International Journal on Advanced Science Engineering Information Technology

Vol.15 (2025) No. 2
ISSN: 2088-5334

Application of Reinforcement Learning Techniques in Software Testing of Android Applications: A Systematic Literature Review

Nadiah Mohd Hanim [a], Johanna Ahmad [a,2], Mohd Arfian Ismail [b], Nor Amalina Mohd Sabri [c,1], Shahdatunnaim Azmi [c]

[a] *Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, Johor, Malaysia*
[b] *Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Pahang, Malaysia*
[c] *Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia, Parit Raja, Johor, Malaysia*
Corresponding author: [1]noramalina@uthm.edu.my; [2]johanna@utm.my

***Abstract*—Software testing is a critical process in ensuring the quality and reliability of applications before they are deployed to production. However, it is resource-intensive and often tedious, particularly in the context of Android applications, which pose unique challenges due to their vast state space, diverse user interactions, and variable behaviors. Reinforcement learning (RL), a machine learning framework where agents interact with environments to improve decision-making policies, has gained attention for its potential in software testing. This systematic literature review examines the application of reinforcement learning in software testing of Android applications, focusing on widely researched areas, prevalent techniques, and emerging trends. The review analyzes 22 selected studies from an initial pool of over 30,000 articles published between 2020 and 2024. The findings highlight that automated testing is the primary focus in this domain, with Q-learning emerging as the dominant RL technique. Actor-critic methods, deep Q-networks (DQN), and policy gradient approaches are also explored in several studies, aiming to improve the adaptability and efficiency of testing processes. Most research emphasizes fault detection and coverage maximization, often targeting event-driven interactions and GUI-based behaviors. Despite significant advancements, the study identifies underexplored areas, such as test case prioritization and the integration of user behavior or user interaction data, as promising directions for future research. This review contributes to understanding the current landscape and offers guidance for future RL-based Android application testing investigations.**

***Keywords*—Android application; machine learning; reinforcement learning; software testing; systematic literature review.**

Manuscript received 5 Feb. 2024; revised 14 Oct. 2024; accepted 27 Dec. 2024. Date of publication 30 Apr. 2025.

I. INTRODUCTION

Software applications usually contain a lot of bugs and errors, as they are developed by software developers who naturally cannot escape making mistakes [1]. This is why software testing is crucial before the application is developed and pushed to production, so the application is free from major and minor bugs. Although software testing does not add direct functionality to an application, it can consume over 50% of resources, including time and money [1], [2]. Other than that, performing regression testing after developers resolve a bug can be tedious for the software testers as they have to write test scripts and perform the testing all over again [3], [4]. This is where artificial intelligence (AI) comes to the rescue by making life easier for the software tester [5], [6].

At present, some applications provide AI functionality, sparing the human testers the challenge to comprehensively evaluate the entire product with a human-level precision [2], [7], [8]. Android applications, especially, are challenging to test due to their variability in user interactions [9]. Android applications have a vast state space, making thorough exploration during testing a significant challenge [10]. Testing must consider different gestures, orientations, and user behaviors, making it challenging to create comprehensive test cases [11]. While regression is a statistical approach used to model relationships between variables and predict continuous outcomes, reinforcement learning is a learning framework in which an agent interacts with an environment to improve a policy based on a specified objective, adapting as it perceives the state of the environment [1], [12].

Reinforcement learning is a learning framework in which an agent interacts with an environment to improve a policy based on a specified objective, adapting as it perceives the state of the environment [1], [12]. Reinforcement learning is widely used in software testing of Android applications

because of its ability to learn and adapt [13]. Due to the nature of this technique, many studies have proposed the usage of various reinforcement learning techniques in software testing of Android applications [14]. The growing number of studies on reinforcement learning in software testing for Android applications underscores the need to systematically review current knowledge and identify areas for future research [1].

II. Materials and Method

This SLR review method was designed to align with established best practices by adhering to both general and specific SLR guidelines in software engineering. Specifically, this SLR follows the general SLR guidelines specified by [15] and specific SLR guidelines established by [16]. By following both sets of guidelines, this SLR methodology is structured into three primary phases: planning the review, conducting the review, and reporting the review as shown in Fig. 1. This phased approach ensures a systematic process, beginning with the careful planning and formulation of research questions, progressing through a comprehensive review and concluding with a thorough reporting of findings [15], [16], [17].

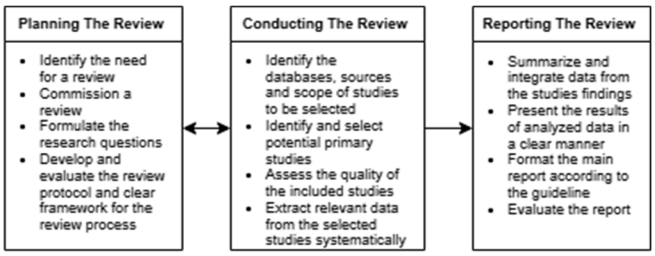

Fig. 1 Systematic Literature Review (SLR) Phases

A. Research Questions

Formulating strong and precise research questions is crucial for an effective SLR. By clearly specifying the research questions, the process of SLR can become more efficient as it helps determine the relevant studies that are aligned with the research objectives. This SLR aims to explore and synthesize the application of reinforcement learning techniques in software testing for Android applications. Aside from that, the researchers aim to investigate how the software testing of Android applications is optimized to address the multimodal user interactions inherent in these applications. The research questions for this SLR comprise five components. These components are known as PICOC and were proposed by [16]. Table 1 shows the criteria and scope of research questions.

TABLE I
CRITERIA AND SCOPE OF RESEARCH QUESTIONS

Criteria	Scope
Population	Reinforcement learning
Intervention	Software testing
Comparison	NA
Outcomes	Reinforcement learning technique applied in the software testing of an Android application
Context	Review(s) of any empirical studies of software testing for Android applications with reinforcement learning

To achieve the aim of this SLR, five research questions have been formulated as follows:

- RQ1: Which part of software testing does the paper discuss?
- RQ2: What reinforcement learning algorithms are used in the paper to improve the area of software testing for Android applications?
- RQ3: What kind of datasets are used in the software testing Android applications using reinforcement learning?
- RQ4: What evaluation metrics are used to assess reinforcement learning algorithms' performance in Android software testing?
- RQ5: Is there any integration of user behavior data or user interaction into the proposed model?

B. Data Sources

Seven online databases have been selected and thoroughly searched for this SLR. The online databases chosen were Google Scholar, Scopus, IEEE Xplore, ScienceDirect, SpringerLink, WOS, and ACM Digital Library. These databases were selected from a list of available online databases subscribed to by UTM's Library.

C. Search Strategy

Establishing a clear search strategy is essential for conducting a systematic literature review. A well-defined search strategy ensures that the search for relevant studies remains aligned with the research objectives. Multiple test runs were done using combinations of keywords and Boolean operators to develop the strategy to determine the optimal search query for finding the relevant studies. The steps taken to create an optimal search query are as follows:

- Derivation of essential keywords based on the research questions formulated.
- Identification of keywords from the relevant studies
- Usage of Boolean operators such as AND to link keywords and OR for alternative keywords

After several test runs, the researcher determined that linking all necessary keywords was optimal to find relevant studies. However, even using AND operators to connect all keywords, the search results still returned over 10,000 results in Google Scholar. The finalized search query is as follows:
("Reinforcement Learning" AND ("Software Testing" OR "Test") AND ("Android Application" OR "Android"))

D. Inclusion and Exclusion Criteria

After several test runs, the researcher determined that linking all necessary keywords was optimal to find relevant studies. As this SLR aims to study all relevant and up-to-date studies on software testing of Android applications using reinforcement learning, it is crucial to define explicit inclusion and exclusion criteria. The inclusion criteria are:

- All papers must be published between 2020 and 2024.
- All papers must be in English.
- All papers must focus on using reinforcement learning techniques for software testing of Android applications.

Each of the studies is reviewed against the exclusion criteria before being accepted for the next phase of data extraction and analysis:

- Theses and dissertations
- Papers that are under three pages in length
- Duplicate studies

- Incomplete papers (e.g., research in progress, lacking empirical results)

E. Quality Assessment

A quality assessment checklist is applied to ensure that the data extracted from the studies meets quality standards. This checklist, designed by [16], is a guideline for evaluating study relevance. Table 2 presents a set of general questions designed to assess the quality of the selected studies. A three-point scale is used for the quality assessment, with scores assigned as follows: Yes = 1, Partially = 0.5, and No = 0. Each paper is assessed against the checklist items, resulting in a quality score ranging from 0 (very poor) to 5 (excellent).

TABLE II
QUALITY ASSESSMENT CHECKLIST

No	Item	Answer
SQ1	Are the aims and objectives of the research clearly stated?	Yes/No
SQ2	Is the research design specified?	Yes/No/Partially
SQ3	Have the researcher(s) adequately carried out the data collection process?	Yes/No/Partially
SQ4	Have the researcher(s) given enough data to support their results and conclusions?	Yes/No/Partially
SQ5	Is there a comparison of other techniques involved in the experiment?	Yes/No

F. Paper Selection

The initial search returned over 30,000 articles. Applying the inclusion criterion of publication years from 2020 to 2024 reduced this number to 16,700. Next, additional inclusion and exclusion criteria—such as language, study area, type of publication (articles and conference papers), and removal of duplicates—were applied, narrowing the results to 160 articles as shown in Fig. 2.

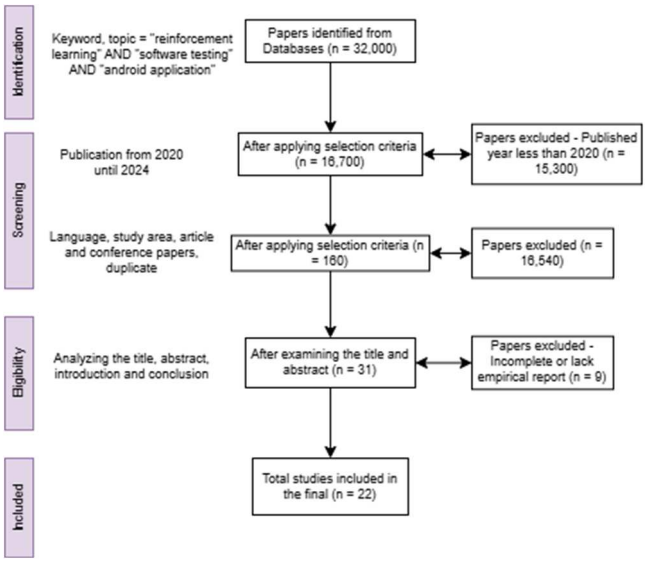

Fig. 2 Papers Selection Process

Following a detailed review of each paper's title, abstract, introduction, and primary content, the selection was further reduced to 31 articles. Finally, after thoroughly reviewing the full content of these 31 papers, 22 were deemed eligible for inclusion in the final study set. Fig. 2 illustrates the flowchart of the paper selection process using the PRISMA approach.

G. Quality Assessment Scores

Table 3 presents the quality assessment scores for each of the 22 studies included in the review. Based on this assessment, 13 papers achieved a good quality score, while 9 received a very good quality score. Many papers rated as good quality did not reach very good quality due to limited use of evaluation metrics. Notably, no papers fell into the fair, poor, or very poor-quality categories, which reflects positively on the quality of studies included in this SLR.

TABLE III
QUALITY ASSESSMENT SCORES

Quality Scale	Very poor (= 1)	Poor (=2)	Fair (=3)	Good (=4)	Very Good (=5)	Total
Number of studies	0	0	0	13	9	22
Percentage	0	0	0	59	41	100

III. RESULTS AND DISCUSSION

This section presents the results of the relevant data extracted from all 22 studies to address the respected research questions. The findings are displayed graphically to enhance visualization and understanding.

A. RQ1: Which part of software testing does the paper discuss?

This question examines the application of reinforcement learning in the software testing of Android applications. It aims to identify which areas of software testing are well-researched, which areas remain underexplored, and which represent emerging fields of study. This analysis helps to highlight both established research trends and potential gaps where further investigation could contribute to the advancement of reinforcement learning in Android software testing. Fig. 3 and Table 4 display the frequency of areas of software testing for Android applications using reinforcement learning.

TABLE IV
FREQUENCY OF RESEARCH AREAS

Research Area	Authors
Automated Testing	[18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33]
Test Case Generation	[34], [35], [36], [37]
Test Case Prioritization	[38], [39]

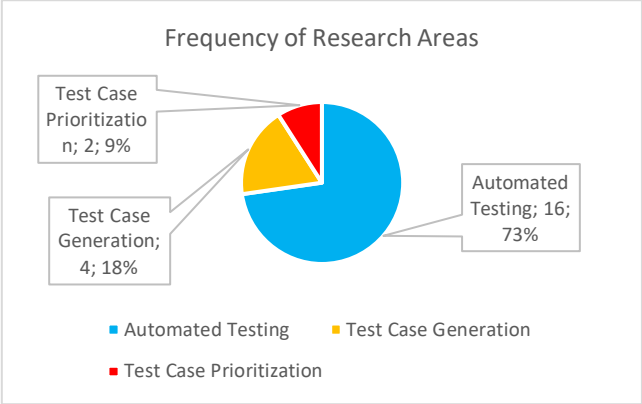

Fig. 3 Frequency of Research Areas

Based on the data summarized, "Automated Testing" is a very popular research area of software testing that utilizes reinforcement learning in software testing of Android applications, with a majority of 16 papers (73%). This data suggests that automated testing is a dominant focus for researchers applying reinforcement learning to Android application testing. In contrast, "Test Case Generation", while recognized, has less coverage, with a frequency of 4 papers (18%) exploring this area. This indicates a moderate level of research interest, focusing on how RL can create diverse test cases based on learned behaviors.

Meanwhile, "Test Case Prioritization" appears to be an emerging area, with only 2 papers (9%) dedicated to it, signaling an opportunity for future research to delve deeper into how reinforcement learning can improve the efficiency of test execution by identifying and prioritizing critical test cases first. Together, these findings highlight a clear preference for automated testing in the field, while also suggesting the potential for expanded RL research in underexplored areas like test case generation and prioritization.

B. *RQ2: What reinforcement learning algorithms are used in the paper to improve the area of software testing for Android applications?*

This research aims to explore the different types of reinforcement learning techniques used in the selected studies. Analyzing this data will provide insights into the gaps and prevalent techniques within the field, highlighting well-researched areas and those requiring further investigation. Fig. 4 displays the frequency of reinforcement learning techniques used in each paper.

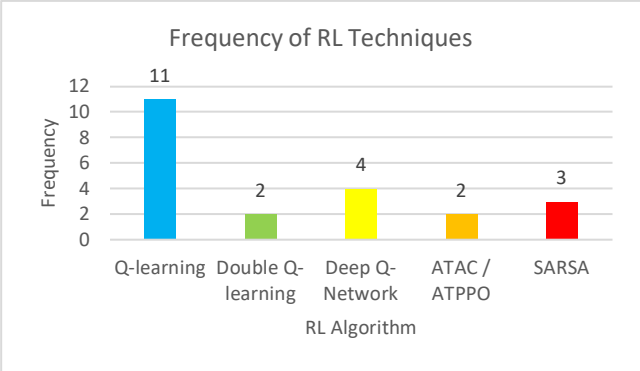

Fig. 4 Frequency of RL Techniques

Based on the chart, Q-learning has the highest frequency of technique utilization, appearing in 11 out of 22 studies. It is widely applied across all areas of software testing, including automated testing, test case generation, and test case prioritization. Q-learning is a model-free, control-temporal difference learning method inspired by behaviorist psychology [23]. It enables an agent to learn optimal actions through trial and error, aiming to maximize cumulative rewards in an unknown environment [34]. The agent gradually identifies the best action to take in similar situations by repeatedly trying actions in various states and receiving delayed rewards. Its popularity in software testing of Android applications suggests that many studies favor simple RL algorithms that can effectively adapt to dynamic environments.

Deep Q-Network (DQN) is the second most common technique, with a frequency of 4, and it is exclusively utilized in the automated testing area of studies. DQN is an extension of the Q-learning algorithm that employs a deep neural network to approximate the action-value function [19]. Because a neural network can input and output high-dimensional state and action spaces, DQN can effectively scale to more complex environments [22]. This capability is advantageous for Android applications, which often involve intricate interfaces and diverse user interactions.

State-Action-Reward-State-Action (SARSA) comes in the third place with a frequency of 3. SARSA is an on-policy algorithm that uses and learns with the same policy to select an action and update the action value [35]. This approach allows SARSA to adjust its policy based on its current actions, making it more cautious and stable in uncertain environments. By learning with the policy, it actively uses, SARSA offers a conservative alternative to off-policy methods like Q-learning, which can benefit scenarios requiring gradual and reliable improvements.

Other techniques like Double Q-learning, ATAC, and ATPPO are least common, with an equal frequency of 2. Both studies that implement the Double Q-learning technique focus on the automated testing area. Double Q-learning in general is an advanced version of the Q-learning algorithm that uses randomization and ensemble learning techniques to reduce overestimation bias that is commonly found in the traditional Q-learning algorithm [32]. In a study by [20], the researcher applied the Double Q-learning algorithm and integrated it with deep reinforcement learning to improve application automation, resulting in ResiDRLTesting (Residual network and Deep Reinforcement Learning Testing).

ATAC is Automatic Testing with Adaptive Coverage that is based on the Advantage Actor-Critic (A2C) algorithm. ATAC is a type of reinforcement learning method that combines both policy (actor) and value (critic) functions to optimize the testing process [31]. A study done by [31] utilizes only ATAC in the paper. The researcher utilizes ATAC to automatically generate test cases to enhance the efficiency and effectiveness of Android GUI testing. A study by [18] proposed ATAC and ATPPO for Android GUI testing to mitigate the state explosion problem. ATPPO stands for "Automated Testing with Proximal Policy Optimization". ATPPO employs the Proximal Policy Optimization (PPO) algorithm, an advanced reinforcement learning technique. PPO is known for its stability and efficiency in training, making it suitable for complex environments like mobile application testing [18]. Similar to ATAC, ATPPO aims to achieve higher levels of code coverage compared to traditional testing tools. This distribution of techniques highlights the dominance of value-based RL methods, while also indicating opportunities for further research into policy-based approaches in Android application testing.

C. *RQ3: What kind of datasets are used in the software testing of Android applications using reinforcement learning?*

This research question aims to investigate the types of datasets used in Android application testing across the 22 studies reviewed. Each of these studies employs Android application datasets for experimental purposes, with two

studies additionally incorporating other dataset types. In a study by [38], the experiment utilizes test-suite execution sequences and user interaction sequences datasets, which are generated from test case execution sequences written in natural language and derived from recorded user interactions with the application. In a study by [25], on the other hand, uses a diverse set of datasets from Kaggle, including categories like banking-related vocabulary, news headlines, movie information, Twitter data, and book excerpts. Fig. 5 illustrates the frequency of each dataset type used across the studies, categorizing them into three types: industry, benchmark, and case study datasets. This analysis provides insight into the variety of data sources used to enhance testing realism and relevance across different studies.

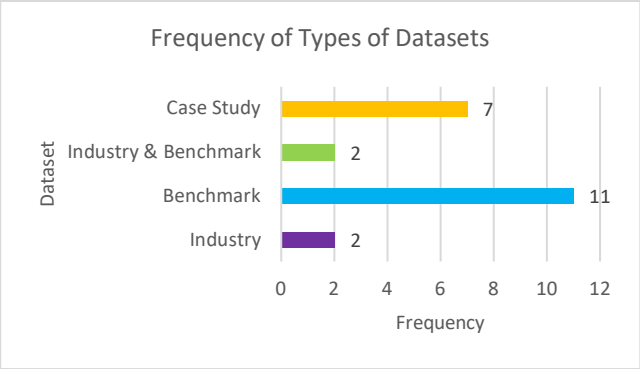

Fig. 5 Frequency of Dataset Type

Benchmark datasets are the most frequently used, appearing in 11 out of the 22 studies reviewed. Case study datasets rank second, utilized in 7 studies. Two studies incorporate a combination of industry and benchmark datasets, leveraging both real-world applications and benchmark applications. Only two studies exclusively use industry datasets, drawing from real-world applications to test their proposed methods in practical scenarios. This distribution highlights a preference for benchmark datasets to ensure comparability.

D. RQ4: What evaluation metrics are used to assess the performance of reinforcement learning algorithms in Android software testing?

The purpose of this research question is to determine the number of evaluation metrics used and what type of evaluation metric is most popular. Fig. 6 shows the number of evaluation metrics used by the papers reviewed.

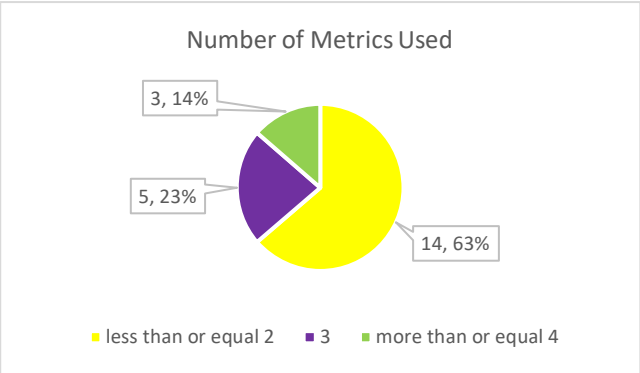

Fig. 6 Frequency of Number of Metrics Used

The chart shows that majority of studies (14 papers) utilize two or fewer evaluation metrics, making this the most common approach. In contrast, only a small subset of studies (3 papers) employ four or more evaluation metrics. This suggests a general tendency among researchers to focus on a limited set of key metrics. In the 22 studies reviewed, over 20 distinct evaluation metrics are used, reflecting the focus on automated testing, test case generation, and prioritization. To facilitate analysis, these metrics have been grouped into broader categories for more precise differentiation. The four main metric types are: code coverage metric, fault detection metric, efficiency metric, and test generation metric. Fig. 7 displays the frequency of evaluation metrics used.

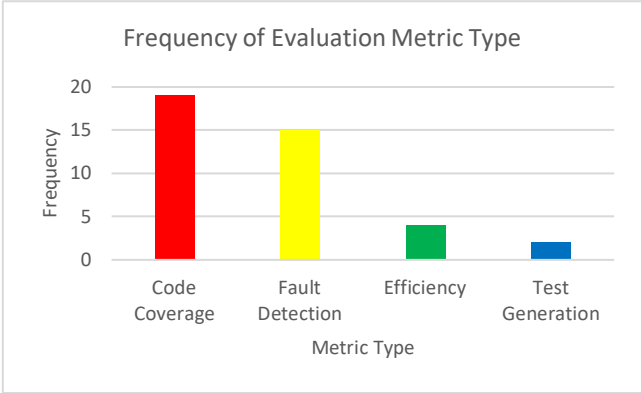

Fig. 7 Frequency of Evaluation Metric Type

The most commonly used metric type is code coverage, with 19 studies incorporating it. This category includes metrics such as instruction coverage, line coverage, branch coverage, method coverage, class coverage, and others that assess the extent of code exercised during testing. Fault detection metrics are the second most frequent, used by 15 studies. These metrics, including the number of failures, fault detection rate, and crashes, measure test cases' effectiveness in identifying defects. Efficiency metrics rank third, appearing in 4 studies, focusing on aspects like execution time and testing speed to evaluate testing performance. Finally, test generation metrics are used in only 2 studies, as only 4 studies focus specifically on test case generation.

E. RQ5: Is there any integration of user behavior data or user interaction into the proposed model?

Android applications often feature complex GUI layouts, presenting unique testing and interaction modeling challenges. This question aims to analyze how many of the 22 selected studies tackle this problem, examining the methods and approaches they employ to handle intricate GUI structures effectively. Fig. 8 shows how many studies integrate user behavior data or user interaction in the proposed model. Based on the result, only 2 papers out of 22 integrate user behavior data or user interaction in the studies. The model proposed by [38] integrates user behavior data through a user interaction recording system. This system captures the interactions of users and testers with the application for effectively prioritizing test cases. The interactions are recorded in a structured manner, allowing the model to analyze the actions users take during their engagement with the application. The recorded data includes details such as activity names and IDs of buttons clicked, providing a

comprehensive log of user interactions from the login state to the logout state. However, this approach does not address the multimodal nature of Android applications. Multimodal interaction would involve capturing various input forms beyond just button clicks and activity names, such as gestures, voice commands, or environmental contexts. The model described here seems focused on capturing specific user actions in a single mode (clicks/taps), rather than incorporating diverse types of interaction data from multiple input channels.

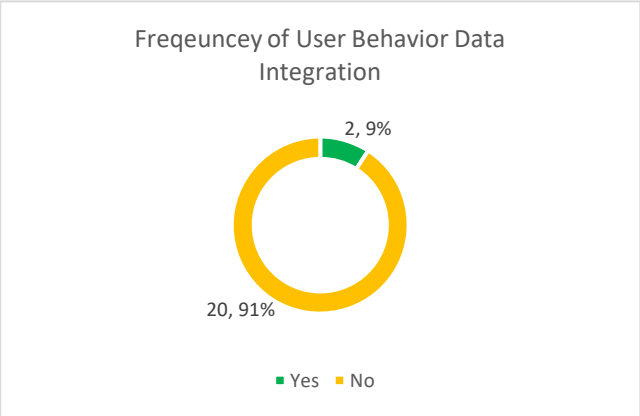

Fig. 8 Frequency of User Behavior Data Integration

The CamDroid [25] model incorporates user behavior data to enhance its testing capabilities. Specifically, it utilizes historical context knowledge from previous testing runs. This context-aware approach allows it to generate more realistic text inputs and select events based on the patterns of user behavior observed during prior tests. Since this paper focuses on generating inputs and selecting events based on historical patterns, likely in a single mode (such as text inputs), it does not address Android applications' complex and vast state space problem.

While the remaining 20 papers did not incorporate user behavior data or user interaction into their proposed models, several studies recognize its relevance and potential impact. For instance, a survey of DroidbotX [34] notes that its current implementation does not fully capture all possible behaviors during exploration and expresses a commitment to extending its capabilities in the future. [39] referenced [38], whose work collects sequence patterns from user interactions to prioritize test cases. This suggests that although the current model does not integrate such data, there is an acknowledgment of its importance and potential application in enhancing test case prioritization strategies in future research work.

IV. Conclusion

To summarize, this systematic literature review explores and evaluates the application of reinforcement learning in software testing of Android applications by examining a mass-researched area of studies, prevalent techniques, and emerging trends. The review reveals that automated testing is currently the primary focus in software testing, with Q-learning dominating the method utilized. However, the study also identifies significant underexplored regions, such as test case prioritization and the integration of user behavior data in Android applications, which present promising opportunities for future research. Addressing these gaps could lead to more robust approaches in Android application testing, ultimately advancing the field of software testing.

Acknowledgment

This work was supported by the Ministry of Higher Education, Malaysia (MOHE) under the Fundamental Research Grant Scheme (FRGS/1/2024/ICT01/UTM/02/1). The full article acknowledges this research: “Communication of this research is made possible through monetary assistance by Universiti Tun Hussein Onn Malaysia and the UTHM Publisher’s Office via Publication Fund E15216”.

References

[1] A. Abo-eleneen, A. Palliyali, and C. Catal, "The role of reinforcement learning in software testing," *Inf. Softw. Technol.*, vol. 165, Dec. 2023, doi: 10.1016/j.infsof.2023.107325.

[2] K. Sugali, C. Sprunger, and V. N. Inukollu, "Software testing: Issues and challenges of artificial intelligence & machine learning," *Int. J. Artif. Intell. Appl.*, vol. 12, no. 1, pp. 101-112, Jan. 2021, doi:10.5121/ijaia.2021.12107.

[3] H. You et al., "Navigating the testing of evolving deep learning systems: An exploratory interview study," in *Proc. IEEE/ACM 47th Int. Conf. Softw. Eng. (ICSE)*, 2025, p. 643, doi:10.1109/ICSE55347.2025.00106.

[4] M. Idham et al., "Test case prioritization for software product line: A systematic mapping study," *JOIV Int. J. Inf. Vis.*, vol. 7, p. 2126, Nov. 2023, doi: 10.30630/joiv.7.3-2.1340.

[5] J. Wang et al., "Software testing with large language models: Survey, landscape, and vision," *IEEE Trans. Softw. Eng.*, vol. 50, no. 4, pp. 911-936, Apr. 2024, doi: 10.1109/TSE.2024.3368208.

[6] S. Haldar, M. Pierce, and L. F. Capretz, "Exploring the integration of generative AI tools in software testing education: A case study on ChatGPT and Copilot for preparatory testing artifacts in postgraduate learning," *IEEE Access*, vol. 13, 2025, doi:10.1109/access.2025.3545882.

[7] Y. Matsuo et al., "Deep learning, reinforcement learning, and world models," *Neural Netw.*, vol. 152, pp. 267-275, Aug. 2022, doi:10.1016/j.neunet.2022.03.037.

[8] M. Binjubier et al., "A GPU accelerated parallel genetic algorithm for the traveling salesman problem," *J. Soft Comput. Data Min.*, vol. 5, no. 2, pp. 137-150, 2024, doi: 10.30880/jscdm.2024.05.02.010.

[9] F. Pecorelli et al., "Software testing and Android applications: A large-scale empirical study," *Empir. Softw. Eng.*, vol. 27, no. 2, Mar. 2022, doi: 10.1007/s10664-021-10059-5.

[10] Z. Chen et al., "Exploring better black-box test case prioritization via log analysis," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, Apr. 2023, doi: 10.1145/3569932.

[11] X. Li, "GUI testing for Android applications: A survey," in *Proc. 7th Int. Conf. Comput., Softw. Model. (ICCSM)*, 2023, pp. 6-10, doi:10.1109/iccsm60247.2023.00010.

[12] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A gentle introduction to reinforcement learning and its application in different fields," *IEEE Access*, vol. 8, pp. 209320-209344, 2020, doi:10.1109/access.2020.3038605.

[13] P. S. N. Mindom, A. Nikanjam, and F. Khomh, "Integrating reinforcement learning in software testing automation: A promising approach," *Empir. Softw. Eng.*, 2023, doi: 10.5753/ise.2023.235976.

[14] S. Elbaum, G. Rothermel, and J. Penix, "Techniques for improving regression testing in continuous integration development environments," in *Proc. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2014, pp. 235-245, doi: 10.1145/2635868.2635910.

[15] H. A. M. Shaffril, S. F. Samsuddin, and A. A. Samah, "The ABC of systematic literature review: The basic methodological guidance for beginners," *Qual. Quant.*, vol. 55, no. 4, pp. 1319-1346, Aug. 2021, doi: 10.1007/s11135-020-01059-6.

[16] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, UK, Tech. Rep. EBSE-2007-01, 2007.

[17] H. Eljak et al., "E-learning-based cloud computing environment: A systematic review, challenges, and opportunities," *IEEE Access*, vol. 12, pp. 7329-7355, 2024, doi: 10.1109/access.2023.3339250.

[18] C. Tao et al., "A reinforcement learning-based approach to testing GUI of mobile applications," *World Wide Web*, vol. 27, no. 2, Mar. 2024, doi: 10.1007/s11280-024-01252-9.
[19] E. Collins et al., "Deep reinforcement learning based Android application GUI testing," in *Proc. ACM Int. Conf. Proceeding Ser.*, 2021, pp. 186-194, doi: 10.1145/3474624.3474634.
[20] L. Cai et al., "Automated testing of Android applications integrating residual network and deep reinforcement learning," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, 2021, pp. 189-196, doi:10.1109/qrs54544.2021.00030.
[21] L. Cai et al., "Reinforcement learning application testing method based on multi-attribute fusion," in *Proc. 9th Int. Conf. Dependable Syst. Appl. (DSA)*, 2022, pp. 24-33, doi: 10.1109/DSA56465.2022.00013.
[22] Y. Zhao, B. Harrison, and T. Yu, "DinoDroid: Testing Android apps using deep Q-networks," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, Jun. 2024, doi: 10.1145/3652150.
[23] H. Guo et al., "SQDroid: A semantic-driven testing for Android apps via Q-learning," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. (QRS)*, 2021, pp. 301-310, doi: 10.1109/qrs54544.2021.00041.
[24] Y. P. Lopez et al., "Q-funcT: A reinforcement learning approach for automated black box functionality testing," in *Proc. 2nd IEEE Int. Conf. Softw. Eng. Artif. Intell. (SEAI)*, 2022, pp. 119-123, doi:10.1109/seai55746.2022.9832177.
[25] H. Wang et al., "CamDroid: Context-aware model-based automated GUI testing for Android apps," *Tsinghua Sci. Technol.*, vol. 30, no. 1, pp. 55-67, Feb. 2024, doi: 10.26599/tst.2024.9010038.
[26] Y. Lan et al., "Deeply reinforcing Android GUI testing with deep reinforcement learning," in *Proc. Int. Conf. Softw. Eng.*, Feb. 2024, doi: 10.1145/3597503.3623344.
[27] M. Pan et al., "Reinforcement learning based curiosity-driven testing of Android applications," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2020, pp. 153-164, doi:10.1145/3395363.3397354.
[28] Z. Lv et al., "Fastbot2: Reusable automated model-based GUI testing for Android enhanced by reinforcement learning," in *Proc. ACM Int. Conf. Proceeding Ser.*, Sep. 2022, doi: 10.1145/3551349.3559505.
[29] K. Murase and S. Takada, "Applying reinforcement learning for automated testing of mobile application focusing on state definition, reward, and learning method," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, 2023, pp. 64-69, doi: 10.18293/SEKE23-154.
[30] C. Peng et al., "Hawkeye: Change-targeted testing for Android apps based on deep reinforcement learning," in *Proc. ACM Int. Conf. Proceeding Ser.*, Apr. 2024, pp. 298-308, doi:10.1145/3639477.3639749.
[31] Y. Gao et al., "A deep reinforcement learning-based approach for Android GUI testing," in *Lect. Notes Comput. Sci.*, vol. 13975, pp. 262-276, 2023, doi: 10.1007/978-3-031-25201-3_20.
[32] F. Wang, C. Tao, and J. Gao, "REDQT: A method for automated mobile application GUI testing based on deep reinforcement learning algorithms," *Softw. Syst. Model.*, 2024, doi: 10.1007/s11761-024-00413-y.
[33] Z. Zhang et al., "UniRLTest: Universal platform-independent testing with reinforcement learning via image understanding," in *Proc. 31st ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2022, pp. 805-808, doi: 10.1145/3533767.3543292.
[34] H. N. Yasin, S. H. A. Hamid, and R. J. R. Yusof, "Droidbotx: Test case generation tool for android applications using q-learning," *Symmetry*, vol. 13, no. 2, Feb. 2021, doi: 10.3390/sym13020310.
[35] M. K. Khan and R. Bryce, "Android GUI test generation with SARSA," in *Proc. IEEE 12th Annu. Comput. Commun. Workshop Conf.*, 2022, pp. 487-493, doi: 10.1109/ccwc54503.2022.9720807.
[36] A. Usman et al., "Test case generation approach for Android applications using reinforcement learning," *Eng., Technol. Appl. Sci. Res.*, vol. 14, no. 4, pp. 15127-15132, Aug. 2024, doi:10.48084/etasr.7422.
[37] Y. Koroglu and A. Sen, "Functional test generation from UI test scenarios using reinforcement learning for android applications," *Softw. Test., Verif. Reliab.*, May 2021, doi:10.1002/stvr.1752.
[38] M. Waqar et al., "Test suite prioritization based on optimization approach using reinforcement learning," *Appl. Sci.*, vol. 12, no. 13, Jul. 2022, doi: 10.3390/app12136772.
[39] M. K. Khan et al., "Post prioritization techniques to improve code coverage for SARSA generated test cases," in *Proc. IEEE 13th Annu. Comput. Commun. Workshop Conf.*, 2023, pp. 1029-1035, doi:10.1109/CCWC57344.2023.10099120.