corresponding authors. These authors contributed equally to this work

ABSTRACT: Agile development involves rapid and dynamic regression testing to ensure that it keeps pace with the frequent code changes and continuous integration. The traditional methods of test case prioritization (TCP) rely on static historical data and thus cannot handle noisy logs, changing test behavior, and inconsistent failure modes typical of Agile systems. In this paper, the authors describe a hybrid approach to prioritization based on the combination of DBSCAN clustering and Q-learning as the means of overcoming the two challenges, namely the quality of data and adaptability. DBSCAN groups similar test cases and removes noise and Q-learning is used to train an execution order in each cluster using a reward function balancing early fault detection and cost of execution. A mixed dataset of approximately 15,000 test executions of both Defects4J and real CI pipeline are used to test the model. Findings indicate that the given strategy attains an accuracy of 90.5 percent, APFD of 0.87, and a decrease in the overall testing time of 28 percent, which outperform the Random, Greedy, and DeepOrder baselines. It is lightweight and scalable with an easy deployment into CI/CD pipelines, which makes it highly applicable in the contemporary Agile testing process.

Keywords: Test Case Prioritization, Agile Software Development, DBSCAN Clustering, Reinforcement Learning, Predictive Analytics, CI/CD.

(Received 01.10.2025 Accepted 01.12.2025)

INTRODUCTION

The use of agile software development methodologies is the main foundation of contemporary software development and offers an adaptable and iterative approach to delivering as well as developing software systems. The experience of Extreme Programming (XP), Scrum, and Kanban allow development teams to adapt to changing customer needs, streamline the process of creating the product, and sustain the feedback with the help of the software development life cycle. Although Agile approaches are versatile, they do pose greater problems in terms of quality assurance where regression testing is concerned. The regular code changes, the frequent integration and deployment (CI/CD), and the short development cycles put an increased burden on the testing cycle, and the regression testing is one of the most time-consuming parts of the Agile workflow.

Among these challenges, test case prioritization is one of the major problem of Agile environment. This study aligns with the emerging trend of integrating hybrid machine learning and reinforcement techniques into Agile test optimization. The process of determining the optimum sequence for executing test cases to identify defects early while minimizing the usage of time and resources. However, traditional TCP methods of testing case prioritization, such as random ordering, rule-of-thumb code coverage, or using historical test results, are ineffective in the Agile environment. These methods tend to be founded on fixed data, such as the failures that have been experienced in the past or the opinion of experts, and do not scale with the evolving lifetime of systems, time-varying trends, and new tests.

These fixed methods, in turn, result in unnecessary testing, long-term fault detection, increased testing time, and the revelation of serious quality flaws in the production process.

Agile projects further complicate TCP due to incomplete or noisy test logs, flaky test results, and inconsistent execution patterns. These issues undermine the performance of static test prioritization methods. In order to overcome these drawbacks, recent studies are considering predictive analytics and machine learning (ML) as smart, data-driven methodologies of test case prioritization. Predictive analytics models are trained on historical data, code changes, defect trends, and other test metrics to identify the patterns and estimate the possibility of defects for a particular test scenario. Techniques such as gradient boosting, deep neural networks, and random forests have shown improvement in fault detection efficiency by learning fault-relevant patterns dynamically. Nevertheless, these models typically assume clean, labeled data and are usually

trained offline, thus did not perform well in Agile environments where data quality is not always high and adjustments to decision making needed to be possible in real time.

Although ML-based models follow supervised learning paradigms requiring extensive labeled data, this is often unavailable for Agile projects. They also lack adaptability to real-time project changes, limiting their scalability and responsiveness. Thus, there still exists a dire need to have a smart TCP that is flexible, robust against the associated noise, and scalable over dynamic project conditions. Given these limitations, there was a need for a model that can learn adaptively from noisy, dynamic data and optimize test case selection in real time.

To address these challenges, this study proposes a hybrid test case prioritization framework that deals with the dual problem of data quality and adaptability in Agile TCP, combining unsupervised clustering (DBSCAN) and reinforcement learning (Q-learning). The proposed model integrates clustering-based data preprocessing with adaptive reinforcement learning to achieve both robustness and adaptability. As illustrated in Figure 1, the workflow of the proposed hybrid model integrates clustering and reinforcement learning for test case prioritization.

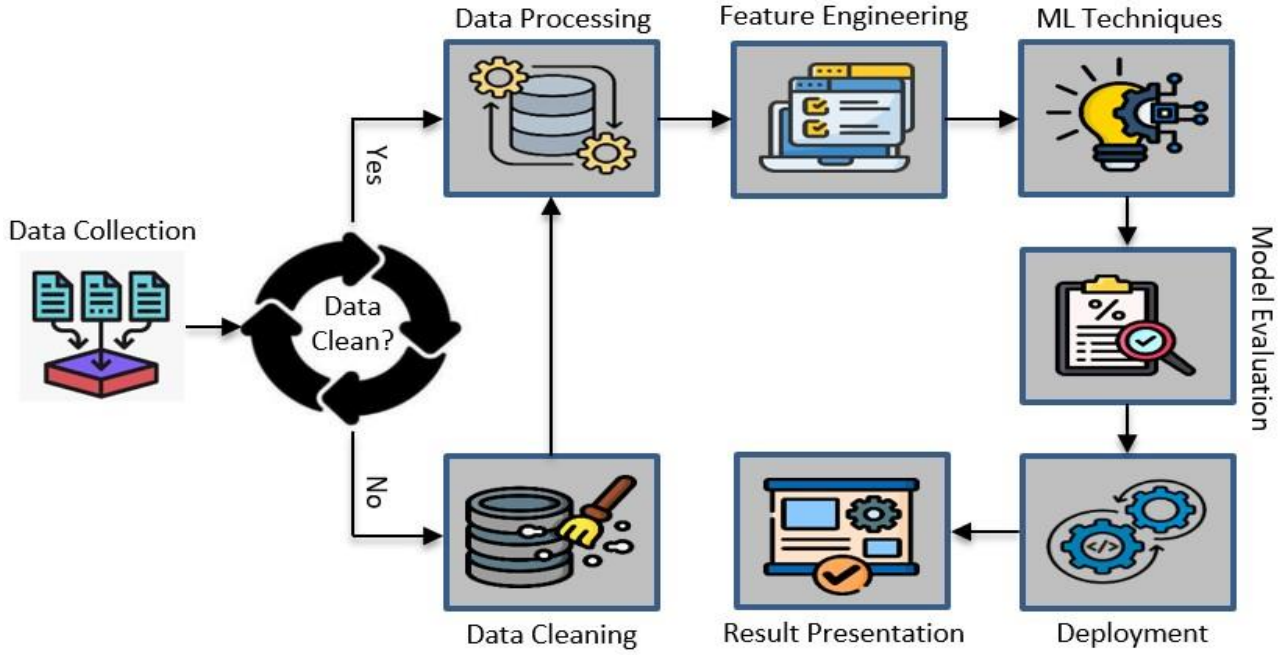

Figure 1: Working Principle of the Hybrid Model for Test Case Prioritization

Figure 1 illustrates the complete working process of the proposed hybrid model for test case prioritization. The workflow starts with data gathering of Agile projects which comprises of test cases execution logs, code change, defect reports and historical outcomes. The preprocessing is then used to clean up noisy or incomplete data. Analysis feature engineering identifies such meaningful measures like code churn, defect rate, and pass/fail history.

DBSCAN clustering is then used to consider similar test cases as dense clusters and outliers, i.e. flaky tests. This step makes sure that the input is structured and high quality such that it can be prioritized without being labeled manually. After clustering has been done, the Q-learning agent continuously engages with the testing environment. In every step, it picks a test case out of the cluster, gets feedback on the cost of detecting defect and cost of executing the test cases and revise its decision policy. In several episodes, the agent is informed of a good prioritization order which results in early fault detection and the minimum time taken to execute. Lastly, the evaluation of prioritized results is done through accuracy, APFD, NAPFD, and reduction of execution time, and the product can be directly integrated into CI/CD pipelines to be used in real-time.

This research aim as follows.

- Create a hybrid ML framework of dynamic TCP in an Agile environment.
- Use multi-feature data (code changes, defect history, execution metrics) for strong test case prioritization.
- Measure performance with accuracy, APFD, NAPFD and reduction of the time used in testing.
- Make scaling and Agile CI/CD pipeline integration possible.

By proposing this hybrid solution, the study contributes to the field in several ways:

- Proposed a hybrid model by connecting DBSCAN clustering and Q-learning for test case prioritization.
- Processed noisy and inconsistent test data using density-based clustering techniques.
- Select dynamic test cases through reinforcement learning with a reward-driven strategy.
- Use multi-feature inputs to improve fault prediction accuracy and adaptability.
- Obtained high accuracy, APFD, and test time reduction over baseline methods.
- Tested the model on large-scale Agile and open-source benchmark data.
- Developed the framework for real-time integration into CI/CD pipelines

This study is significant because it fills a critical gap between the predictive techniques that are static and the dynamic and real-world Agile testing requirements. Although previous studies have either concentrated on noise handling through clustering or prioritization through reinforcement learning, this study is the first to integrate both to produce a synergistic effect. With the ongoing scaling of Agile development practices across industries, the necessity of smart and responsive TCP approaches is becoming more and more critical not only to enhance the quality of software but also to ensure the speed of delivery and customer satisfaction in the rapid development cycles.

This research is designed as follows: Section 2 summarizes prior studies with examine current TCP limitations for Agile environments. Section 3 presents the proposal of the hybrid framework structure and realization. Section 4 outlines the arrangement of experiments, the sources of data, and the set of criteria. Section 5 will show the results and discuss them in detail. Finally, Section 6 concludes the paper with summaries of extended research avenues, i.e., the inclusion of Natural Language Processing (NLP) to derive test intent and to fine-tune reward functions.

Related Work: Test Case Prioritization (TCP) aims to order test cases so that faults are detected as early as possible, reducing feedback time and resource consumption in regression testing, as discussed by Yoo et al. [1]. In Agile environments, frequent commits, rapid iteration, and continuous integration significantly increase the need for effective TCP, because test outcomes, coverage patterns, and code behaviors change quickly as described by Zhang et al. [2]. The demands of contemporary CI/CD pipelines increase these requirements further with the need of fast, automated, and flexible prioritization approaches that can keep up with the development speed as reported by Sami et al. [3]. Traditional methods of prioritizing using static methods do not work well in these conditions because they rely on a set of historical data and historical heuristics that Sharif et al.[4] define as deeporder .

Recent studies have paid attention to machine-learning-TCP that makes use of historical data to enhance previous decisions during prioritization and respond to the varying development conditions as Ajorloo et al. described them in their study of machine-learning-TCP in the paper of 2024 [5]. To exploit synergies of several techniques, more recently hybrid methods are suggested, especially clustering to reduce noise and reinforcement learning to provide flexibility a more scalable and robust alternative in dynamic settings as described by Fokrul et al. [6]. This development is indicative of a larger movement to smart, data-driven TCP options that can be used in the Agile and CI/CD processes according to Cheng et al. [7].

Challenges of Traditional TCP in Agile Environments: The coverage-based heuristics, previous fault-detection-based orderings, or predetermined rule-based prioritization methods of traditional TCP remain developed based on stable plan-driven development processes as described by Arshad et al. [8]. Traditional TCP methods coverage-based, previous faultdetection-based ordering, or pre-established rule-based prioritization were originally designed to use in a stable plan-driven development process. These methods are efficient to identify faults when conditions remain consistent, although when sprint cycles are small and changes are continuous, they are ineffective as well as fault detectors are concerned as explained by Mamata et al. [9]. Since Agile processes are subject to continuous adaptation, fixed strategies with recourse to historical measurements usually become obsolete at every iteration of the process as explained by Tawosi et al. [10]. A significant issue is brought about by the fact that historical testing data and the present state of the system will not correlate the code is constantly evolving, and past test results and coverage patterns will become very unreliable in the present case, therefore, they are less likely to succeed in test driving the present code as described by Elbaum et al. [11]. A test case that is a priority in one sprint can be outdated by the next because of new features or refactoring as discussed by Rodr´ıguez S´anchez et al. [12]. This old-fashioned priority may make it take long to identify defects, perform unnecessary test, and diminish the testing efficiency as explained by Ajorloo et al. [5].

Agile environments also introduce data-quality issues such as flaky tests, incomplete logs, and inconsistent execution behavior as explained by Siddique et al. [13]. Factors like flaky tests, partial logging, and rapid iteration cycles reduce the reliability of historical data that most of the traditional metrics are based as described by Birchler et al. [14]. The data quality

problems such as models that are based on predefined heuristics will not identify the most critical test scenarios properly, and leave untested faults, along with a lower level of testing process efficiency, as discussed by Rajasingh et al. [15]. CI/CD pipelines increase these challenges, as test suites may execute multiple times per day with limited time for computationally expensive prioritization methods as shown by Chen et al. [16]. Traditional techniques lack the scalability and responsiveness required for fast-moving agile environments as described by Zhang et al. [17].

Empirical studies show the limitations of static TCP in dynamic Agile contexts. Performance inconsistencies arise from unstable test results and shifting defect patterns as explained by Omri and Sinz theorize [18]. Although clustering methods like DBSCAN can group similar tests effectively, they cannot adapt to changing project conditions without combining adaptive algorithms like reinforcement learning as discussed by Chen et al. [19]. These findings point to the importance of hybrid approaches that balance noise handling, scalability, and real-time adaptability.

TCP using Machine Learning: The field of machine learning (ML) has made it into a highly discussed area in order to overcome the shortcomings of the conventional TCP methods. ML-based methods utilize the historical execution process data and the dynamic software metrics, to make more adaptive and informed decisions on prioritization decisions in advance as presented by Felding et al. [20]. The usefulness of ML in test case prioritization was also mentioned by Singhal [21]. Unlike static approaches, ML models can learn evolving patterns in test outcomes, fault severity, code churn, and defect density, enabling them to respond more effectively to continuous change in Agile environments as outlined by Mahdieh et al.
[22].

TCP has a number of ML paradigms applied to it. The supervised learning models are based on labelled execution histories to forecast high-risk or fault-prone test cases. Reinforcement learning (RL) views prioritization as a sequence decision-making, where one learns an optimal ordering policy based on the feedback of performed tests as discussed by Pan et al. [23]. Clustering methods cluster similar tests together and they share some similarity which allows noise to be reduced and it also helps to manage the large test suites in a more efficient way as argued by Li et al. [24]. Together these strategies uphold the data-driven and adaptive manner of the Agile development according to P.K. Gupta [25].

Supervised Learning Approaches: Application Supervised machine learning methods have been extensively considered in TCP as they can be trained to learn prioritization trends based on past test executions of the tests as introduced by Felding et al. [26]. Common features of these methods are past fault detection, execution time, test complexity and code coverage to rank test cases as a more effective ranking methodology as explained by Bugayenko et al. [27]. A prominent example is DeepOrder using deep neural networks to rank tests cases in a sequence according to past execution results and can increase APFDs by up to 40 per cent on industrial data sets as put across by Bajaj and Sangwan [28]. Nevertheless, approaches such as DeepOrder can be unsuccessful in quickly changing Agile settings because they require consistent, collected training data and are not so much flexible to changing test characteristics as explained by by Khatibsyarbini et al. [29].

Other monitored models-such as Gradient Boosting and Random Forest-have also been used on TCP, using several test-level and code-level characteristics to predict the possibility of fault-detection potential with the use of multiple test-level and code-level characteristics as discussed by Chen et al. [19]. These models were investigated by Tiutin as well [30]. Even though such models are effective under constant conditions, they usually need large, clean and well-labeled datasets. This is a problematic requirement in Agile environments where test data can often be too noisy, too incomplete, or too old-fashioned as discussed by Gokilavani and Bharathi [31]. This difficulty was also mentioned by Rosenbauer [32]. Consequently, pure supervised methods might not be able to generalize in dynamic CI/CD settings.

Reinforcement Learning (RL) for TCP: Reinforcement Learning (RL) has become an attractive alternative to supervised learning to TCP especially on Agile and CI/CD setting where changes in system behavior occur quickly at any given time of the day and night as suggested by Su et al. [33]. In contrast to supervised approaches, RL does not use labeled data; rather, it can learn prioritization policies through interaction with the environment and by getting reward signals, such as early fault detection, less execution time, or better resource utilization, as explained by Bagherzadeh et al. [34]. Such a feedback-guided learning allows RL agents to dynamically transform their decision-making strategies as the Agile development is very dynamic and constantly changing in nature as explained by Mirzaei and Keyvanpour [35].

Recent experiments show that RL is effective in the dynamic adaption of prioritization. Indicatively, Bagherzadeh (2024) found that the rate of fault detection improved by up to 50% in the CI pipelines, which underscores the capability of RL to revise its policy depending on the real-time execution patterns as found by Yaraghi et al. [36]. Further enhancement such as time-windowed reward functions enhance further RL responsiveness to recent changes and fault trend variation as suggested by Li et al. [37].

Despite these advantages, RL has problems with large-scale test suites. Real-time applicability can be hampered by exploration costs, low convergence rates, and the necessity of many iterations to explore the phenomena and facilitate their future implementation and application in real-time scenarios as explained by Han et al. [38]. Incorporating RL into CI/CD pipelines can also be engineered with lots of attention without adding delays or computational overhead that could cause a disruption of quick build cycles as explained by Pandhare et al. [39]. These drawbacks have led to the desire hybrid methods that combine RL with techniques like clustering to improve scalability.

Test Case Prioritization Clustering Techniques: TCP extensively uses clustering methods especially the unsupervised learning methods to maintain large and diversified test repositories. Such algorithms as the DBSCAN and K-Means cluster test cases based on their features (e.g., execution time, failure rate, and similarity of the code) and limit the decision space, allowing more efficient prioritization processes to be carried out, a feature of algorithms that are not new but has only recently been embraced in research as outlined by Vescan et al. [40].

DBSCAN, specifically, works well with test data since it can find clusters and outliers without specifying the number of clusters involved and irregular data distribution can also be dealt with.

Hybrid types of clustering have been examined with the aim of optimizing the performance of TCP. Indicatively, Zhang and Chen (2023) integrated DBSCAN with Firefly Optimization to rank tests within clusters, and showed better APFD results with the help of reinforcers as explained by Bagherzadeh et al. [34]. Nevertheless, these approaches are mostly stagnant and cannot keep up with the constant changes prevalent in Agile setting as explained by Wang and Zhang [41]. To address this issue, Cluster-based Adaptive Prioritization (CAP) strategies dynamically update cluster boundaries and test priorities according to the latest test results and code changes as explained by Vishwanath Karad et al. [42]. Nevertheless, the majority of clustering-based TCP models do not include the learning mechanisms such as RL and can therefore not fine-tune their decisions using feedback as it appears as described by Pan et al. [23].

Although clustering is useful in sorting large test suites and in reducing redundancy, the fact that it is a static concept restricts its adaptability to rapid Agile workflows as noted by Gupta and Mahapatra [43]. Real-time decision-making can not be done in pure clustering methods which cannot dynamically respond to changing fault patterns in real-time. All these limitations have given rise to an increased desire to use hybrid methods and introduce clustering with adaptive learning methods to provide stability and responsiveness in the prioritization process together with stability and responsiveness in the prioritization process as outlined by Ahmad et al. [44].

Hybrid Approaches: Combining Strengths: Clusters increased with reinforcement learning (RL) have been popular in TCP study, especially in Agile frameworks where scalability and flexibility must be ensured by Singh et al. [45]. Clustering can be used to manage the large test suites by grouping like test cases and RL can be used to make adaptive decisions using real-time feedback and long-term performance objectives as per Vescan et al. [40].

A popular design is a hierarchical two stage process where clustering is followed by prioritization space reduction and a RL agent is used to decide the sequence to be followed in each cluster. As an example, Berisha (2024) and Prado Lima et al. [46] applied clustering to create homogeneous groups of test cases and further prioritized them using RL, which was applied intra-cluster. This architecture decreases the computational load and enables the prioritization policy to adapt using the measured fault detection and execution time measurements in sight of the data acquired during execution and monitoring of faults as described by Vescan et al. [47]. Such hybrid models have been reported to lead to better fault detection and CI/CD performance, based on empirical research, and have demonstrated higher efficiency in these fields of study as described by Chen et al. [48].

Based on this paradigm, the hybrid model adopted in the present paper applies DBSCAN to find natural clusters of test cases based on such features of code churn, historical defects, and runtime data as code churn, historical defects, and runtime data. In comparison to K-Means, DBSCAN does not demand specification of the number of clusters and is more efficient at noise and outliers-common to Agile test data as explained by Tiutin and Vescan [30]. Once clustering is completed, a Q-learning agent ranks test cases in a cluster by a rewarding function that is consistent with the Agile objectives: the prompt detection of faults, less time to execute, and adjustment to changing defect behavior. This combination offers the benefit of scaling with clustering, noise tolerance via DBSCAN, and adaptability with continuous learning through RL. The approach is appropriate to be applied to Agile workflows where quick, adaptable, and contextual TCP is required due to the localized decision-making process and the availability of real-time feedback as outlined Li et al. [49].

Tools, Datasets, and Benchmarks: Benchmarking TCP techniques has been performed on a number of datasets and tools, with the most common ones being Defects4J, LRTS, and Apache test suites described by Zhang et al. [50]. The drawbacks of Defects4J are especially common because it has repeatable faults and detailed execution

logs as explained by Han et al. [38]. Nevertheless, its comparatively rigid structure does not make it applicable to Agile environments, where codebases and test behavior change at a high pace. More general discussions of TCP-based methodologies point to the fact that most currently available data do not cover Agile dynamics sufficiently well described by Khatibsyarbini et al. [29].

Some researcher model development pipelines based on synthetic data or mine execution traces of CI systems like Jenkins to better model real-world behavior in development pipelines, which are executed in practice as proposed by Vescan et al. [51]. These sources have more realistic presentations of the variating test results, code fluctuations, and noisy logs that reflect closely well the actual Agile workflows.

A similar weakness relates to evaluation measures. The majority of TCP research is based on APFD or NAPFD, which does not adequately capture Agile principles, including time and scalability and real-time responsiveness as projected by Rosenbauer et al. [32]. Recent research suggests the development of more rigorous evaluation criteria, including capture of reduction in execution time, responsiveness, robustness in the presence of dynamic conditions as stated by Li et al. [49]. This wider range of metrics is proposed in the given model that tries to offer a more realistic and Agile-friendly measure of TCP performance as reported by Yaraghi et al. [52].

Research Gaps and Motivation: Although there has been progress in the domain of ML and RL-driven TCP, there are a number of gaps to be filled when implementing these techniques in Agile settings. Scalability still remains a significant issue with RL models usually needing a lot of training and potentially not converging fast enough to allow quick development cycles to occur on a model with a high level of scalability at the same time as describes by Yaraghi et al. [52]. Another urgent concern is data quality: numerous models expect clean, labeled data, but in Agile pipelines, test results are often noisy, incomplete, or inconsistent in most cases as described by Cheng et al. [53].

Many of the current approaches are also restricted in terms of adaptability. The traditional or semistatic models in most cases do not modify the prioritization techniques based on the changing defect patterns, changing codebases or changing patterns of execution in most circumstances, such as test executions, code-written patterns, and so forth as described by Iqbal and Al-Azzoni [54]. Moreover, real-world integration with CI/CD pipelines is often not considered; only several models can be used with lightweight and low-overhead deployment that is critical in automated processes when feedback and low latency are the key factors to consider as described by Geetha et al. [55]. This requirement was also highlighted by Tasneem [56]. Such gaps drive the necessity to have hybrid, adaptive, and noise-tolerant TCP solutions that can be successfully used in current Agile and CI/CD environments.

Proposed Methodology: The proposed hybrid model is a variant of DBSCAN clustering that is integrated with the Q-learning reinforcement learning addressing the issues of Agile regression testing, such as noisy data, flaky tests, and fast-changing codebases. The pipeline is based on six stages: Data Collection, Preprocessing, Feature Selection, Machine Learning (DBSCAN + Q-Learning), Model Evaluation and Deployment (Figure 6). This workflow guarantees the scalability and flexibility as well as the openness to CI/CD platforms, including Jenkins, Azure Devops, Docker, and Selenium.

After data collection and preprocessing, such as removing duplicates, normalizing and treating flaky tests, are completed, then the relevant features to include in the model training are chosen. DBSCAN clusters group test cases based on structural and behavioral similarities, considering factors like past failures, code changes, and execution patterns. Its density-based methodology removes noise and finds clusters without specifying the number of groups, and thus is appropriate to heterogeneous Agile data.

In each cluster, Q-learning was used to establish the best execution sequence by maximizing fault detection (quantified through APFD) and minimizing the cost of execution. DBSCAN with Q-learning allows its application in reducing noise, adaptive prioritization, and effective early fault detection on time.

Finally, the system is implemented within CI/CD pipelines, where it constantly updates clusters and recalculates prioritization policies using real-time feedback. This guaranties continued flexibility in modification of Agile development and enhances accuracy, APFD, and total efficiency in regression testing.

Overview of the Hybrid Model: The entire operational workflow of the proposed method is shown in figure 2. It begins by gathering heterogeneous Agile test data, and then preprocessing (normalization, deduplication, flaky-test detection). The features that are selected are sent to DBSCAN to remove noise and cluster, and the Q-learning algorithm is used to rank the tests within a cluster. The model is updated by a feedback loop that reacts on evaluation metrics (APFD, NAPFD, accuracy, and execution time), making it possible to constantly adapt it.

Dataset Collection: Open-source software repositories and industrial Agile CI/CD environments, such as Defects4J, Apache Commons, DeepOrder GitHub CI logs, and simulated Agile pipelines were used to gather data. The combined data set has about 15,000 test runs

with commit data, historical failure data, run time, code churn data, levels of defect severity, and test logs.

Table 1: Comparative Analysis of Recent Studies on Predictive Analytics for TCP in Agile Software Development.

Author	Dataset	Predictive Analytics	ML/AI	Method Focus on TCP	Agile/CI	Adaptability	Scalability
Yoo et al. [1]	Literature survey	Yes	No	Yes	No	No	
Zhang et al. [2]	Open-source logs (DBSCAN)	No	Yes	Yes	No	No	
Sami et al. [3]	OSS test suites	No	Yes	Yes	Yes	No	
Ajorloo et al. [5]	Agile project data	Yes	Yes	No	Yes	No	
Cheng et al. [7]	Long-running industrial test suites	Yes	Yes	Yes	Yes	No	
Arshad et al.[8]	Industrial defect data	Yes	Yes	No	Yes	No	
Tawosi et al. [10]	Survey data	Yes	No	No	Yes	No	
Rajasingh et al. [15]	OSS fault detection datasets	No	Yes	Yes	No	No	
Chen et al. [16]	Historical execution info	No	Yes	Yes	No	No	
Felding et al. [20]	Synthetic datasets	No	Yes	Yes	Yes	Yes	
Pan et al. [23]	CI log data	No	Yes	Yes	No	No	
Bugayenko et al. [27]	OSS task datasets	Yes	Yes	No	Yes	No	
Bajaj et al. [28]	Synthetic test cases	No	Yes	Yes	No	No	
Felding et al. [26]	OSS test data	No	Yes	Yes	Yes	No	
Khatibsyarbini et al. [29]	Literature review	Yes	Yes	Yes	No	No	
Bagherzadeh et al. [34]	Historical logs	No	Yes	Yes	Yes	No	
Gupta et al. [43]	Unit + integration tests	No	Yes	Yes	No	No	
Bagherzadeh et al. [52]	Open-source test suites	No	Yes	Yes	No	No	
Vescan et al. [40]	Multiobjective test suites	No	Yes	Yes	Yes	No	
Prado Lima et al. [46]	Configurable CI data	Yes	Yes	Yes	Yes	No	
Z. Zhang et al. [53]	Open-source test logs	No	Yes	Yes	No	No	
Han et al. [38]	Reinforcement learning test data	Yes	Yes	Yes	Yes	No	
Vescan et al. [51]	Regression test suites	No	Yes	Yes	No	No	
Li et al. [49]	Semanticaware CI data	No	Yes	Yes	Yes	Yes	
Cheng et al. [53]	Large OSS config test sets	No	Yes	Yes	No	Yes	
Qingran et al. [33]	CI execution logs from open-source systems	No	Yes	Yes	Yes	Yes	
Yaraghi et al. [36]	TARBENCH: 45K tests, 59 projects	No	Yes	No	No	No	
Tasneem et al. [56]	76 studies from 5 databases	Yes	Yes	No	Yes	No	
Pandhare et al. [39]	Synthetic CI tests (Mabl, Launchable)	Yes	Yes	No	Yes	No	

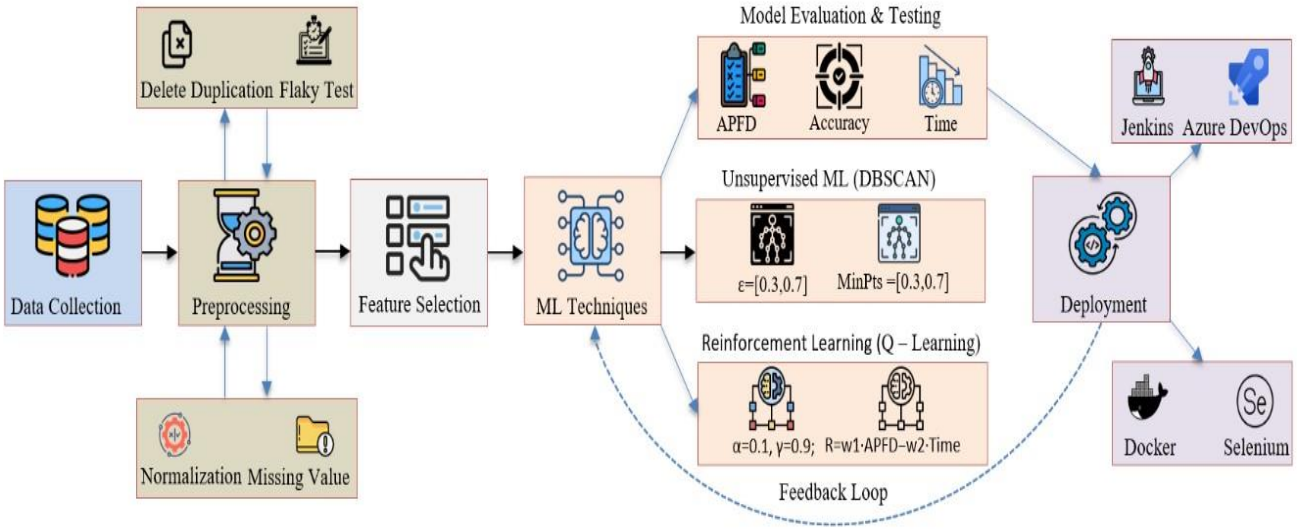

Figure 2: Methodology of Proposed Hybrid Model for Test Case Prioritization.

This mixed data set is also indicative of the diversity and heterogeneity of actual Agile processes and ensures that the hybrid model is trained and tested under real and heterogeneous testing conditions. Table 2 presents a detailed account of all the data sources and the attributes that were recorded.

Table 2: Summary of Datasets and Tools Used

Source	Repository/Tool	Test Cases	Data Collected
DeepOrder	GitHub Java CI Projects	5,000	Historical CI logs, commit history, and test failures
Defects4J Benchmark	Defects4J v2.0	Approx. 3,500	Test cases, real-world bugs, and patch history
Apache Commons	Apache Commons Lang, Math	Approx. 3,000	Bug report, Code revisions and test coverage
Proposed Hybrid Model	Simulated Agile CI Projects	Approx. 2,000	Regression testing, bug-fix data, and code changes

Preprocessing: Agile CI/CD pipelines are known to produce a lot of noisy, inconsistent and incomplete test data because of quick code change, unstable test environment, and parallel execution. Hence, a strict preprocess step was implemented in order to deliver high quality input to both the DBSCAN clustering as well as the Q-learning reinforcement model.

To eliminate redundant data, first, duplicate records (2.4%) were eliminated to avoid an influence of duplicate information on cluster density estimation. The missing values (5.7%), which are typically present in execution time logs and test results, were imputed using mean imputation in the case of numerical attributes and mode imputation in the case of categorical attributes. Also, 3.1 percent of flaky tests-detected by inconsistent pass/fail behavior under the same execution conditions were eliminated to improve data stability and avoid misleading reward signals during training

In order to equalize the heterogeneous numeric features, Min-Max Normalization was used to normalize all the numerical variables within a range [0,1]. This action stabilizes the distance based clustering of DBSCAN and speeds up the convergence of Q-learning because huge features are not allowed to dominate. The normalization can be defined as:

$$X\text{Scaled} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Where X represents the original value of the feature, $X_{\min}$ and $X_{\max}$ is the minimum and maximum value of the feature observed.

Categorical variables (such as defect severity, type of the test, etc) were coded with the one-hot encoding to avoid ordinal bias and be compatible with both the clustering and the reinforcement learning state representations. The complete preprocessing was performed with the help of Pandas, NumPy, and Scikit-learn, so that it can be easily integrated with the reinforcement learning component of TensorFlow.

Feature Selection: In order to match the dynamic needs of Agile, we have determined the most influential features in three areas code changes (e.g., churn, commit frequency), defect history (e.g., severity, recurrence), and execution data (e.g., test runtime, recent failures). Pearson's Correlation Coefficient was used to assess linear relationships, while Recursive Feature Elimination (RFE) iteratively removes low-effect features. Both techniques were implemented using Scikit-learn. The top 10 features were retained to maintain balance across domains and increase the predictive accuracy of the model.

Model Selection: The Machine Learning module integrates DBSCAN clustering with Q-Learning reinforcement learning to form an adaptive two-phase prioritization strategy. DBSCAN first organizes test cases into coherent structural groups, after which Q-Learning optimizes the execution order within each cluster. Algorithm 1 summarizes the complete hybrid workflow, showing how clustering and reinforcement learning interact to produce the final prioritized test suite.

DBSCAN Clustering: The first component of the machine learning module applies DBSCAN to identify groups of structurally and behaviorally similar test cases based on density characteristics. This step corresponds to the unsupervised ML component illustrated in Figure 2. DBSCAN is well suited for Agile regression testing because it does not require a predefined number of clusters and is inherently robust to noise an important advantage when dealing with heterogeneous, rapidly changing test data as described by Ester et al. [57]. Unlike K-Means, DBSCAN does not depend on spherical cluster assumptions or a predefined number of clusters, making it more suitable for irregular and evolving Agile test datasets. There were two important hyper parameters that were tuned through grid search:

$$\varepsilon \in [0.3, 0.7], \qquad \text{MinPts} \in [3, 7]$$

Here, ε is the radius of the neighborhood that is used to define the density connectivity and MinPts is the minimum number of points that are needed to create a dense region. The evaluation of the quality of clustering was done with a Silhouette Score based measure that evaluated the intracluster cohesion and the separation between clusters. There were higher silhouette values, which validated that the selected hyperparameters produced and well structured clusters that can be used to develop downstream reinforcement learning. DBSCAN was provided on Scikit-learn, and Q-learning was provided on TensorFlow.

Q-Learning Based Test Case Prioritization: After the formation of clusters, Q-Learning is used in a cluster to obtain an ideal order of execution of the test cases contained in the cluster. This design is based on the reinforcement-learning TCP approach that Spieker et al. studied in continuous integration environment [58]. Q-Learning is a cluster-level algorithm that minimizes the computational costs and still provides fault-detection capability.

The reward function used by the RL agent to evaluate each action (i.e. which test case to execute next) is as follows the early fault detection and cost of execution are balanced:

$$R = w_1 \cdot APFD - w_2 \cdot Time$$

Where $w_1 = 0.7$ is used to maximise the Average Percentage of Faults Detected (APFD) and $w_2 = 0.3$ is used to discourage longer test execution times. This incentive system encourages the process of focusing on high-impact and low-cost tests, which are closely associated with the Agile hardship demands.

The Q-values are solved through recurrence of the Bellman optimality equation suggested by Watkins and Dayan [59]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

where $\alpha = 0.1$ is the learning rate and $\gamma = 0.9$ is the discount factor. The learning rate determines the speed at which new experience impacts the policy of the agent, whereas the discount factor ensures that the agent is not greedy but optimizes over long-term goals.

This setup was directly equivalent to the Reinforcement Learning (RL) block implemented in Figure 2, in which the combination of DBSCAN-generated clusters, the APFD time reward model, and the $\alpha = 0.1, \gamma = 0.9$ learning rate allows the generation of adaptive, data-driven prioritization of test cases that would be suitable in dynamic Agile CI/CD contexts.

Algorithmic Workflow: The full hybrid Test Case Prioritization (TCP) algorithm presented in algorithm 1 combines the results of DBSCAN clustering along with the adaptive Q-Learning decision-making. Each cluster is processed independently, allowing the model to scale effectively to large Agile test suites while maintaining local adaptability and noise robustness.

Algorithm 1: Hybrid Clustering and Q-Learning Based Test Case Prioritization

Input: Clusters $C_1, C_2, \ldots, C_n$

Output: Final prioritized test suite

Initialize: $\alpha = 0.1$, $\gamma = 0.9$, $w_1 = 0.7$, $w_2 = 0.3$, *episodes* = 100, $\epsilon = 0.1$;

Define:;
State s: current test case and its features;
Action a: select next test case;
Terminal state: all test cases executed or testing budget exhausted;

foreach *cluster* C_i **do**
  Initialize $Q(s,a) \leftarrow 0$;
  for *episode* $\leftarrow 1$ **to** N **do**
    Initializestate s;
    while *notterminal* **do**
      Chooseaction a using ϵ-greedypolicy;
      Executetestcase,measure *APFD* and *Time* ;
      Computereward: $R = w_1 \times APFD - w_2 \times Time$;
      Update $Q(s,a)$ usingtheBellmanequation;
      Set $s \leftarrow s$;
  Ranktestcasesby Q-values;
Mergeprioritizedlistsfromallclusters;
return finalprioritizedtestsuite;

Baseline Methods: To analyze the effectiveness of the suggested hybrid model (DBSCAN + Q-learning) comparing with four widely adopted baselines that represent heuristic, metaheuristic, and machine learning families.

Random Prioritization: A lower-bound baseline that executes test cases in random order. It is included in almost all TCP studies as a minimal reference for early fault detection performance [80], [81]. Its weakness lies in a complete lack of learning or adaptation, resulting in poor fault detection.

Greedy Additional (Additional Coverage): A classical heuristic that iteratively selects the test case providing the maximum additional coverage or fault detection with respect to already executed tests [81]. It is efficient and historically forms the benchmark in regression testing. However, it is fixed and fails to keep up with changing Agile pipelines.

Deep Learning-Based TCP (DeepOrder): A strategy based on learning using historical CI logs to learn neural models to predict the best test sequences. It embodies the current state of the art in ML-based TCP, however, it demands very big training datasets and great computation capabilities which limits the scalability when the resource is scarce.

Proposed Hybrid Model (DBSCAN + Q-learning): In the first stage of our method, we cluster test cases together with the help of DBSCAN and use the Q-learning method in each cluster to select the tests adaptively. The reward function balances the early fault detection (APFD) and the execution time. In comparison with baselines, this hybrid model is scalable, noise resistant and adaptable to Agile CI/CD pipelines.

Table 3: Baseline Comparison Matrix

Method	Type	Strengths	Limitations
Random	Heuristic (baseline)	Simple; provides a lower bound	Very poor detection rate; no adaptation
Greedy Additional	Heuristic	Widely used; efficient for small test suites	Static; not adaptive to Agile dynamics
GA-TCP	Metaheuristic	Finds optimized prioritization sequences	Computationally expensive; slow for CI/CD
DeepOrder (2021)	Deep Learning	Learns from CI logs; strong empirical performance	Requires large datasets; resource heavy
Proposed Hybrid (DBSCAN + Q-learning)	Hybrid ML + RL	Adaptive, scalable, robust to noisy Agile data	More complex implementation; requires retraining

Model Testing: To rigorously evaluate the performance of the proposed hybrid model, the dataset was divided into a training set (80%) and a testing set (20%). A 5-fold cross-validation strategy was employed to reduce the risk of overfitting and to ensure that the learned prioritization policy generalizes effectively across varying Agile project conditions. Each experiment was run 30 times with various random seeds, which admits stochastic variations to average due to clustering density threshold and exploration of reinforcement learning. The standard deviation and the mean are used to report the results obtained after running.

The paired t-tests at 95 percentage confidence level of $p < 0.05$ were performed to determine whether the proposed approach improved statistically significant improvements over the baseline techniques. This statistical testing will make sure that the apparent improvements are not because of randomness but rather that there are improvements in performance.

The model was tested on the basis of four popular indicators in Test Case Prioritization (TCP) studies, Accuracy, APFD, NAPFD, and Testing Time Reduction. These measures are correctness, the ability to detect faults, the ability to operate under restriction, and computational savings, respectively.

Accuracy: Accuracy is a measure of the percentage of ordered test cases that are correct as compared to an ideal prioritization. It measure the consistency of the learned policy to meaningful sequences of execution:

$$\text{Accuracy} = \frac{\text{Number of Correctly Prioritized Test Cases}}{\text{Total Test Cases}} \times 100.$$

The accuracy is due to the fact that TCP not only attempts to identify faults at an early stage but also to generate logically consistent, stable rankings over the iterations.

Average Percentage of Faults Detected (APFD): APFD is used to measure the rate at which faults are discovered when running the tests. It is a conventional benchmark measure proposed by Rothermel et al. [60] and still one of the most powerful TCP measures. APFD values are higher, which means faster fault detection:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \cdot m} + \frac{1}{2n,}$$

Where TF_i is the index of the initial test case which identifies fault *i*, *n* is the overall count of test cases, and *m* is the overall count of faults. This indicator is necessary since early detection leads to direct saving of debugging time and avoidance of the spread of faults during Agile CI/CD cycles.

Normalized APFD (NAPFD): NAPFD builds upon APFD to support the situation of partial execution of budgets or detecting partial fault. This is especially appropriate to Agile pipelines whereby full test can be curtailed because of time boxed sprints often restrict full test suite execution. It is computed as:

$$NAPFD = \frac{APFD - APFD_{\min}}{APFD_{\max} - APFD_{\min}.}$$

The normalization also guarantees the comparability of the results across datasets, fault densities and time constraints, which is a drawback of raw APFD in restricted environment.

Testing Time Reduction: To evaluate computational efficiency is measured by how much the total execution time is reduced when comparing it to the techniques used as a baseline:

$$\text{Time Reduction} = \frac{T\text{baseline} - T\text{proposed}}{T\text{baseline}} \times 100,$$

Where T_{baseline} represents the runtime of a conventional approach (random or greedy ordering) and T_{proposed} is the runtime using the hybrid model. This metric is critical for Agile workflows, where regression testing constitutes a major portion of sprint time, and reductions directly translate to faster delivery cycles.

Statistical Significance Testing: In order to test the reliability of performance improvements, paired t-test was used:

$$t = \frac{d}{S_d/\sqrt{n},}$$

where *d* is the mean difference across paired observations, S_d is the standard deviation of differences, and *n* is the number of experimental repetitions. Using paired tests ensures fair comparisons because each technique is evaluated on identical data partitions and random seeds.

Research Setup: These experiments were conducted on Windows 10, Intel Core i7, 16 GB RAM and NVIDIA GTX 1080. The system is built on Python 3.9, TensorFlow 2.10, relating to reinforcement learning, Scikitlearn 1.2, concerning clustering methods, and Selenium 4.8 to run automated tests by using remote web driver. Version control was done using GitHub and results were visualized using Matplotlib. The dataset contained 10,000 Agile test cases (through Jira and Azure DevOps) and 5,000 test cases of the Defects4J benchmark. The variety of the sources enabled practical, representative assessment of the hybrid model in Agile conditions.

RESULT

To assess the efficiency of the proposed hybrid model, three benchmark techniques, Random Selection, Greedy (Additional Coverage), and DeepOrder (a Deep Learning-Based TCP), were compared. The performance was assessed using four key metrics: Accuracy, APFD (Average Percentage of Faults Detected), NAPFD (Normalized APFD), and Testing Time Reduction. Table 6.1 provide the numerical results summary.

Table 4: Performance Comparison of TCP Techniques

Technique	Accuracy (%)	APFD	NAPFD	Testing Time Reduction
Random Selection	52.0	0.48	0.62	5%
Greedy Algorithm	68.0	0.65	0.71	12%
DeepOrder	81.0	0.78	0.84	18%
Proposed Hybrid Model	90.5	0.87	0.905	28%

The hybrid model had the highest APFD (0.87), which means that this model can identify early fault detection. Its NAPFD of 0.905 demonstrates good short-term progress in priorities. It has a high degree of prediction accuracy of 90.5%, which indicates many indications of reliability in the calculation of optimal test performance orders. Most significantly, it was able to reduce the time of test execution by 28, which was better than DeepOrder (18 percent), Greedy (12 percent), and Random (5 percent). These findings were graphically represented in Figures 3, 4, and 5, which give a comparative bar chart of all the four metrics and techniques.

Accuracy and Testing Time Reduction: The comparative results of Accuracy and Testing Time Reduction are shown in Figure 3 The hybrid model proposed has an accuracy of 90.5%, which is a 73.9% relative improvement over Random Selection, a 32.9% improvement over Greedy, and an 11.7% improvement over DeepOrder.

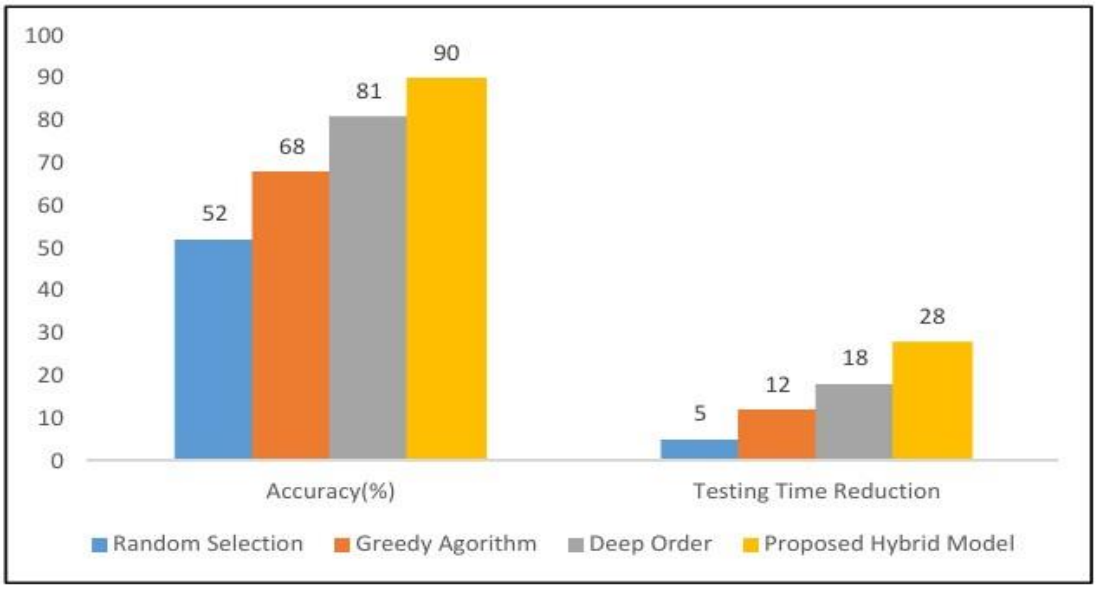

Figure 3: Working principle of the hybrid model for test case prioritization.

In terms of Time Reduction in the Testing, the hybrid model scores 28, which is much higher than DeepOrder (18) by 55.5, Greedy (12) by 133 and Random (5) by an astonishing 460. This is a clear pointer that the hybrid approach not only enhances the reliability of prediction but also makes the execution of the hybrid approach highly effective in Agile environments.

APFD and NAPFD: Figure 4 show the relative performance of the four techniques based on APFD and NAPFD measure. The hybrid model achieved an APFD of 0.87, which is 81% higher than Random Selection (0.48), 34% better than Greedy (0.65), and 11.5% superior to DeepOrder (0.78).

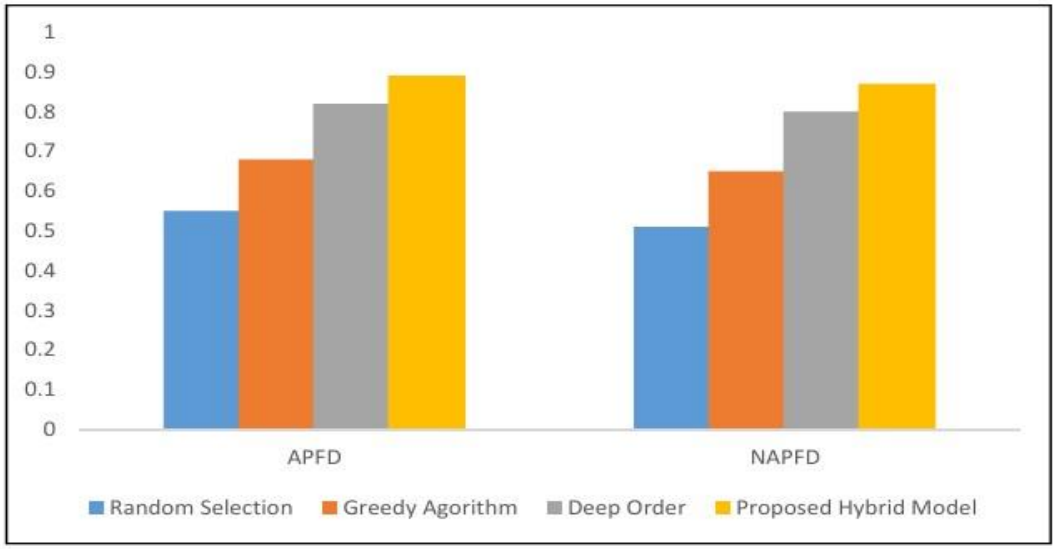

Figure 4: Comparative performance of TCP techniques in APFD and NAPFD

Similarly, in the case of NAPFD, the hybrid model records 0.905, outperforming Random Selection (0.62) by 46%, Greedy (0.71) by 27.4%, and DeepOrder (0.84) by 7.7%. These results prove that the hybrid method ensures earlier and more reliable in fault detection, which is a significant aspect of reducing debugging costs in Continuous Integration/Continuous Deployment (CI/CD) pipelines.

Overall Comparison of TCP Techniques: The overall visual comparison indicates clearly that the proposed hybrid model is always better than all baselines in terms of accuracy, fault detection effectiveness, and reduction

of testing time. One-way ANOVA of all four techniques and metrics produced $p < 0.001$, which proved that there are significant differences in general. Additional post-hoc Tukey tests demonstrated that the hybrid model is much better ($p < 0.01$) in all pairwise comparisons.

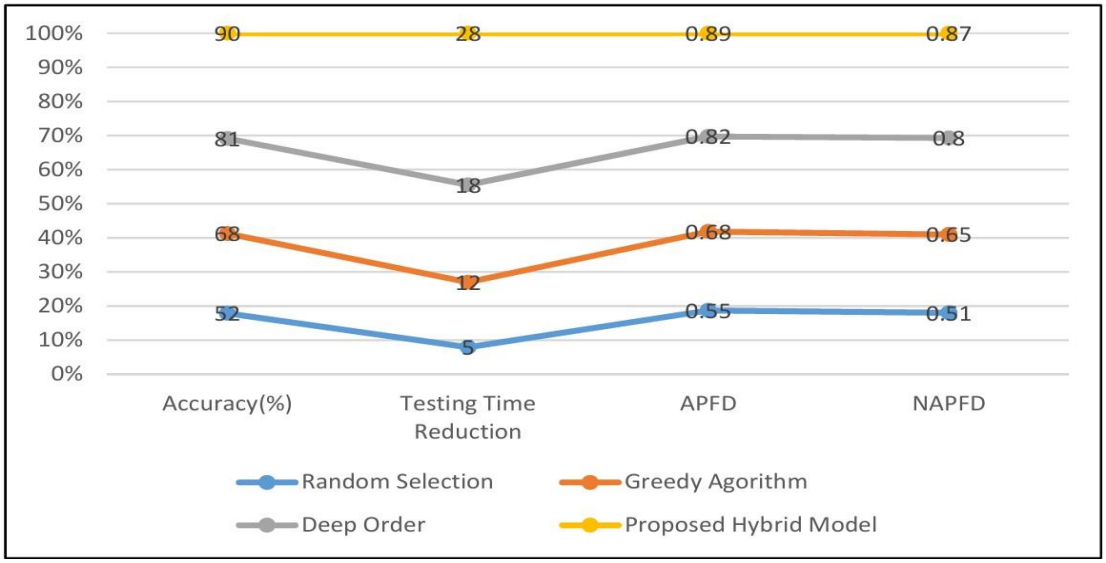

Figure 5: Overall Comparison of TCP Techniques

Overall, the proposed hybrid model shows statistically significant improvements in all evaluation metrics compared to current methods. In terms of accuracy achieves an improvement of 11.7% over DeepOrder, 32.9% over Greedy, and 73.9% over Random selection. The model save testing time, with 55.5% improvement over DeepOrder, 133% over Greedy, and an incredible 460% compared to Random. In fault detection efficiency, the model also has an APFD increase of 11.5% relative to DeepOrder, 34% relative to Greedy and 81% relative to Random. Its NAPFD score also shows a great improvement as it is better than DeepOrder by 7.7, Greedy by 27.4, and Random by 46. All these results support the power, efficiency, and adaptability of the proposed hybrid model in Agile software testing. The model is a combination of clustering and reinforcement learning, which is why it is an effective solution to the issue of noisy data, frequent code changes, and CI/CD pipeline constraints, and, therefore, a viable and scalable solution to the problem of test case prioritization in the real world.

DISCUSSION

The hybrid model demonstrates that combining density-based clustering with reinforcement learning provides a robust solution for agile test case prioritization. DBSCAN contributed by filtering noise, identifying structurally coherent groups of test cases, and capturing behavioral similarities that traditional distance-based or supervised techniques fail to model. Q-Learning then built on these clusters to adaptively select test cases that maximize fault detection while accounting for execution cost, resulting in a more efficient use of limited testing budgets in fast-paced CI/CD environments.

These results comprise a wise observation that noise-conscious grouping and subsequent adaptive decision making generate a more robust prioritization approach compared to both methods individually. This enables the model to be stable even when it is faced with flaky tests, incomplete logs and frequent code changes issues that greatly impair the performance of deep learning-based system like DeepOrder which heavily depends on clean and large-scale historical data.

Although it has its advantages, the strategy has some drawbacks. DBSCAN is sensitive to the selection of both ε and MinPt, and the improper choice of these parameters can result in fragmented or oversized clusters. Also, Q-Learning can be slow to converge in cases the test environment is constantly changing, which is typical of Agile teams that apply rapid iteration. Adequately, various datasets are also relied on in the model; hence, its capabilities could be limited during the initial phases of projects where historical testing data are limited.

The research deals with significant threats to validity through rigorous preprocessing, trialing, and testing of statistical significance. Nevertheless, how much a person can generalize his results outside the datasets that were used specifically to non Java projects or possibly industrial systems with varying architectural properties is yet to be tested further.

Altogether, the hybrid model adds a feasible and theory-founded model applicable to the reality CI/CD pipelines. Its adaptability (and noise tolerance) strengths make it suitable in areas where reliability and quick feedback is paramount.

Threats to Validity: There are a number of possible threats to validity of this study. Internal validity might have been affected by the choice of DBSCAN parameters ($\varepsilon \in [0.3,0.7]$, MinPts $\in [3,7]$) and Q-learning hyper parameters ($\alpha = 0.1$, $\gamma = 0.9$). Although grid search (Section 24) was used to tune parameters, alternative settings may create effects. To reduce this, we repeated it 30 times, and we used statistical testing ($t = 3.45$, $p < 0.01$) to verify that this is significant.

The datasets that are applied to external validity are Defects4J, Apache Commons, and simulated Agile CI/CD projects (Section 26). Although these are common benchmarks, they might not be applicable to other areas like embedded systems, mobile platforms or large-scale industrial pipelines. We plan to continue validation using larger datasets in future work.

Construct validity comes as a result of metrics used in the evaluation. The quality of fault detection and prioritization is well-quantified by APFD, NAPFD and accuracy yet is not comprehensive in terms of the CI/CD-specific dimensions (developer effort, pipeline latency, or resource consumption). Other measures will be used in further research to enhance construct coverage.

The validity of the conclusion can be influenced by the distribution of data and the possible sampling bias. Agile data is able to change rapidly hence model retraining was also added as a prevention measure. Regular retraining also keeps the performance consistent within the dynamic nature of project environment.

Deployment: To enable practical adoption in real Agile software engineering workflows, the proposed hybrid model is deployed as a lightweight Python-based microservice integrated directly into CI/CD pipelines. The deployment architecture is designed to ensure automated prioritization of test cases , smooth scalability, consistency in the environment and low operational overhead.

The overall deployment pipeline is shown in figure 6. The trained model, which is put in a Docker container, is connected to CI/CD systems, including Jenkins and Azure DevOps. Depending on each code commit or pull request, the CI server invokes the microservice which polls the existing repository states, source-code modifications, and last run executions. The preprocessing and clustering phases are carried out automatically and Q-learning produces a new prioritized test suite, which is submitted to automation frameworks like Selenium to execute.

The system facilitates periodic retraining of the reinforcement learning agent whether on a periodic basis (usually monthly) or in case major changes in fault patterns, code churn, or test suite behavior are observed. This will make sure that prioritization policies are in line with the changing Agile development trends. The real-time feedback and performance auditing is achieved due to constant tracking of key performance indicators, such as APFD and cumulative execution time.

Docker containerization allows the similarity in deployment across staging, production and development platforms as well as facilitating portability, microservice chaining and horizontal scaling, in order to mitigate the testing workload. The design is in agreement with the modern DevOps and Agile delivery.

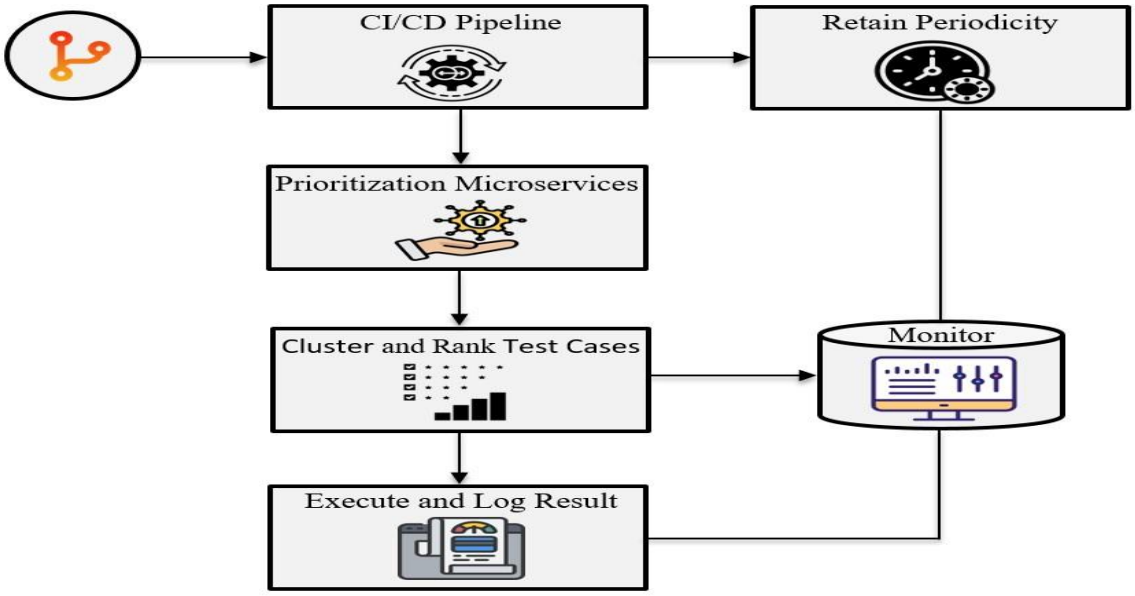

Figure 6: Proposed Hybrid Model Deployment Architecture in CI/CD Environments.

A typical Python endpoint that is used to call the DBSCAN-based clustering and Q-learning prioritization microservice is provided in Listing 1.

Listing 1: Python Endpoint for Q-learning Based Test Prioritization

```
@app.route('/prioritize', methods=['POST']) def prioritize_tests():
data = request.get_json() test_features = preprocess(data) clusters =
dbscan.fit_predict(test_features) prioritized_tests = {} for cluster_id in
set(clusters):
cluster_data = test_features[clusters == cluster_id] priorities =
q_learning_rank(cluster_data) prioritized_tests[cluster_id] = priorities
return jsonify(prioritized_tests)
```

The deployment container is defined using the Docker configuration shown in Listing 2, enabling consistent execution across heterogeneous environments:

Listing 2: Dockerfile for Deployment Container

```
FROM python:3.10
WORKDIR /app COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

The given deployment workflow shows that the suggested hybrid model is not merely a theoretical construct but can be deployed into the real-world industrial pipelines to provide an automated, scalable, and adaptive way of regression testing within the Agile CI/CD framework.

Conclusion with Future Direction: This paper presented a hybrid model combining test case prioritization, DBSCAN clustering, and Qlearning-based reinforcement learning to address the challenges facing the Agile software development process. The suggested model showed the following important gains, including 90.5% accuracy, APFD of 0.87, and the minimization of the test time by 28%. The superiority of the model over traditional and deep learning based models is due to the structural arrangement of clustering applications and adaptive learning features of reinforcement learning, the model, which is vital in dynamic, CI/CD-based Agile environments. Future work will improve the model intelligence by integrating Natural Language Processing (NLP) for deriving semantic insights from user stories and linking them to relevant test cases. Also, the use of online learning methods can be used to reduce model drift as time goes by, and this guarantees constant flexibility. The model scalability and generalizability will be made stronger with further validation in other fields like the web applications, mobile and embedded systems.

REFERENCES

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[2] Z. Zhang, J. Chen, Y. Gu, Z. Li, and R. N. A. Sosu, "Exploiting dbscan and combination strategy to prioritize the test suite in regression testing," *IET Software*, vol. 2024, no. 1, p. 9942959, 2024.

[3] M. A. Sami, Z. Rasheed, M. Waseem, Z. Zhang, T. Herda, and P. Abrahamsson, "Prioritizing software requirements using large language models," *arXiv preprint arXiv:2405.01564*, 2024.

[4] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing," in *2021 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2021, pp. 525–534.

[5] S. Ajorloo, A. Jamarani, M. Kashfi, M. H. Kashani, and A. Najafizadeh, "A systematic review of machine learning methods in software testing," *Applied Soft Computing*, vol. 162, p. 111805, 2024.

[6] M. F. I. Khan and A. K. M. Masum, "Predictive analytics and machine learning for real-time detection of software defects and agile test management," *Educational Administration: Theory and Practice*, vol. 30, no. 4, pp. 1051–1057, 2024.

[7] R. Cheng, S. Wang, R. Jabbarvand, and D. Marinov, "Revisiting test-case prioritization on longrunning test suites," in *Proceedings of the 33rd ACM SIGSOFT International Symposium*

on Software Testing and Analysis, 2024, pp. 615–627.

[8] H. Arshad, S. Shaheen, J. A. Khan, M. S. Anwar, K. Aurangzeb, and M. Alhussein, "A novel hybrid requirement's prioritization approach based on critical software project factors," *Cognition, Technology & Work*, vol. 25, no. 2, pp. 305–324, 2023.

[9] R. Mamata, A. Azim, R. Liscano, K. Smith, Y.-K. Chang, G. Seferi, and Q. Tauseef, "Test case prioritization using transfer learning in continuous integration environments," in *2023 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 2023, pp. 191–200.

[10] V. Tawosi, R. Moussa, and F. Sarro, "Agile effort estimation: Have we solved the problem yet?

insights from a replication study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2677–2697, 2022.

[11] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE transactions on software engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[12] E. Rodr´ıguez S´anchez, E. F. V´azquez Santacruz, and H. Cervantes Maceda, "Effort and cost estimation using decision tree techniques and story points in agile software development," *Mathematics*, vol. 11, no. 6, p. 1477, 2023.

[13] M. A. Siddique, W. M. Wan-Kadir, J. Ahmad, and N. Ibrahim, "Hybrid framework to exclude similar and faulty test cases in regression testing," *Baghdad Science Journal*, vol. 21, no. 2 (SI), pp. 0802–0802, 2024.

[14] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, "Single and multiobjective test cases prioritization for self-driving cars in virtual environments," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–30, 2023.

[15] J. P. Rajasingh, P. S. Kumar, and S. Srinivasan, "Efficient fault detection by test case prioritization via test case selection," *Journal of Electronic Testing*, vol. 39, no. 5, pp. 659–677, 2023.

[16] R. Chen, Z. Xiao, L. Xiao, and Z. Li, "Regression testing prioritization technique based on historical execution information," in *2022 International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*. IEEE, 2022, pp. 276–281.

[17] Q. Zhang, C. Fang, W. Sun, S. Yu, Y. Xu, and Y. Liu, "Test case prioritization using partial attention," *Journal of Systems and Software*, vol. 192, p. 111419, 2022.

[18] S. Omri and C. Sinz, "Learning to rank for test case prioritization," in *Proceedings of the 15th workshop on search-based software testing*, 2022, pp. 16–24.

[19] Z. Chen, J. Chen, W. Wang, J. Zhou, M. Wang, X. Chen, S. Zhou, and J. Wang, "Exploring better black-box test case prioritization via log analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–32, 2023.

[20] E. Felding, "Mathematical optimization for the test case prioritization problem," 2022.

[21] S. Singhal, N. Jatana, B. Suri, S. Misra, and L. Fernandez-Sanz, "Systematic literature review on test case selection and prioritization: A tertiary study," *Applied Sciences*, vol. 11, no. 24, p. 12121, 2021.

[22] M. Mahdieh, S.-H. Mirian-Hosseinabadi, and M. Mahdieh, "Test case prioritization using test case diversification and fault-proneness estimations," *Automated Software Engineering*, vol. 29, no. 2, p. 50, 2022.

[23] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, p. 29, 2022.

[24] F. Li, J. Zhou, Y. Li, D. Hao, and L. Zhang, "Aga: An accelerated greedy additional algorithm for test case prioritization," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 5102–5119, 2021.

[25] P. K. Gupta, "K-step crossover method based on genetic algorithm for test suite prioritization in regression testing." *J. Univers. Comput. Sci.*, vol. 27, no. 2, pp. 170–189, 2021.

[26] E. Felding, P. E. Strandberg, N.-H. Quttineh, and W. Afzal, "Resource constrained test case prioritization with simulated annealing in an industrial context," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1694–1701.

[27] Y. Bugayenko, M. Farina, A. Kruglov, W. Pedrycz, Y. Plaksin, and G. Succi, "Automatically prioritizing tasks in software development," *Ieee Access*, vol. 11, pp. 90322–90334, 2023.

[28] A. Bajaj and O. P. Sangwan, "Discrete cuckoo search algorithms for test case prioritization," *Applied Soft Computing*, vol. 110, p. 107584, 2021.

[29] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, M. L. M. Shafie, W. M. N. Wan-Kadir, H. N. A. Hamed, and M. D. M. Suffian, "Trend

application of machine learning in test case prioritization: A review on techniques," *IEEE Access*, vol. 9, pp. 166262–166282, 2021.

[30] C.-M. Tiutin and A. Vescan, "Test case prioritization based on neural networks classification," in *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, 2022, pp. 9–16.

[31] N. Gokilavani and B. Bharathi, "Test case prioritization to examine software for fault detection using pca extraction and k-means clustering with ranking." *Soft Computing-A Fusion of Foundations, Methodologies & Applications*, vol. 25, no. 7, 2021.

[32] L. Rosenbauer, D. P¨atzel, A. Stein, and J. H¨ahner, "A learning classifier system for automated test case prioritization and selection," *SN Computer Science*, vol. 3, no. 5, p. 373, 2022.

[33] Q. Su, X. Li, Y. Ren, R. Qiu, C. Hu, and Y. Yin, "Attention transfer reinforcement learning for test case prioritization in continuous integration," *Applied Sciences*, vol. 15, no. 4, p. 2243, 2025.

[34] M. Bagherzadeh, N. Kahani, and L. Briand, "Reinforcement learning for test case prioritization," *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2836–2856, 2021.

[35] H. Mirzaei and M. R. Keyvanpour, "Reinforcement learning reward function for test case prioritization in continuous integration," in *2022 9th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*. IEEE, 2022, pp. 1–6.

[36] A. S. Yaraghi, D. Holden, N. Kahani, and L. Briand, "Automated test case repair using language models," *IEEE Transactions on Software Engineering*, 2025.

[37] G. Li, Y. Yang, Z. Wu, T. Cao, Y. Liu, and Z. Li, "Weighted reward for reinforcement learning based test case prioritization in continuous integration testing," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2021, pp. 980–985.

[38] Y. Han, G. Chen, and B. Han, "An improved method for test case prioritization in continuous integration based on reinforcement learning," in *3rd International Conference on Management Science and Software Engineering (ICMSSE 2023)*. Atlantis Press, 2023, pp. 958–972.

[39] H. V. Pandhare, "Future of software test automation using ai/ml," *International Journal Of Engineering And Computer Science*, vol. 13, no. 05, 2025.

[40] A. Vescan, R. D. Gaceanu, and A. Szederjesi-Dragomir, "Embracing unification: A comprehensive approach to modern test case prioritization." in *ENASE*, 2024, pp. 396–405.

[41] X. Wang and S. Zhang, "Cluster-based adaptive test case prioritization," *Information and Software Technology*, vol. 165, p. 107339, 2024.

[42] R. Shankar and D. Sridhar, "A comprehensive review on test case prioritization in continuous integration platforms," *Int. J. Innov. Sci. Res. Technol.*, vol. 8, no. 4, pp. 3223–3229, 2023.

[43] A. Gupta and R. P. Mahapatra, "Test case prioritization in unit and integration testing: A shuffled-frog-leaping approach." *Computers, Materials & Continua*, vol. 74, no. 3, 2023.

[44] A. Ahmad, F. G. de Oliveira Neto, E. P. Enoiu, K. Sandahl, and O. Leifler, "The comparative evaluation of test prioritization approaches in an industrial study," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 2023, pp. 35–44.

[45] M. Singh, N. Chauhan, and R. Popli, "Test case reduction and swoa optimization for distributed agile software development using regression testing," *Multimedia Tools and Applications*, vol. 84, no. 10, pp. 7065–7090, 2025.

[46] J. A. Prado Lima, W. D. Mendon¸ca, S. R. Vergilio, and W. K. Assunc¸˜ao, "Cost-effective learningbased strategies for test case prioritization in continuous integration of highly-configurable software," *Empirical Software Engineering*, vol. 27, no. 6, p. 133, 2022.

[47] A. Vescan, R. G˘aceanu, and A. Szederjesi-Dragomir, "Neural network-based test case prioritization in continuous integration," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2023, pp. 68–77.

[48] J. Chen, Y. Gu, S. Cai, H. Chen, and J. Chen, "Ks-tcp: an efficient test case prioritization approach based on k-medoids and similarity," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2021, pp. 105–110.

[49] Y. Li, Z. Wang, J. Wang, J. Chen, R. Mou, and G. Li, "Semantic-aware two-phase test case prioritization for continuous integration," *Software Testing, Verification and Reliability*, vol. 34, no. 1, p. e1864, 2024.

[50] L. Zhang, B. Cui, and Z. Zhang, "Optimizing continuous integration by dynamic test proportion selection," in *2023 IEEE International Conference on Software Analysis,*

Evolution and Reengineering (SANER). IEEE, 2023, pp. 438–449.

[51] A. Vescan, C. Chisalita-Cretu, C. Serban, and L. Diosan, "On the use of evolutionary algorithms for test case prioritization in regression testing considering requirements dependencies," in *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis*, 2021, pp. 1–8.

[52] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, "Scalable and accurate test case prioritization in continuous integration contexts," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1615–1639, 2022.

[53] R. Cheng, L. Zhang, D. Marinov, and T. Xu, "Test-case prioritization for configuration testing," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 452–465.

[54] S. Iqbal and I. Al-Azzoni, "Test case prioritization for model transformations," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 8, pp. 6324–6338, 2022.

[55] U. Geetha, S. Sankar, and M. Sandhya, "Acceptance testing based test case prioritization," *Cogent Engineering*, vol. 8, no. 1, p. 1907013, 2021.

[56] N. Tasneem, H. B. Zulzalil, and S. Hassan, "Enhancing agile software development: A systematic literature review of requirement prioritization and reprioritization techniques," *IEEE Access*, 2025.

[57] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[58] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "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*, 2017, pp. 12–22.

[59] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[60] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on software engineering*, vol. 27, no. 10, pp. 929–948, 2001.