2024 5th International Conference on Image Processing and Capsule Networks (ICIPCN) | 979-8-3503-6717-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICIPCN63822.2024.00135

A Novel Software Test Data Generation Framework based on Multi-level Fuzzy Clustering Algorithm

Hongjie Lin
School of Economics and Management, Xiamen University of Technology, Xiamen, Fujian,361024 ,China
lin800506@126.com

***Abstract*— Software testing plays a crucial role in ensuring the quality and reliability of software products. This study presents a novel software test data generation framework based on the multi-level fuzzy clustering algorithm. The framework aims to optimize the testing process by efficiently generating test cases and prioritizing them for execution. The proposed methodology integrates hierarchical reinforcement learning and hierarchical clustering techniques to improve the effectiveness and comprehensiveness of software testing. A detailed review of recent advances in software testing methodologies is provided, focusing on differential regression testing for REST APIs, test case prioritization, program repair using neural translation models, and other key areas. The proposed framework is evaluated through experiments on real-world software datasets, demonstrating its superiority in terms of non-redundancy rate and vulnerability count detection compared to existing methods. The results highlight the effectiveness and relevance of the proposed framework in improving the efficiency and reliability of software testing.**

Keywords*— *Software Testing; Data Generation; Multi-level Fuzzy Clustering; Algorithm Framework

I. INTRODUCTION

The software testing process is crucial for ensuring the software quality [1]-[3] and involves the strict execution of business processes. It provides users with a comprehensive overview of software testing work, including the process, workload, and content, and also provides guidance for testers. In recent years, due to the widespread adoption of software applications, the importance of the software testing has become increasingly prominent, prompting researchers to invest in the extensive research on the software testing process. Research on software testing process primarily encompasses the following aspects:

1. Researchers are committed to exploring the different software testing models, such as the waterfall model, agile model, DevOps model, etc. By comparing and analyzing different models, the latest studies can find the best practices suitable for different projects and teams, and improve testing efficiency and quality.

2. Researchers have also done a lot of work in designing and implementing software testing process management systems or tools to help the testing team better organize and manage testing tasks, test cases, defect tracking, etc., and improve the overall collaboration efficiency.

3. Improving and optimizing the software testing process has also attracted a lot of attention. By introducing new testing technologies, improving the existing testing methods, and optimizing testing processes and strategies, the latest studies can further improve the efficiency and coverage of testing, thereby improving the quality and stability of software.

Among all testing methodologies, continuous integration testing is the core approach. In continuous integration testing, as new versions of programs integrate into the mainline, corresponding test case suites evolve accordingly. Executing the entire initial test suite sequentially can be time-consuming. Furthermore, without incorporating test cases for newly added features, comprehensive system testing may become impossible, which introduces vulnerabilities in regression testing. Researchers have then proposed diverse testing optimization methods to address these challenges, which are tailored to specific application scenarios. These optimization techniques are outlined in Figure 1.

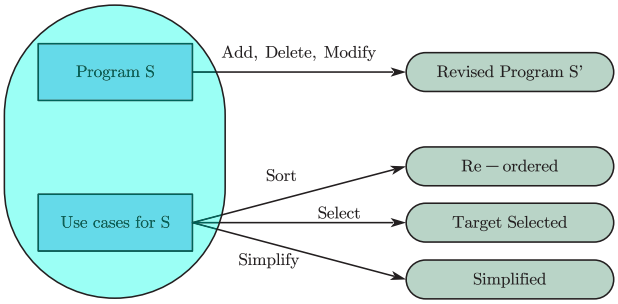

Fig. 1. The Continuous Integration Testing Steps

To better test the software from the perspectives of the robustness and effectiveness (R&E), this study proposes implementing a novel software test data generation framework based on the multi-level fuzzy clustering algorithm. In the subsequent sections, the novel methodology and its associated tests are illustrated, commencing with the review of software testing approaches.

II. LITERATURE REVIEW

This review consolidates recent advances in the software testing methodologies, emphasizing some various techniques designed to address the complexity inherent in the modern software systems. Key areas covered include differential regression testing for REST APIs, test case prioritization in black-box testing, program repair using neural translation models, the use of computational qualitative data analysis for theory generation, validation approaches in autonomous driving, bug triaging, software reliability modeling, robustness

979-8-3503-6717-1/24/$31.00 ©2024 IEEE
DOI 10.1109/ICIPCN63822.2024.00135

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.

testing of the REST services, data center load balancing, and testing techniques for deep neural networks.

Godefroid, Lehmann, and Polishchuk [4] present a novel and efficient strategy for pinpointing critical changes in REST APIs via a differential examination procedure. This method, which utilizes RESTler to create HTTP request progressions, contrasts the outcomes of different API iterations to uncover regressions. Applied to 17 iterations of Azure networking APIs, the approach effectively unearthed numerous regressions, demonstrating its effectiveness in maintaining API coherence amidst continuous enhancements. Chen et al [5] propose an innovative test case prioritization (TCP) technique to improve the effectiveness of regression testing. By using K-medoids clustering, their approach overcomes the limitations of conventional string distance-based TCP. This methodology involves categorizing test cases, applying an eager sorting mechanism within clusters, and employing a polling approach to refine test execution sequences, thereby increasing both temporal efficiency and anomaly detection rates.

Lutellier et al. [6] present CoCoNuT, a program repair technique that integrates convolutional neural networks (CNNs) with a context-aware neural machine translation (NMT) architecture. CoCoNuT uniquely represents erroneous code and its context, enabling effective bug fixing across programming languages. The ensemble learning approach combines multiple models, resulting in superior bug-fixing performance over traditional techniques on the six popular benchmarks. Dalkin et al [7] explored the use of NVivo, a computer-assisted qualitative data analysis software, in real-world assessment. NVivo helps manage complex data sources and create detailed audit records, thereby increasing the transparency and rigor in the theory generation and improvement process. This structured approach addresses the need for systematic documentation in real-world research and supports iterative and comprehensive theory development. Stević et al. [8] describe the development and validation of an advanced driver assistance system (ADAS) using the Robot Operating System (ROS) and the CARLA simulator. Their approach uses software-in-the-loop (SIL) technology to test sensing algorithms in a simulated environment before real-world application. This strategy reduces the risks and costs associated with physical testing and ensures the reliability and safety of autonomous driving capabilities through rigorous virtual verification.

Rathor et al [9] propose a novel approach to bug triage using Genetic Algorithms and Time Convolutional Neural Networks (GA-TCN). This method automates the severity classification of bug reports, addressing the inefficiency and subjectivity of traditional triage methods. By pre-processing and normalizing data, the model improves the accuracy and reliability of bug severity assessments, thereby improving the overall efficiency of software maintenance. Pradhan et al [10] develop a multi-release software reliability growth model (SRGM) for open source software (OSS) using a generalized modified Weibull distribution (GMWD). Their non-homogeneous Poisson process (NHPP)-based model addresses the high release frequency of OSS projects. Experimental results validate the model's effectiveness in predicting the software reliability, providing valuable insights for developers and project managers in the OSS community. Laranjeiro et al. [11] present bBOXRT, a black-box tool designed for robustness testing of the REST services. Focusing on minimal interface information, bBOXRT evaluates the resilience of services under unexpected conditions. Analysis of 52 REST services revealed various robustness issues and security vulnerabilities, highlighting the importance of specialized tools to ensure the reliability and security of REST applications.

Liu et al [12] propose RSLB, a robust and scalable load balancing protocol for software-defined data center networks. Using flow cells and link delay metrics, combined with a three-stage routing strategy, RSLB improves load balancing and reduces flow completion time. The distributed control structure enables the effective congestion management and demonstrates significant improvements over existing schemes in both symmetric and asymmetric topologies. Feng et al. [13] present DeepGini, a test prioritization technique aimed at improving the robustness of deep neural networks (DNNs). By statistically measuring set impurity to identify likely mis-classifications, the DeepGini prioritizes tests that effectively detect DNN errors. Empirical studies show that DeepGini outperforms coverage-based techniques in both efficiency and effectiveness, providing a valuable tool for improving the quality and reliability of DNN-based systems. Semenov et al [14] develop a mathematical model for the initial stages of software security testing, focusing on the vulnerability identification. Their model improves the accuracy of security testing by using moment generating functions and considering the cryptographic data protection methods. This structured approach increases the accuracy of vulnerability identification and provides recommendations for improving security testing algorithms.

III. Proposed Methodology

A. The Hierarchical Reinforcement Learning and Hierarchical Clustering Algorithms for Software Testing: An Initial Modelling

Hierarchical reinforcement learning [15]-[17] is a popular research direction in reinforcement learning. In practical reinforcement learning problems, rewards are often sparse, and the combination of large state and action spaces makes direct modeling and training for learning optimal strategies challenging. Hierarchical reinforcement learning addresses this problem by operating at different levels of the temporal abstraction, decomposing complex tasks into a hierarchical structure of sub-problems or sub-tasks. This allows higher-level agents to learn to perform tasks by selecting the best sub-task agents as their actions.

By integrating hierarchical clustering techniques, this approach can effectively perform multidimensional software testing, increasing comprehensiveness and efficiency of the testing. This method not only handles complex reinforcement learning tasks more effectively, but also plays an important role in software testing. The preliminary testing framework adopts a hierarchical reinforcement learning design, forming a two-tier coordinator-executor architecture. The top-level coordinator is responsible for selecting and managing the

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.

bottom-level executors to generate test cases. The bottom layer includes two executors: the dynamic symbolic executor and the fuzz testing executor. The dynamic symbolic executor analyzes program paths and generates corresponding test data, while the fuzz testing executor generates test cases using random or semi-random methods to detect potential defects. This two-tier architecture leverages the benefits of hierarchical reinforcement learning to increase the efficiency and the effectiveness of test case generation and to ensure broader coverage of the test scenarios, thereby further improving the comprehensiveness and reliability of software testing.

Fuzz testing uses initial seeds to generate test cases without constraints, but the quality depends on the seeds. Typically, fuzz testing generates test cases through mutations based on a probability distribution that theoretically covers all possible paths. Its efficiency depends on the selection and prioritization of the generated test cases. Test cases generated by symbolic execution represent unexplored paths and can be used by fuzz testing. Paths generated by replacing symbolic values with concrete values, even if initially inaccessible, can be covered by mutations. Thus, the fuzz testing effectively complements hybrid testing; when symbolic execution struggles to solve for new paths, fuzz testing's randomness can explore new paths, while symbolic execution ensures that the seeds used are meaningful. In the Figure 2, the scheme of the initial test model is demonstrated.

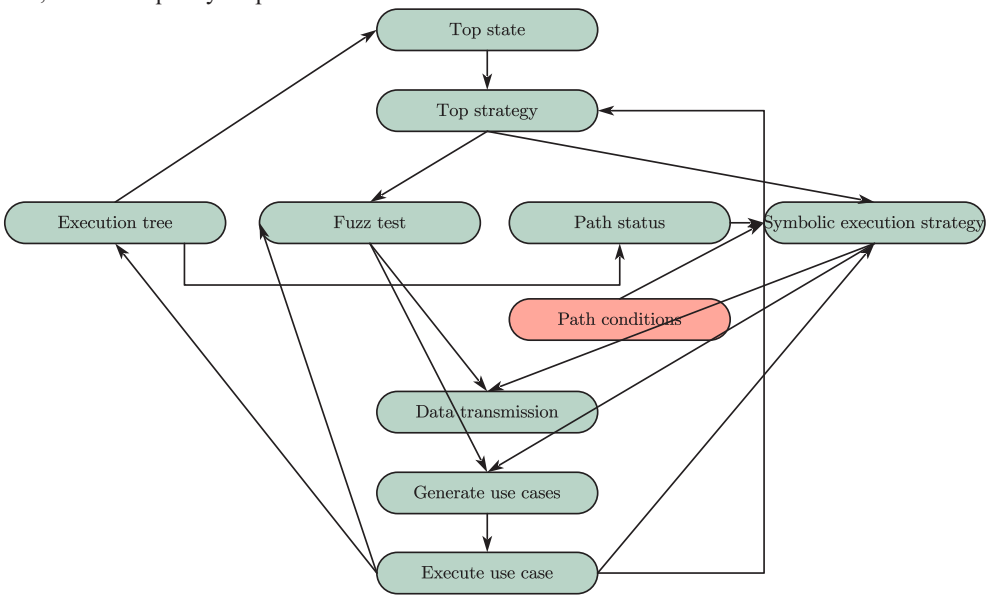

Fig. 2. Scheme of the Initial Test Model

Fuzz testing can quickly generate multiple test cases through random mutations, making it cost-effective. However, without effective seeds, fuzz testing struggles to cover complex conditions or deep branches. The deeper the branch and the more complex the condition, the more constraints there are to satisfy, reducing the likelihood that a mutation will satisfy those constraints. Symbolic execution, on the other hand, can explore deeper branches of the execution tree, but at a higher constraint-solving cost. Therefore, when there are many shallow negatable nodes, symbolic execution should be prioritized to quickly explore deep branches and obtain inputs that satisfy complex path constraints. Conversely, when there are many deep negatable branches, it is more effective to use test cases generated by symbolic execution as seeds for fuzz testing. Then, to improve the initial model, several methods of improvements can be defined in the next sections.

B. Applying the Reinforcement Learning to Training

As discussed above, the reinforcement learning is the core part and this section focuses on the training step. The test optimization process based on reinforcement learning includes the following key steps:

1. Environment modeling: First, the agent needs to model the test environment, including a collection of test cases, the state of the test environment, and the possible results of executing each test case. This can serve as the environment in reinforcement learning with which the agent interacts.

2. Priority assignment: The agent assigns priorities to test cases based on the current situation in the test environment. This can be based on previous experience, characteristics of the current environment, and expected test goals. The assigned priority will affect the subsequent ordering of test cases.

3. Test case sorting: According to the assigned priority, the agent sorts the test cases to determine the execution order. The sorted test cases will be executed sequentially according to priority to maximize testing efficiency and the ability to discover potential problems.

4. Execution and feedback: The agent executes the sorted test cases and calculates the sorting performance based on the execution results. This includes metrics such as the number of errors detected, the location where the errors were found, and the error detection efficiency of the test case sequence. These indicators will be fed back to the agent as reward signals for adjusting the sorting strategy.

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.

5. Strategy adjustment: The agent adjusts the sorting experience and strategy based on the reward signals received. If the test results are good, the corresponding strategy will be enhanced; if problems are found, the strategy will be corrected. In this way, the agent can gradually adapt to different testing environments and learn the optimal test case sequencing strategy.

6. Iterative optimization: Repeating the above steps, the agent continuously interacts with the test environment, executes test cases, collects feedback, and adjusts the strategy based on the feedback. Through the continuous iterative optimization, the agent can learn the most effective test case sequencing strategy in a specific testing environment, thereby improving testing efficiency and accuracy.

In Figure 3, the interaction process between the reinforcement learning agent and environment is shown.

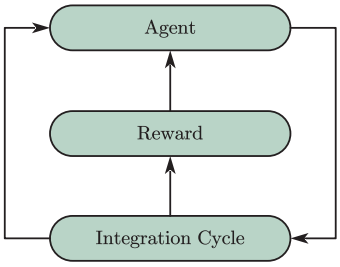

Fig. 3. Interaction Process between Reinforcement Learning Agent and Environment

C. The Designed Automated Testing Architecture

After understanding the test principles, the designed automated testing architecture is introduced in this section, which is based on the Selenium [18]-[20]. To begin with, the Figure 4 shows the history of Selenium.

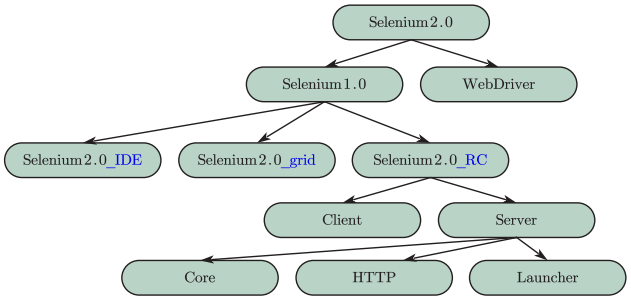

Fig. 4. History of Selenium

In Figure 5, the Selenium test processes are defined.

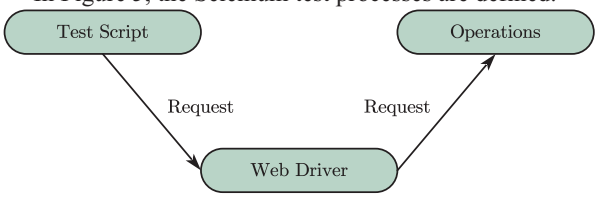

Fig. 5. Selenium Test Processes

IV. Experimental Analysis

To effectively test the proposed model, the experiment used a diverse set of real-world software to comprehensively evaluate the performance of the enhanced hybrid fuzz testing system. These test sets include software from various fields and application scenarios, ensuring coverage of a wide range of the usage conditions and diverse user requirements. The experimental data has been manually collected by the team and includes various analyses and scenarios, including different operating systems, network environments, and user interaction modes, to ensure comprehensive and representative testing. This data includes different types of software, such as financial applications, social media platforms, and enterprise management systems, as well as the complex operating environments, such as high concurrent access, data-intensive operations, and real-time response requirements.

This comprehensive and realistic data enables a detailed evaluation of the performance, reliability, and stability of the hybrid fuzzing systems under the various conditions. Experimental results show that the enhanced hybrid fuzzing test system excels in identifying potential software defects and improving test coverage. The system effectively identifies problems in the complex paths and deep branches, while maintaining low testing costs and efficient execution speeds. For the numerical analysis, 2 major factors are considered:

1. **Non-redundancy rate.**
2. **Vulnerability count detection results.**

In order to evaluate their effectiveness and relevance in the context of this study, the comparison methods mentioned in [9] and [11] were critically analyzed. The real world test results are shown below.

A. The Non-redundancy Rate Test

In the Figure 6, the comparison test result regarding the the non-redundancy rate is illustrated.

It is observed that in all the 50 test data sets, the non-redundant rate of the proposed method is generally higher than the methods described in references [9] and [11]. For example, the non-redundant rate of this method in the test data set exceeds 97%, while most of the methods in the reference literature are around 97%. This result suggests that the proposed method may be more effective in assessing the effectiveness and relevance. This trend of the high non-redundancy rate may originate from the uniqueness of the proposed method in terms of data processing and feature extraction. This method may capture key information in the data more effectively, reduce redundancy, and improve the information utilization of the data. In addition, the method may also be more adaptable and better able to handle various types of data sets.

In further, it is found that in some test data sets, there are certain differences between the proposed method and the method in the reference. These differences may be due to differences in characteristics of the data set, the characteristics of the method itself, or the experimental setup. In future research, the reasons for these differences can be further

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.

investigated and attempts can be made to improve the method to further improve its performance.

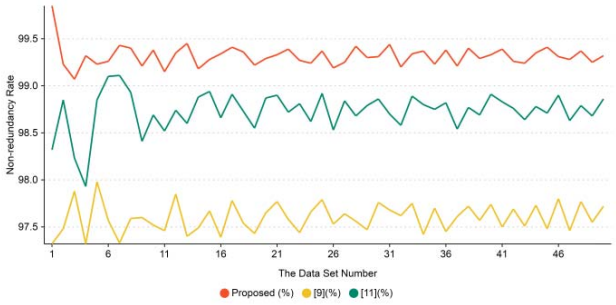

Fig. 6. Non-Redundancy Test Result

B. *Vulnerability Count Detection Results*

The vulnerability counting detection method was used in the experiment, and a comparative analysis was conducted between it and the methods mentioned in the references, that is, the methods described in the literature [9] and [11]. In Figure 7, the testing results are visualized.

First, it is observed that the proposed method shows certain advantages in detecting the number of vulnerabilities in all datasets. In particular, the proposed method detects a higher number of vulnerabilities in most datasets than the methods in Ref. This indicates that the proposed method may be more sensitive and accurate in detecting vulnerabilities. However, it is also noted that on some datasets, the method in the reference may perform much better. This may be due to characteristics of the data-set or limitations of the method itself. For example, for some specific types of vulnerabilities, the reference method may have better detection capabilities. It is also observed that on some datasets there are small differences between the proposed method and the methods in the reference. These differences may be due to noise in the data set, differences in feature selection, or adjustments to algorithm parameters.

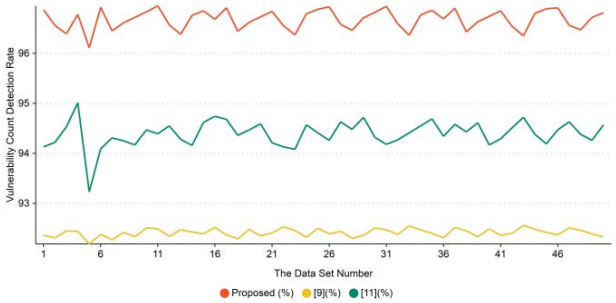

Fig. 7. Vulnerability Count Detection Test Results

V. Conclusion

This study presents a novel software test data generation framework that utilizes the multi-level fuzzy clustering algorithm. By integrating hierarchical reinforcement learning and the hierarchical clustering techniques, the framework effectively generates and prioritizes test cases for execution. Experimental evaluation on the real software datasets demonstrated the effectiveness of the proposed framework. In particular, superior performance was observed in terms of non-redundancy rate and vulnerability count detection compared to existing methods. These results highlight the potential impact of the framework in improving the efficiency and reliability of software testing. In conclusion, the proposed framework offers a promising solution to address the challenges in software testing and provides valuable insights for both researchers and practitioners. Future research efforts can focus on further refining the framework and exploring its adaptability in different software testing scenarios.

References

[1] Pargaonkar, Shravan. "Enhancing Software Quality in Architecture Design: A Survey-Based Approach." International Journal of Scientific and Research Publications (IJSRP) 13, no. 08 (2023).

[2] Mishra, Alok, and Ziadoon Otaiwi. "DevOps and software quality: A systematic mapping." Computer Science Review 38 (2020): 100308.

[3] Gezici, Bahar, and Ayça Kolukısa Tarhan. "Systematic literature review on software quality for AI-based software." Empirical Software Engineering 27, no. 3 (2022): 66.

[4] Godefroid, Patrice, Daniel Lehmann, and Marina Polishchuk. "Differential regression testing for REST APIs." In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 312-323. 2020.

[5] Chen, Jinfu, Yuechao Gu, Saihua Cai, Haibo Chen, and Jingyi Chen. "A novel test case prioritization approach for black-box testing based on K-medoids clustering." Journal of Software: Evolution and Process 36, no. 4 (2024): e2565.

[6] Lutellier, Thibaud, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. "Coconut: combining context-aware neural translation models using ensemble for program repair." In Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis, pp. 101-114. 2020.

[7] Dalkin, Sonia, Natalie Forster, Philip Hodgson, Monique Lhussier, and Susan M. Carr. "Using computer assisted qualitative data analysis software (CAQDAS; NVivo) to assist in the complex process of realist theory generation, refinement and testing." International Journal of Social Research Methodology 24, no. 1 (2021): 123-134.

[8] Stević, Stevan, Momčilo Krunić, Marko Dragojević, and Nives Kaprocki. "Development of ADAS perception applications in ROS and" Software-In-the-Loop" validation with CARLA simulator." Telfor Journal 12, no. 1 (2020): 40-45.

[9] Rathor, Ketan, Jaspreet Kaur, Ullal Akshatha Nayak, S. Kaliappan, Ramya Maranan, and V. Kalpana. "Technological Evaluation and Software Bug Training using Genetic Algorithm and Time Convolution Neural Network (GA-TCN)." In 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), pp. 7-12. IEEE, 2023.

[10] Pradhan, Vishal, Ajay Kumar, and Joydip Dhar. "Modeling Multi-Release Open Source Software Reliability Growth Process with Generalized Modified Weibull Distribution." Evolving Software Processes: Trends and Future Directions (2022): 123-133.

[11] Laranjeiro, Nuno, João Agnelo, and Jorge Bernardino. "A black box tool for robustness testing of REST services." IEEE Access 9 (2021): 24738-24754.

[12] Liu, Yong, Huaxi Gu, Zhaoxing Zhou, and Ning Wang. "RSLB: Robust and scalable load balancing in software-defined data center networks." IEEE Transactions on Network and Service Management 19, no. 4 (2022): 4706-4720.

[13] Feng, Yang, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks." In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 177-188. 2020.

[14] Semenov, Serhii, Zhang Liqiang, Cao Weiling, and Viacheslav Davydov. "Development a mathematical model for the software security testing

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.

first stage." Eastern-European Journal of Enterprise Technologies 3, no. 2 (2021): 111.

[15] Hutsebaut-Buysse, Matthias, Kevin Mets, and Steven Latré. "Hierarchical reinforcement learning: A survey and open research challenges." Machine Learning and Knowledge Extraction 4, no. 1 (2022): 172-221.

[16] Xie, Ruobing, Shaoliang Zhang, Rui Wang, Feng Xia, and Leyu Lin. "Hierarchical reinforcement learning for integrated recommendation." In Proceedings of the AAAI conference on artificial intelligence, vol. 35, no. 5, pp. 4521-4528. 2021.

[17] Pateria, Shubham, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. "Hierarchical reinforcement learning: A comprehensive survey." ACM Computing Surveys (CSUR) 54, no. 5 (2021): 1-35.

[18] Wardhan, Harshita, and Suman Madan. "Study On Functioning Of Selenium TestingTool." International Research Journal of Modernization in Engineering Technology and Science Www. Irjmets. Com@ International Research Journal of Modernization in Engineering (2021): 2582-5208.

[19] Nyamathulla, S., P. Ratnababu, and Nazma Sultana Shaik. "A review on selenium web driver with python." Annals of the Romanian Society for Cell Biology (2021): 16760-16768.

[20] Thooriqoh, Hazna At, Tiara Nur Annisa, and Umi Laili Yuhana. "Selenium Framework for Web Automation Testing: A Systematic Literature Review." Jurnal Ilmiah Teknologi Informasi 19, no. 2 (2021): 65-76.

Authorized licensed use limited to: Universita degli Studi di Salerno. Downloaded on June 11,2025 at 17:33:20 UTC from IEEE Xplore. Restrictions apply.