# A Comprehensive Review on Test Case Prioritization in Continuous Integration Platforms

R.Shankar
Assistant Professor
Department of ICT and Cognitive Systems
Sri Krishna Arts and Science College
Coimbatore, Tamil Nadu, India

Dr. D. Sridhar
Assistant Professor
Department of Computer Science
Sri Krishna Adithya College of Arts and Science
Coimbatore, Tamil Nadu, India

**Abstract:- Continuous Integration (CI) platforms enable recurrent integration of software variations, creating software development rapidly and cost-effectively. In these platforms, integration, and regression testing play an essential role in Test Case Prioritization (TCP) to detect the test case order, which enhances specific objectives like early failure discovery. Currently, Artificial Intelligence (AI) models have emerged widely to solve complex software testing problems like integration and regression testing that create a huge quantity of data from iterative code commits and test executions. In CI testing scenarios, AI models comprising machine and deep learning predictors can be trained by using large test data to predict test cases and speed up the discovery of regression faults during code integration. But these models attain various efficiencies based on the context and factors of CI testing such as varying time cost or the size of test execution history utilized to prioritize failing test cases. Earlier research on TCP using AI models does not often learn these variables that are crucial for CI testing. In this article, a comprehensive review of the different TCP models using deep-learning algorithms including Reinforcement Learning (RL) is presented to pay attention to the software testing field. Also, the merits and demerits of those models for TCP in CI testing are examined to comprehend the challenges of TCP in CI testing. According to the observed challenges, possible solutions are given to enhance the accuracy and stability of deep learning models in CI testing for TCP.**

***Keywords:-*** *Software Testing, Continuous Integration Testing, Regression Testing, Test Case Prioritization, Artificial Intelligence, Deep Learning, Reinforcement Learning.*

## I. INTRODUCTION

Agile adoption by software firms is increasing the popularity of continuous integration (CI) solutions [1]. CI platforms make it possible to incorporate software upgrades more frequently, which leads to quicker and more affordable software testing [2]. They automatically promote functions including build, test execution, and test results analysis to address problems and identify faults. Regression testing is a process that takes up a significant amount of time in a CI cycle (also known as a build) [3]. Minimization, selection, and priority are three categories for the techniques that support regression testing [4-5]. Typically, the test case minimization method minimizes the test set depending on specific circumstances and eliminates redundant test cases. The methods used for test case selection choose a subset of test cases, leaving just the most crucial ones for software testing. The TCP approaches make an effort to rearrange a test suite in order to determine the ideal arrangement of test cases, which improves some goals, such as early failure detection. A broad comparison of these three regression testing techniques is shown in Table 1.

TCP methods, one of three types of approaches, are the most well-known in the field and the focus of this study. To deal with the test cost, insufficient resources, and limitations of CI platforms, most traditional TCP approaches require adjustments [6]. For instance, due to time constraints and test cost for a design, the adoption of search-based approaches or others that need extensive code analysis and coverage may be unfeasible.

Table 1. Comparison of Different Regression Test Methods in CI Platform

| **Component** | **Regression test methods** | | |
|---|---|---|---|
| | **Minimization** | **Selection** | **Prioritization** |
| *Plan* | Remove the test case. | Change-aware test case. | Test case permutation by ordering and prioritizing. |
| *Benefit* | Useful in decreasing test cases. | Useful in choosing change-aware test cases. | Suitable while new test cases can often be considered in the test case permutation. |
| *Drawback* | Test cases are not change-aware. | New test cases might be lost in the temporary selection, i.e., change-aware. | Time-consuming, large test suite. |

### *A. Continuous Integration Platforms*

In former times, a small number of developers would work independently for an extended period of time and would only integrate their adjustments to the master unit once they were done. On the other hand, this method includes drawbacks such a significant time commitment, unneeded administrative expenses for the projects, and faults that go undetected over extended periods of time. The quick delivery of updates to customers was delayed by such factors. Continuous software engineering practices like CI, Continuous Deployment (CD), and Continuous Delivery (CDE) have become popular and accepted by many industries [7] as a result of the growth of the agile development model. These practices allow for the regular combination, testing, deployment, and quick client feedback in a very short cycle. Figure 1 shows how these behaviors are related to one another.

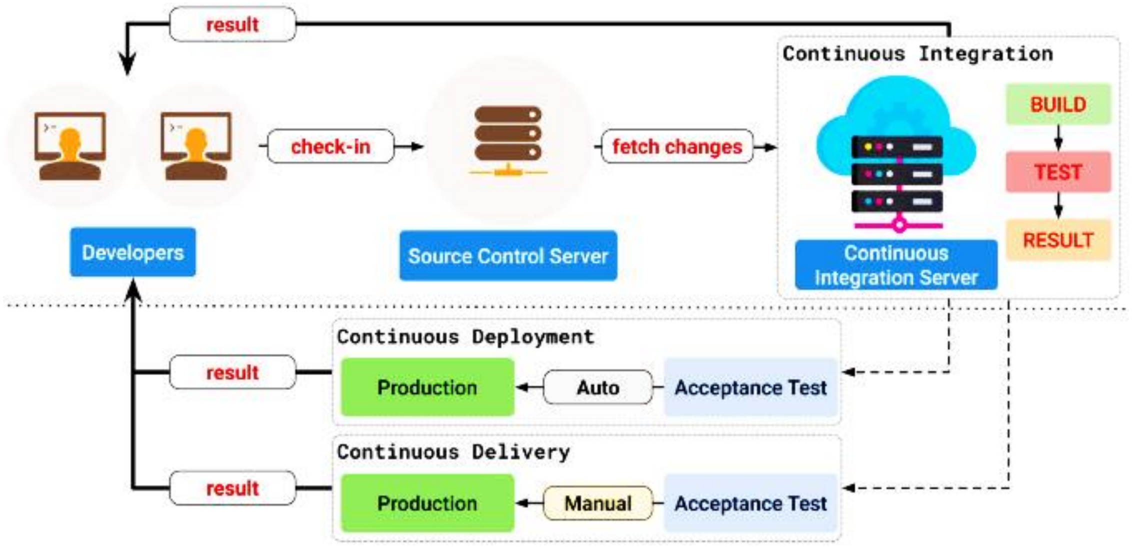

Fig.1 Overview of Association between CI, CD, and CDE Practices (Source: [3])

Before CD and CDE, CI is a vital practice to implement. CI platforms include automated software design and testing, helping engineering teams scale up personnel and distribute results while also enabling software designers to work independently on feature sets concurrently. They can accomplish this independently and quickly if they are willing to incorporate such traits into the final result. Buildbot [8], GoCD [9], Integrity [10], Jenkins [11], and Travis CI [12] are the most well-known public CI servers.

CDE focuses on packaging an artifact (i.e., the application's construction-ready state) for distribution for acceptance testing. The artifact must be ready to be provided to end-clients (construction) in this manner at any time. A CD, on the other hand, can autonomously pack, start, and deliver the software artifact to the building site. Using CI, the artifact used in CD and CDE was successfully transported to the integration stage [13].

To evaluate and release new software updates quickly and affordably, improve error identification and software performance, integration and regression testing is a critical task in CI platforms. This is because CI allows for quick test feedback, which results in test cycles being time-constrained [14]. The time costs might vary from cycle to cycle, and they include time for selecting critical tests to run, running the tests, and reporting test results to designers.

Therefore, using traditional TCP techniques in the CI platform requires certain modifications. The methods must take into account specific aspects of CI platforms, such as parallel test case execution and resource distribution, the unpredictable nature of test cases, which can be included and removed in subsequent commits, and the discovery of numerous errors as quickly as possible.

### *B. Test Case Prioritization*

The TCP issue is described as follows, per Rothermel et al. [15], for a test suite T, the collection of all promising prioritizations (orderings) of T (PT), and a function f that gauges the effectiveness of a certain prioritization from PT to real number:

$$T' \in PT \text{ s.t. } (\forall T'' \in PT)\ (T'' \neq T')[f(T') \geq f(T'')]$$

The goal of a TCP issue is to find the best $T'$ possible while achieving specific goals. Due to insufficient resources, a full regression test suite cannot be implemented in a regression testing scenario. TCP approaches can be time-consuming and require each test location in some circumstances. The fact that the coverage is maintained, however, also makes this attribute advantageous. Additionally, TCP permits failing test cases to be implemented initially due to the time and cost constraints for the test activity that delay the execution of each test case. This ensures that the greatest possible fault coverage is established while using fewer resources and lowering test costs.

The following are a few objectives of TCP techniques, according to Rothermel et al. [15]: (1) raising the test suite's error recognition rate even before regression test execution begins, (2) raising the system code coverage under test, (3) raising the rate at which high-risk errors are identified, and

(4) raising the likelihood that errors connected to specific code modifications will be discovered. Several TCP approaches have been developed in the literature to achieve these goals.

Such approaches are divided into cost-aware, coverage-based, distribution-based, history-based, requirement-based, model-based, and AI-based categories based on the data used in the TCP (as shown in Table 2) [16].

Table 2. Different Categories of TCP Methods

| Category | Description |
|---|---|
| Cost-aware | It prioritizes test cases depending on the test case costs since their costs are not equal. |
| Coverage-based | It prioritizes test cases depending on the code coverage. |
| Distribution-based | It prioritizes test cases depending on the test case profile distribution. |
| History-based | It prioritizes test cases depending on test case execution history data and code modifications. |
| Requirement-based | It prioritizes test cases depending on data extracted from requirements. |
| Model-based | It prioritizes test cases depending on data extracted from models like UML graphs. |
| AI-based | It prioritizes test cases depending on AI models such as machine learning, deep learning, and probabilistic theories. |
| Search-based | It prioritizes test cases depending on multiple objectives of TCP or test case execution. |

The exploration of AI models incorporating machine learning, deep learning, reinforcement learning, and probabilistic theories along with search-based algorithms is a recent trend that is being explored. Utilizing deep learning and reinforcement learning models for approaching TCP techniques in the CI platform has the potential to be a promising fix.

The deep reinforcement learning-based TCP models used in the CI platform for software testing are covered in-depth in this paper. The advantages and disadvantages of various models are also examined in order to solve the issues and offer viable options for enhancing TCP in the CI platform. The remaining paragraphs are organized as follows: The various AI models for TCP in the CI platform are reviewed in Section II. The study's conclusion and recommendations for further research are presented in Section III.

## II. SURVEY ON TEST CASE PRIORITIZATION IN CONTINUOUS INTEGRATION PLATFORM USING ARTIFICIAL INTELLIGENCE MODELS

In order to automatically learn test case prioritization and selection in the CI platform, Spieker et al. [17] presented a novel approach called Retecs. This shortens the time between code pushes and developer feedback on failed test cases. Based on their execution speed, previous final executions, and failure histories, test cases in Retecs were chosen and prioritized using RL. Additionally, the Retecs was taught to recognize earlier CI cycles and rank error-prone test cases according to reward value.

An improved regression testing technique called CTFF was created by Ali et al. [18] for CI and agile software development. Initially, test cases that frequently change were grouped together and given a priority. In the event of a tie, test cases were ranked based on the corresponding failure frequencies and coverage requirements. Then, from all clusters, test cases with a greater frequency of failure or coverage requirements were picked for execution.

To independently forecast the initial build in a string of build failures as well as the residual build failures, Jin & Servant [19] created a system called SmartBuildSkip. With the aid of this approach, developers were given the freedom to decide how much they wanted to sacrifice by storing numerous builds or waiting until a build had failed. Based on the automated build-result prediction, it can lower the cost of CI.

An approach using Combinatorial VOlatiLE Multi-Armed Bandit (COLEMAN) for TCP in the CI platform was presented by Lima & Vergilio [20]. Using historical test case failure data and RL, the MAB was merged with combinatorial and volatile elements to adaptively discover a sufficient prioritized test suite for all CI cycles. To prioritize test cases and improve software privacy, Shi et al. [21] created a new ReLU (Rectified Linear Unit)-weighted Historical Execution (RHE) data reward function. Many historical execution outcomes received weighted rewards based on previous execution data with varied lengths, hence weighted reward functions with multiple lengths for historical outcomes were established.

In order to prioritize DNN testing based on the statistical view of DNN for classifying high-dimensional objects, Feng et al. [22] created a method called DeepGini. With this approach, the problem of assessing set impurity can be reduced along with the problem of computing misclassification chance. As a result, the tests that were presumably misclassified were quickly found, and the DNN's resilience in various classification tasks was increased.

By implementing a batch update method akin to Monte Carlo control for automatically ranking test cases, Rosenbauer et al. [23] enhanced the learning strategy of XCS as an RL model. Additionally, they investigated if the prioritized experience reply for the test prioritization problem has similar favorable impacts on XCS as it does on the neural network. A scalable strategy for CI and regression testing in IoT-based systems was created by Medhat et al. [24]. This model was built using IoT-related TCP and selection criteria. An optimized prioritized set of test cases was initially obtained using search-based approaches. In order to periodically ensure the overall dependability of IoT-based systems, the selection was then based on a trained prediction model for IoT standard devices using supervised deep-learning algorithms.

For the purpose of prioritizing hybrid and consensus regression tests, Mondal & Nasre [25] presented the Hansie approach. The TCP was represented as a social choice theory rank aggregation dilemma. Priority-aware hybridization and priority-blind computation of a consensus ordering from different prioritizations were two components of the Hansie. It utilized normal windows in the absence of ties and irregular windows in the presence of ties to execute the combined test-case orderings in parallel using several processes, leading to a high test execution speed.

In order to increase the number of test failures found while reducing the number of tests, Nguyen & Le [26] created the RLTCP test prioritizing technique. To represent the underlying association between test cases for the user interface testing, a weighted coverage graph was constructed. The coverage graph and RL for TCP were integrated in the RLTCP.

For the purpose of prioritizing regression tests in CI, Sharif et al. [27] created a time-effective deep learning-based regression system called DeepOrder. The DNN was trained as a regression approach using historical test data regarding the timing and status of test cases. As a result, the number of failed test cases was decreased and the priority of the test cases inside a particular test suite was determined. By incorporating the multidimensional properties of the Extended Finite State Machine (EFSM) under test, Huang et al. [28] created a new learn-to-rank technique for prioritizing test cases. In order to train the ranking model for TCP, the random forest approach included numerous heuristic prioritization techniques.

Two distinct approaches, including the test suite-based dynamic sliding window and the individual test case-based dynamic sliding window for TCP, were presented by Yang et al. [29]. For all CI tests, a fixed-size sliding window with a preset length of recent historical data was first implemented. Following that, techniques for dynamic sliding windows were created, where the size of the window was constantly adaptable to all CI testing.

At a conceptual level, Yaraghi et al. [30] proposed a data model that gathers data sources and their linkages in a typical CI platform. Then, using this data model, a thorough set of features was developed, consisting of every feature previously used by related studies. Additionally, these attributes were used to appropriately prioritize test cases and train machine learning models.

A brand-new Black-box TCP (BTCP) model with log preprocessing, log representation, and TCP modules was created by Chen et al. [31]. The LogTCP paradigm states that different log-based BTCP schemes were created by combining different log representation technologies with diverse priority techniques. Using various ranking algorithms, Bagherzadeh et al. [32] analyzed the sequential interactions between the CI platform and the TCP as an RL problem. Additionally, the TCP strategy was automatically and continually learned using RL models with the aim of getting as close to the optimum strategy as possible.

Table 3 compares all of the aforementioned AI models for TCP in CI platforms based on their benefits and drawbacks.

Table 3. Comparison of Different AI Models for TCP in CT Platforms

| Ref. No. | Models | Benefits | Drawbacks | Dataset | Performance |
|---|---|---|---|---|---|
| [17] | Retecs | Time consumption was less since it did not need to execute computationally intensive tasks during prioritization. | In this model, only a few metadata of a test case and its history were used. | Paint Control, IOF/ROL, and GSDTSR | - |
| [18] | CTFF | It achieved a high error recognition rate and can detect more errors fastly. | The results were not statistically verified due to the use of already collected datasets. | Case A (CA), Case B (CB), and Case C (CC) datasets from already presented case studies | Precision: CA=0.92; CB=0.9; CC=0.92 Recall: CA=1; CB=0.99; CC=1 F-measure: CA=0.96; CB=0.93; CC=0.96 |
| [19] | SmartBuildSkip | It can save cost in CI while maintaining most of its value, with the ability to modify its | It may not be useful for software projects that cannot afford a single delay in observing failing | TravisTorrent dataset | Recall=100%; Saved builds=89%; Saving efficiency=92% |

| | | | | | |
|---|---|---|---|---|---|
| | | cost-value trade-off. | builds. | | |
| [20] | COLEMAN | Better performance for early fault recognition. | It needs to learn a relationship between the number of test cases and the prioritization time for increasing model scalability. | Paint Control, IOF/ROL, and GSDTSR | Prioritization period=0.0654sec |
| [21] | RHE reward function | It can maximize the number of testing cases that had already discovered faults within the available time. | The time spent executing functions based on the overall rewards was relatively longer than that consumed by the partial rewards. | Paint Control, IOF/ROL, and GSDTSR | Execution time=595000sec |
| [22] | DeepGini | Highly beneficial. | It needs more features to increase scalability. | MNIST, CIFAR-10, Fashion (Zalando's article images), and Street View House Numbers (SVHN) | Mean accuracy=0.96 |
| [23] | Learning strategy of XCS | It was a viable solution to the adaptive test case selection problem. | The sample efficiency and reproducibility needed to improve further. | Paint Control, IOF/ROL, and GSDTSR | Estimated variances of the agents=0.019997 |
| [24] | Scalable model using Long Short-Term Memory (LSTM) classifier | It can ensure the total reliability of IoT-based systems. | Advanced deep learning such as RL was needed to increase the accuracy. | IoT device connection efficiency Dataset and MIoT dataset | Accuracy: Regression testing=90%; Integration testing=92% |
| [25] | Hansie | It achieved better performance without considering the test execution history from earlier code modifications. | It considered only independent unit test cases. | 12 real-time codes from Software-artifact Infrastructure Repository and 8 public projects from GitHub | Effectiveness of change coverage=1 |
| [26] | RLTCP | It can be resilient to drastic and structural changes in the test suite because of using the weighted coverage graph. | The performance was influenced by the reward policy. | Spectrum_OR, and Mattermost_OR | Mean ratio of test suite execution needed for 100% fault coverage=55.6% |
| [27] | DeepOrder | Better flexibility and time complexity to efficiently deal with large-scale datasets. | It needs more features of test cases or data such as modification in source code, etc., for increasing efficiency. | Cisco dataset, Paint Control, IOF/ROL, and GSDTSR | Average mean ratio of errors recognized=0.723 |
| [28] | Learn-to-rank method | Efficient for prioritizing test cases. | The performance may affect the construct validation of the experiment because it does not consider the time cost of test case execution and the fault security level. Also, it was time-consuming. | Monitor, INRES, Class II, OLSR, and SCP from EFSM | Average mean ratio of errors recognized=0.884; Total time cost=3.21 |
| [29] | Test suite-based | It can effectively | The failure rate was | 14 different | Average normalized mean |

| | | | | | |
|---|---|---|---|---|---|
| | dynamic sliding window and the individual test case-based dynamic sliding window | improve the prioritization effect of test cases. | low, resulting in many test executions without effective reward values that provide slow learning or complicated convergence. | industrial datasets | ratio of errors recognized=37.45%; Mean recall=72.5% |
| [30] | Data model at a conceptual level | It can achieve promising results across most subjects. | High-quality datasets were required to evaluate TCP models. | Four publicly accessible datasets | Mean accuracy=83.975% Average mean ratio of errors recognized=0.82 |
| [31] | LogTCP | Effective for detecting faults. | More advanced techniques were needed to improve the efficiency of real-time TCP. | - | Average mean ratio of errors recognized=0.7884 |
| [32] | RL algorithms like pairwise ranking, ACER, and an actor critic-based RL | It can achieve a significant accuracy enhancement to prioritize test cases. | Hyperparameters of RL were not optimized. Only a limited number of reward functions were analyzed, which influence the RL efficiency. | Simple and enriched history datasets. | Average mean ratio of errors recognized=0.79 |

## III. CONCLUSION

The numerous TCP approaches based on RL and deep learning in the CI platform were discussed in this article in detail. Additionally, the pros and shortcomings of each model were highlighted along with its efficiency. This investigation revealed that the TSP problem requires the optimum integration of various data, which deep learning or RL enables. The TSP problem is a crucial topic in CI platforms, with regular builds and regression testing activities. On the other hand, RL-based or deep learning-based TCP models solve ongoing issues. In the future, more sophisticated deep learning and RL algorithms will need to be looked into in order to create extremely reliable TCP models. Furthermore, because prior RL research mainly used execution history information for TCP, a more comprehensive feature set may be taken into consideration for further enhancing RL performance.

## REFERENCES

[1]. I. C. Donca, O. P. Stan, M. Misaros, D. Gota, and L. Miclea, Method for continuous integration and deployment using a pipeline generator for agile software projects. *Sensors*, *22*(12), 1-18, 2022.
[2]. V. K. Makam, Continuous integration on cloud versus on premise: a review of integration tools. *Advances in Computing*, *10*(1), 10-14, 2020.
[3]. M. Shahin, M. A. Babar, and L. Zhu, Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, *5*, 3909-3943, 2017.
[4]. D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, Coverage-based regression test case selection, minimization and prioritization: a case study on an industrial system. *Software Testing, Verification and Reliability*, *25*(4), 371-396, 2015.
[5]. M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, D. N and R. Tumeng, Test case prioritization approaches in regression testing: a systematic literature review. *Information and Software Technology*, *93*, 74-93, 2018.
[6]. 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*, *27*(2), 1-34, 2022.
[7]. O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, Exploring the intersection between software industry and software engineering education-a systematic mapping of software engineering trends. *Journal of Systems and Software*, *172*, 1-28, 2021.
[8]. "Buildbot Basics," Buildbot. [Online]. Available: https://www.buildbot.net/. [Accessed: 21-Apr-2023].
[9]. "Open source continuous delivery and release Automation Server," GoCD. [Online]. Available: https://www.gocd.org/. [Accessed: 21-Apr-2023].
[10]. Integrity. [Online]. Available: https://integrity.github.io/. [Accessed: 21-Apr-2023].
[11]. Jenkins. [Online]. Available: https://www.jenkins.io/. [Accessed: 21-Apr-2023].
[12]. "Home – travis-ci," Travis, 29-Nov-2022. [Online]. Available: https://www.travis-ci.com/. [Accessed: 21-Apr-2023].
[13]. S. Buchanan, J. Rangama and N. Bellavance, CI/CD with Azure Kubernetes Service. *Introducing Azure Kubernetes Service: A Practical Guide to Container Orchestration*, 191-219, 2020.

[14]. E. A. Da Roza, J. A. P. Lima, R. C. Silva, and S. R. Vergilio, Machine learning regression techniques for test case prioritization in continuous integration environment. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*, pp. 196-206, 2022.

[15]. G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, *27*(10), 929-948, 2001.

[16]. R. Mukherjee, and K. S. Patnaik, A survey on different approaches for software test case prioritization. *Journal of King Saud University-Computer and Information Sciences*, *33*(9), 1041-1054, 2021.

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

[18]. S. Ali, Y. Hafeez, S. Hussain, and S. Yang, Enhanced regression testing technique for agile software development and continuous integration strategies. *Software Quality Journal*, *28*, 397-423, 2020.

[19]. X. Jin, and F. Servant, A cost-efficient approach to building in continuous integration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 13-25, 2020.

[20]. J. A. P. Lima, and S. R. Vergilio, A multi-armed bandit approach for test case prioritization in continuous integration environments. *IEEE Transactions on Software Engineering*, *48*(2), 453-465, 2020.

[21]. T. Shi, L. Xiao, and K. Wu, Reinforcement learning based test case prioritization for enhancing the security of software. In *IEEE 7th International Conference on Data Science and Advanced Analytics*, pp. 663-672, 2020.

[22]. Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In *ACM Proceedings of the 29th International Symposium on Software Testing and Analysis*, pp. 177-188, 2020.

[23]. L. Rosenbauer, A. Stein, R. Maier, D. Pätzel, and J. Hähner, Xcs as a reinforcement learning approach to automatic test case prioritization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1798-1806, 2020.

[24]. N. Medhat, S. M. Moussa, N. L. Badr, and M. F. Tolba, A framework for continuous regression and integration testing in IoT systems based on deep learning and search-based techniques. *IEEE Access*, *8*, 215716-215726, 2020.

[25]. S. Mondal, and R. Nasre, Hansie: Hybrid and consensus regression test prioritization. *Journal of Systems and Software*, *172*, 1-42, 2021.

[26]. V. Nguyen, and B. Le, RLTCP: a reinforcement learning approach to prioritizing automated user interface tests. *Information and Software Technology*, *136*, 1-16, 2021.

[27]. A. Sharif, D. Marijan, and M. Liaaen, DeepOrder: Deep learning for test case prioritization in continuous integration testing. In *IEEE International Conference on Software Maintenance and Evolution*, pp. 525-534, 2021.

[28]. Y. Huang, T. Shu, and Z. Ding, A learn-to-rank method for model-based regression test case prioritization. *IEEE Access*, *9*, 16365-16382, 2021.

[29]. Y. Yang, C. Pan, Z. Li, and R. Zhao, Adaptive reward computation in reinforcement learning-based continuous integration testing. *IEEE Access*, *9*, 36674-36688, 2021.

[30]. A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. Briand, Scalable and accurate test case prioritization in continuous integration contexts. *IEEE Transactions on Software Engineering*, 1-27, 2022.

[31]. Z. Chen, J. Chen, W. Wang, J. Zhou, M. Wang, X. Chen, and J. Wang, Exploring better black-box test case prioritization via log analysis. *ACM Transactions on Software Engineering and Methodology*, 1-33, 2022.

[32]. M. Bagherzadeh, N. Kahani, and L. Briand, Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering*, *48*(8), 2836-2856, 2022.