# Test Case Generation Approach for Android Applications using Reinforcement Learning

**Asmau Usman**
Department of Computer Science, Abdu Gusau Polytechnic Talata Mafara, Nigeria | Department of Computer Science, Faculty of Computing, Nile University of Nigeria, Nigeria
asmee08@gmail.com (corresponding author)

**Moussa Mahamat Boukar**
Department of Computer Science, Faculty of Computing, Nile University of Nigeria, Nigeria
musa.muhammed@nileuniversity.edu.ng

**Muhammed Aliyu Suleiman**
Department of Software Engineering, Faculty of Computing, Nile University of Nigeria, Nigeria
muhammad.suleiman@nileuniversity.edu.ng

**Ibrahim Anka Salihu**
Department of Software Engineering, Faculty of Computing, Nile University of Nigeria, Nigeria
ibrahim.salihu@nileuniversity.edu.ng

*Received: 5 April 2024 2024 | Revised: 25 April 2024 | Accepted: 27 April 2024*

*Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.7422*

## ABSTRACT

**Mobile applications can recognize their computational setting and adjust and respond to actions in the context. This is known as context-aware computing. Testing context-aware applications is difficult due to their dynamic nature, as the context is constantly changing. Most mobile testing tools and approaches focus only on GUI events, adding to the deficient coverage of applications throughout testing. Generating test cases for various context events in Android applications can be achieved using reinforcement learning algorithms. This study proposes an approach for generating Android application test cases based on Expected State-Action-Reward-State-Action (E-SARSA), considering GUI and context events for effective testing. The proposed method was experimentally evaluated on eight Android applications, showing 48-96% line of code coverage across them, which was higher than Q-testing and SARSA.**

***Keywords-Android applications; test case generation; GUI event; context event; reinforcement learning; expected SARSA***

## I. INTRODUCTION

Mobile devices have become a significant part of our daily lives, allowing easy access to communication, banking, social media, etc. The popularity of smartphones is ever-increasing due to their portability and increasing capabilities [1]. Mobile applications are now regularly used for numerous tasks in business, education, and healthcare, due to their growth in functionality and compatibility [2-4]. According to [5], an average person uses mobile phones for 3 hours a day and spends 90% of this time on mobile applications. In [6], it was stated that a mobile device has between 60 and 90 applications installed on average. Almost all global mobile application developers target Android as their first choice [7]. In line with the robust reputation of mobile applications, their quality is very important [8, 9]. Users often face denials or crashes in applications due to low quality and deficiencies in mobile testing [10]. Therefore, inspecting an application's reliability is an important task.

Mobile applications can now recognize their computational context and adjust and respond to actions in that context. A context can be seen as any information that can be used to characterize the situation of an entity [11]. An entity can be anything that is considered relevant to the interaction between a user and an application, such as a place, person, or object, including the user and the applications themselves [12]. Context-aware applications are increasingly popular because they can simplify end-user responsibilities in many sectors, such as entertainment, healthcare, and smart homes [13, 14]. Continuous variation in context makes testing context-aware applications a challenging task. Therefore, software engineers must increase the quality of their products by considering context variations and thoroughly testing their applications to detect faults [15]. A test engineer needs context data and a sequence of actions performed by the end user, which increases the difficulty of developing test cases [14, 16]. Some of the challenges of testing context-aware applications are: how to identify context events during testing, generating context-aware test data and test cases, and testing a wide variety of context data types and context variations [17-21]. Currently, only a few testing techniques and tools address testing context variations in mobile applications [14, 22, 23].

Many studies have recently proposed Machine Learning (ML) algorithms to automate software testing [24, 25]. ML developments have shown that automation can match or exceed human performance in different domains [26]. Recently, ML models have been proposed to directly generate input [27], automate functional tests [28], or test Android applications [29, 30]. ML offers the possibility of adapting test case generation for mobile applications. In [31], it was stated that of more than 600 Android application projects, only 14% had test cases, and approximately 9% had code coverage above 40%. Although previous studies have proposed testing models for Android application GUIs, more studies are needed to increase code coverage. Most mobile application testing techniques focus on testing only GUI events. This adds to the deficient coverage of the applications throughout testing. Generating test cases for variations in context events in Android applications can be achieved using Reinforcement Learning (RL) ML algorithms. This study proposes an approach to generate Android application test cases based on Expected State-Action-Reward-State-Action (E-SARSA), considering GUI and context events for effective testing.

## II. BACKGROUND AND RELATED WORKS

### *A. Reinforcement Learning (RL)*

RL is a branch of ML that primarily emphasizes successive policy-making that takes into account uncertainties. According to [32, 33], RL can be seen as a computational method for ML that follows behavioral psychology concepts and learning from connections. The application of RL in Android application testing has attracted the interest of many researchers [34]. RL has two mechanisms: the environment and the agent. An agent is an independent unit that can perform activities in an environment to reach a goal. The RL concept examines how software agents should take action in an environment in which the cumulative reward is maximized. The reward is given only when the agent achieves an objective [35]. The purpose of RL is to learn the optimal policies for agents that interact with an unidentified environment *E*. The goal of an agent is to choose a sequence of actions, by observing the environment, that maximizes the cumulative reward over all time steps through trial and error. RL has become increasingly popular due to its success in addressing challenging and successive policy-making issues. Q-learning is an off-policy method that updates its $Q$ values using the following update rule:

$$Q(s_t, e_t) \leftarrow Q(s_t, e_t) + \alpha\,[R(s_t, e_t) + \gamma\,\mathrm{max}a\; Q(s_{t+1}, e) - Q(s_t, e_t)] \quad (1)$$

The max operator makes the estimation policy greedy, ensuring the $Q$ values converge to $Q(s_{t+1}, e)$. The behavior policy of Q-learning is exploratory and based on $Q(s_t, e_t)$. The behavioral and estimated policies are equal in SARSA, which updates the $Q$ value using the following rule:

$$Q(s_t, e_t) \leftarrow Q(s_t, e_t) + \alpha\,[R(s_t, e_t) + \gamma\, Q(s_{t+1}, e_{t+1}) - Q(s_t, e_t)] \quad (2)$$

SARSA will not congregate to optimal $Q$ values as long as exploration occurs, because it is on policy. However, SARSA's convergence requires each state to be visited recurrently, and the behavior and the approximation policy are classically stochastic to certify suitable exploration. E-SARSA is a variation of SARSA that decreases update changes. The agent acts, perceives the reward, and updates the $Q$-value. It does so by updating the action using the expected $Q$-value $Q(s_{t+1}, e)$, instead of merely using $e_{t+1}$. Using this probability decreases update changes. According to [36], the $Q$-value function is defined as:

$$Q(s_t, e_t) \leftarrow Q(s_t, e_t) + \alpha(R(s_t, e_t) + \gamma \Sigma_e \pi(e|s_{t+1}) Q(s_{t+1}, e) - Q(s_t, e_t)) \quad (3)$$

where $Q(s_t, e_t)$ on the left is the new Q-value of $e$ later in performing $e_t$ and moving to state $s_{t+1}$. $Q(s_t, e_t)$ on the right side is the current $Q$-value of event $e_t$ in state $s_t$, and $\alpha$ is the learning rate that signifies the effect of an original remark on the predicted value of the $Q$-function. $R(s_t, e_t)$ is the instant reward for taking event $e_t$ in state $s_{t+1}$, and $\gamma$ is the discount factor. When $s_{t+1}$ executes an update of $Q(s_t, e_t)$, it will also update $Q(s_{t+1}, e)$ through an improved estimate before choosing the action. $\Sigma_e\, \pi(e|s_{t+1},)$ is the weighted sum of all possible next events, and $e$ is the $Q$-value of the next selected event $e_t$ in the state $s_{t+1}$.

### *B. Related Works*

Several studies have used RL algorithms to test GUI, such as enhancing branch coverage for C applications [27], Java/Swipe desktop applications [37], and desktop and web applications [38]. These tools are not appropriate for testing mobile applications due to their exceptional features, such as screen size, operating system versions, input procedures, and interaction devices [39]. Therefore, focus was given to previous studies related to testing Android applications using RL techniques. In [40], a method was presented that allowed the classification of each activity into a specific type and was used to test several expected behaviors on different screens. In [41], Q-Learning-Based Exploration (QBE) was proposed to explore GUI actions using Q-Learning. QBE performed better than Sapienz, Monkey, A3E, and Swifthand in terms of activity and instruction coverage, except Sapienz, which achieved higher instruction coverage. This method also achieved better results in terms of the number of crashes detected. In [42], a black-box testing tool was proposed that adopted the Advantage Actor-Critic (A2C) algorithm to automatically generate test cases. The algorithm comprises an actor (policy) and a critic (value function). This method was tested on 17 Android applications and was compared with Monkey and ARES. The results showed that this method achieved higher code coverage and detected more errors.

In [43], DroidbotX was proposed, which is an approach that generates a GUI test case based on Q-learning. DroidbotX uses tabular Q-learning and an effective exploration strategy to minimize redundant action executions while using different states and action spaces. DroidbotX outperformed Android Monkey, Droidbot, Sapienz, Stoat, and Humanoid, as it achieved better test coverage and triggered more crashes. In [39], a test case generation method was proposed using SARSA. This method employed a similar policy to both choose an action and update the $Q$ value. In [28], Q-funcT was proposed, which used Q-learning to learn the best policy and implement dissimilar functions in the environment rather than exploring the application to search for errors. In [44] AimDroid was presented to automatically test Android applications based on SARSA, aiming to investigate activities and minimize redundant changes between them.

## III. METHODOLOGY

The proposed approach generates test cases from Android applications focusing on both context and GUI events. This approach uses E-SARSA to generate the test, where the Application Under Test (AUT) serves as the agent's environment. UIautomator [45] is used to extract the XML sign of the AUT GUI and learn the accessible widgets and actions recognized by a unique ID. Testing events for mobile applications have several challenges. Generating test cases for variations of context events in an Android application using E-SARSA involves dealing with changing environmental conditions and accordingly adapting the testing strategy. The main part of E-SARSA is avoiding stochasticity in the policy by using more cumulative variance [36]. It does so by creating an update on the expected value $Q(s_{t+1}, e_{t+1})$.

The agent implements the selected event, perceives the reward, and updates the Q-value function. Each state uses data that aid the agent in choosing the next event. In RL, the sequence is identified as an episode. In each episode, the agent changes the $Q$-value until it reaches the next state. Thus, the approach creates a reward function that considers the exploration of the sequence state and execution of the action/events (GUI and context). Events, states, and actions are defined according to the Markov Decision Process (MPD), as shown in (4), to generate a test case for an Android application.

$$M := (S, E, T, \gamma, P, R) \quad (4)$$

where:

- $S$ is a set of states that is defined by the activity name and the set of available events (GUI/context). A state $s$ is represented by an $n$-tuple, as:

  $$s = (e_1, e_2, e_3, e_4, \dots, e_n) \quad (5)$$

  where $e_i$ is the current event and $n$ is the total number of events.

- $E$ is the set of events known as the actions in MPD. An event resembles an action that can be executed on a GUI component (e.g. a swipe button), or context event (e.g. Bluetooth or a sensor). There is no variance between events and actions. The event $e$ is represented by a 3-tuple, as:

  $$e = (w, e_t, v) \quad (6)$$

  where $w$ is a widget on a particular screen, $e_t$ is the event type, which can be either click, long click, or swipe, and $v$ grasps random text if the widget is a text field or null if the widget is non-text. Each event is linked to an exact state, which allows also practicing the state-action pairs to characterize an event executable in the application state.

- $R$ is the distribution of rewards that returns a numeric value that specifies a transition (current state, event, new state). The reward function calculates the result of taking the action so that the test generation algorithm can differentiate between the actions that were explored earlier.

- $T$ is the transition function. When $e_t$ is executed in the current state, the transition to a new state will be determined in response to the AUT. This defines the state the application will reach after the execution of an action, and it is resolute by the application.

- $\gamma$ is the discount factor. The agent uses trial-and-error relations to gain information about the environment, directed by positive or negative rewards rather than obvious commands.

ALGORITHM 1: E-SARSA TEST GENERATION

```
1. Input: AUT: Application under test, C: Completion
     Criteria, T: Transition, E: Events, S: States,
     Q: Q-functions, IQ: Initial q-value
2. Output: TC: Test Case
3. Start AUT
4.   while C ≠ true do
5      state_now ← getcurrentstate(AUT)
6.     events e ← env(possible events)
7.     foreach e ϵ E do
8.       event_id ← get_event_id(e)
9.       Event_type ← get_event_type(et)
10.      update S, statenow, Eids
11.      return (S, Eids)
12.     End foreach
13.     Update Q-function
14.     nxtstate_prob ← P(e/next_state)
15.     qNxtstate ← qValues
16.     for each e in Nxtstate
17.       if qCurrent is none, then
18.         qValue(s, e) ←R
19.       else
20.         qValues(s,e)←q_current+0.1*R+
              0.99*Σe (nxtstate) − qValue(e, s)
21.         return(nxtstate)
22.      End foreach
23.      for all s in S do
24.        visit all unexplored A in s
25.        if E is not empty do
26.          choose e from E according to T
27.        else
28.          extract s from unexplored e
29.        repeat until all E in S are explored
30.      End for
31.    End update q function
32.    And TC is CC
33.  End while
34. End AUT
```

E-SARSA updates its weights by learning from the transitions in the table. The approach gradually interrelates with the AUT to approximate a behavioral model of the application. The proposed approach aims to generate test cases that maximize code coverage. The execution of a sequence of actions in the AUT results in the formation of a test case. Algorithm 1 illustrates the algorithm the test case generation method based on E-SARSA. This algorithm receives as input: AUT, completion criteria, transition function, actions, states, Q-functions, and initial q-value. From these input parameters, this approach performs a sequence of actions and outputs a test case. The completion criterion is to visit all the unexplored states. The procedure starts at line 3 of the algorithm.

## IV. EXPERIMENTAL SETUP

### *A. Environment*

All experiments were run on a PC with Ubuntu 20.04, equipped with 8GB of RAM and an Intel i5 processor. An emulator was used to test the mobile app, based on x86 Intel with Android version 10.0 'Q' (API level 29).

### *B. Benchmark*

The proposed approach was evaluated on 8 open-source Android applications of different categories and sizes. The applications were obtained from F-droid [46], a frequently used database for downloading Android applications. Table I shows the characteristics of the selected applications, such as versions, categories, number of downloads, and Lines of Code (LOC). LOC varied from 273 to 29,126, with an average of 6,240.

TABLE I. CHARACTERISTICS OF SELECTED APPLICATIONS

| **Applications** | **Categories** | **Version** | **#Downloads** | **LOC** |
|---|---|---|---|---|
| Alarmclock | Productivity | 2.11 | 10 million+ | 1,201 |
| Alogcat | Tools | 2.6.1 | 100 thousand+ | 974 |
| Ankidroid | Education | 2.16.5 | 10 million+ | 29,126 |
| A2dp | Map and Navigation | 2.13 | 100 million+ | 4,522 |
| Munchlife | Entertainment | 1.4.4 | 10 thousand+ | 273 |
| Simple Reminder | Tools | 2.8.1 | 1 thousand+ | 1,126 |
| The Kana Quiz | Education | 0.15 | 500+ | 4,453 |
| TrickyTripper | Travel and Local | 1.6.2 | 10 thousand+ | 8,244 |
| Average | | | | 6,240 |

## V. RESULTS AND DISCUSSION

Code coverage was used to check the efficiency of the proposed method on the selected applications. Code coverage is the sum of passes that have been executed at least once as a percentage of the total number of passes. LOC coverage measures the percentage of lines executed during testing compared to the total number of lines of code in the application. There are several tools for measuring code coverage for different programming languages, such as Jacoco [47], coverage.py [48], and Emma [49]. This study used Emma, an open-source code coverage tool, to measure the coverage achieved by the proposed approach.

The proposed approach was compared with other approaches/techniques in the literature, such as Q-testing [50], and SARSA [39]. As each approach was evaluated using a different set of mobile applications, the comparison was performed by considering the applications used to evaluate each approach. Based on this, this approach was compared separately with the others. Figure 1, shows the LOC coverage obtained by the proposed approach, with approximately 50% on Alarmclock, 79% on Ankidroid, 90% on Alogcat, and 96% on MunchLife. Q-testing achieved approximately 36% on Alarmclock, 32% on Ankidroid, 78% on Alogcat, and 90% on MunchLife respectively. Figure 2, shows the LOC coverage obtained by the proposed approach compared to SARSA. SARSA achieved a range of 39.60% on Ankidroid, 67.18% on SimpleReminder, 63.30% on the Kana Quiz, and 36.61% on TrickyTripper, respectively.

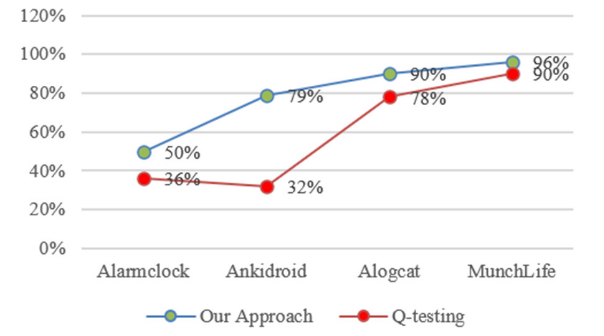


Fig. 1. Comparison of the proposed approach with Q-testing.

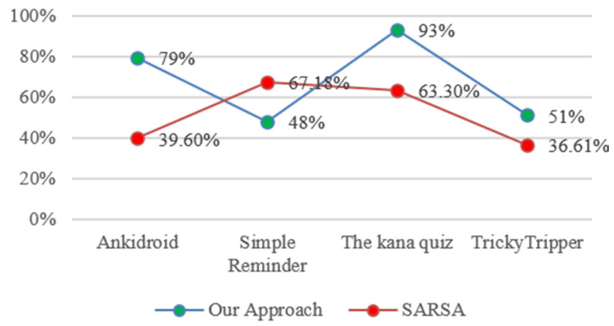


Fig. 2. Comparison of the proposed approach with SARSA.

Based on the average coverage of the tools compared using the selected applications, the proposed approach outperformed the others in terms of LOC, as shown in Figure 3. The proposed approach achieves these results due to the use of E-SARSA, which reduces changes by taking into account the probability of each action under the current policy, triggered by randomly selecting actions.

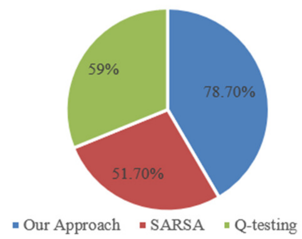


Fig. 3. Comparison of LOC average.

## VI. CONCLUSION

Mobile context-aware application testing is complicated due to its dynamic nature as the context changes. Most mobile testing tools and approaches focus only on GUI events. This adds to the deficient coverage of applications throughout testing. This study presented an approach to generate Android application test cases using E-SARSA, which considers both context and GUI events for effective testing. An experimental analysis was executed using real-world open-source mobile applications to evaluate the approach. The results showed that the proposed approach had 48-96% coverage on the selected applications. The proposed method outperformed Q-testing on all four common applications and achieved better LOC coverage than SARSA on Ankidroid, The Kana Quiz, and TrickyTripper. However, SARSA had a better LOC on Simple Reminder by 19%. The proposed approach had an average LOC of 78.70% while Q-testing and SARSA had 59% and 51.70%, respectively. Future work will investigate additional evaluation metrics, such as fault detection ability, and integrate the approach into a tool.

## REFERENCES

[1] I. A. Salihu, R. Ibrahim, B. S. Ahmed, K. Z. Zamli, and A. Usman, "AMOGA: A static-dynamic model generation strategy for mobile apps testing," *IEEE Access*, vol. 7, pp. 17158–17173, Jan. 2019, https://doi.org/10.1109/ACCESS.2019.2895504.

[2] H. Muccini, A. Di Francesco, and P. Esposito, "Software testing of mobile applications: Challenges and future research directions," in *2012 7th International Workshop on Automation of Software Test (AST)*, Zurich, Switzerland, Jun. 2012, pp. 29–35, https://doi.org/10.1109/IWAST.2012.6228987.

[3] A. S. Alkarim, A. Al-Malaise Al-Ghamdi, and M. Ragab, "Ensemble Learning-based Algorithms for Traffic Flow Prediction in Smart Traffic Systems," *Engineering, Technology & Applied Science Research*, vol. 14, no. 2, pp. 13090–13094, Apr. 2024, https://doi.org/10.48084/etasr.6767.

[4] I. A. Salihu, R. Ibrahim, and A. Usman, "A Static-dynamic Approach for UI Model Generation for Mobile Applications," in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India, Aug. 2018, pp. 96–100, https://doi.org/10.1109/ICRITO.2018.8748410.

[5] D. Chaffey, "Statistics on consumer mobile usage and adoption to inform your mobile marketing strategy mobile site design and app development," Smart Insights, 2021.

[6] H. Packard, "Failing to meet mobile app user expectations: a mobile user survey". Technology Representatives Inc, 2015.

[7] "Mobile Developer Population Reaches 12M Worldwide, Expected to Top 14M by 2020". Evans Data Corporation, May 2016, https://evansdata.com/press/viewRelease.php?pressID=244.

[8] N. C. Eli-Chukwu, J. M. Aloh, and C. O. Ezeagwu, "A Systematic Review of Artificial Intelligence Applications in Cellular Networks," *Engineering, Technology & Applied Science Research*, vol. 9, no. 4, pp. 4504–4510, Aug. 2019, https://doi.org/10.48084/etasr.2788.

[9] I. A. Salihu and R. Ibrahim, "Systematic Exploration of Android Apps' Events for Automated Testing," in *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*, Singapore, Nov. 2016, pp. 50–54, https://doi.org/10.1145/3007120.3011072.

[10] A. Usman, N. Ibrahim, and I. A. Salihu, "Comparative Study of Mobile Applications Testing Techniques for Context Events," *Advanced Science Letters*, vol. 24, no. 10, pp. 7305–7310, Oct. 2018, https://doi.org/10.1166/asl.2018.12933.

[11] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci, "Considering Context Events in Event-Based Testing of Mobile Applications," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, Luxembourg, Mar. 2013, pp. 126–133, https://doi.org/10.1109/ICSTW.2013.22.

[12] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. E. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Handheld and Ubiquitous Computing, Proceedings*, H. W. Gellersen, Ed., Berlin, Germany: Springer, 1999, pp. 304–307.

[13] Z. A. Almusaylim and N. Zaman, "A review on smart home present state and challenges: linked to context-awareness internet of things (IoT)," *Wireless Networks*, vol. 25, no. 6, pp. 3193–3204, Aug. 2019, https://doi.org/10.1007/s11276-018-1712-5.

[14] A. M. Mirza *et al.*, "ContextDrive: Towards a Functional Scenario-Based Testing Framework for Context-Aware Applications," *IEEE Access*, vol. 9, pp. 80478–80490, May 2021, https://doi.org/10.1109/ACCESS.2021.3084887.

[15] A. Usman, N. Ibrahim, and I. A. Salihu, "Test Case Generation from Android Mobile Applications Focusing on Context Events," in *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, Kuantan, Malaysia, Feb. 2018, pp. 25–30, https://doi.org/10.1145/3185089.3185099.

[16] S. S. Priya and B. Rajalakshmi, "Testing Context Aware Application and its Research Challenges," in *2022 International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, Villupuram, India, Mar. 2022, pp. 1–7, https://doi.org/10.1109/ICSTSN53084.2022.9761331.

[17] M. A. Mehmood, M. N. A. Khan, and W. Afzal, "Automating Test Data Generation for Testing Context-Aware Applications," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, Nov. 2018, pp. 104–108, https://doi.org/10.1109/ICSESS.2018.8663920.

[18] A. Usman, N. Ibrahim, and S. Anka, "TEGDroid: Test Case Generation Approach for Android Apps Considering Context and GUI Events," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 1, pp. 16–23, Feb. 2020, https://doi.org/10.18517/ijaseit.10.1.10194.

[19] B. R. Siqueira, F. C. Ferrari, K. E. Souza, V. V. Camargo, and R. de Lemos, "Testing of adaptive and context-aware systems: approaches and challenges," *Software Testing, Verification and Reliability*, vol. 31, no. 7, pp. 1–46, May 2021, https://doi.org/10.1002/stvr.1772.

[20] S. Yue, S. Yue, and R. Smith, "A Survey of Testing Context-aware Software: Challenges and Resolution," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, Las Vegas, Nevada, USA, Sep. 2016, pp. 102–108.

[21] D. R. Almeida, P. D. L. Machado, and W. L. Andrade, "Testing tools for Android context-aware applications: a systematic mapping," *Journal of the Brazilian Computer Society*, vol. 25, no. 1, pp. 1–22, Dec. 2019, Art. no. 12, https://doi.org/10.1186/s13173-019-0093-7.

[22] A. M. Mirza and M. N. A. Khan, "An Automated Functional Testing Framework for Context-Aware Applications," *IEEE Access*, vol. 6, pp. 46568–46583, Aug. 2018, https://doi.org/10.1109/ACCESS.2018.2865213.

[23] T. B. Nguyen, T. T. B. Le, O. E. K. Aktouf, and I. Parissis, "Mobile Applications Testing Based on Bigraphs and Dynamic Feature Petri Nets," in *The First International Conference on Intelligence of Things (ICIT 2022)*, Hanoi, Vietnam, Aug. 2022, vol. 148, pp. 215–225, https://doi.org/10.1007/978-3-031-15063-0_20.

[24] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *Verification and Validation 2011 Fourth IEEE International Conference on Software Testing*, Berlin, Germany, Mar. 2011, pp. 1–10, https://doi.org/10.1109/ICST.2011.38.

[25] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Santa Barbara, CA, USA, Jul. 2017, pp. 12–22, https://doi.org/10.1145/3092703.3092709.

[26] B. Ahmed, G. Ali, A. Hussain, A. Baseer, and J. Ahmed, "Analysis of Text Feature Extractors using Deep Learning on Fake News," *Engineering, Technology & Applied Science Research*, vol. 11, no. 2, pp. 7001–7005, Apr. 2021, https://doi.org/10.48084/etasr.4069.

[27] J. Kim, M. Kwon, and S. Yoo, "Generating test input with deep reinforcement learning," in *Proceedings of the 11th International Workshop on Search-Based Software Testing*, Gothenburg, Sweden, May 2018, pp. 51–58, https://doi.org/10.1145/3194718.3194720.

[28] Y. P. López, J. G. Colonna, E. De Araujo Silva, R. H. Degaki, and J. M. Silva, "Q-funcT: A Reinforcement Learning Approach for Automated Black Box Functionality Testing," in *2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI)*, Xiamen, China, Jun. 2022, pp. 119–123, https://doi.org/10.1109/SEAI55746.2022.9832177.

[29] A. Romdhana, A. Merlo, M. Ceccato, and P. Tonella, "Deep Reinforcement Learning for Black-box Testing of Android Apps," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 4, pp. 1–29, Apr. 2022, Art. no. 65, https://doi.org/10.1145/3502868.

[30] Y. Zhao, B. Harrison, and T. Yu, "DinoDroid: Testing Android Apps Using Deep Q-Networks," *ACM Transactions on Software Engineering and Methodology*, Nov. 2024, https://doi.org/10.1145/3652150.

[31] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the Test Automation Culture of App Developers," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, Graz, Austria, Apr. 2015, pp. 1–10, https://doi.org/10.1109/ICST.2015.7102609.

[32] S. Thrun and M. L. Littman, "Reinforcement Learning: An Introduction," *AI Magazine*, vol. 21, no. 1, pp. 103–103, Mar. 2000.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 2nd ed.Cambridge, MA, USA: MIT Press, 2018.

[34] A. Usman, M. M. Boukar, M. A. Suleiman, I. A. Salihu, and N. O. Eke, "Reinforcement Learning for Testing Android Applications: A Review," in *2023 2nd International Conference on Multidisciplinary Engineering and Applied Science (ICMEAS)*, Abuja, Nigeria, Nov. 2023, vol. 1, pp. 1–6, https://doi.org/10.1109/ICMEAS58693.2023.10429864.

[35] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, Oct. 2021, Art. no. 105400, https://doi.org/10.1016/j.cor.2021.105400.

[36] H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering, "A theoretical and empirical analysis of Expected Sarsa," in *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, Nashville, TN, USA, 2009, pp. 177–184, https://doi.org/10.1109/ADPRL.2009.4927542.

[37] L. Mariani, M. Pezzè, O. Riganelli, and M. Santoro, "AutoBlackTest: a tool for automatic black-box testing," in *Proceedings of the 33rd International Conference on Software Engineering (ICSE11)*, Honolulu, HI, USA, May 2011, pp. 1013–1015, https://doi.org/10.1145/1985793.1985979.

[38] A. Esparcia-Alcazar, F. Almenar, M. Martınez, U. Rueda, and T. E. J. Vos, "Q-learning strategies for action selection in the TESTAR automated testing tool: 6th International Conference on Metaheuristic and Nature inspired Computing," in *Proceedings of the 6TH International Conference on Metaheuristics and Nature Inspired Computing META'2016*, Marrakech, Morocco, 2016, pp. 174–180.

[39] M. K. Khan and R. Bryce, "Android GUI Test Generation with SARSA," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, Jan. 2022, pp. 0487–0493, https://doi.org/10.1109/CCWC54503.2022.9720807.

[40] A. Rosenfeld, O. Kardashov, and O. Zang, "ACAT: A Novel Machine-Learning-Based Tool for Automating Android Application Testing," in *Hardware and Software: Verification and Testing*, Haifa, Israel, 2017, pp. 213–216, https://doi.org/10.1007/978-3-319-70389-3_14.

[41] Y. Koroglu *et al.*, "QBE: QLearning-Based Exploration of Android Applications," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Vasteras, Sweden, Apr. 2018, pp. 105–115, https://doi.org/10.1109/ICST.2018.00020.

[42] Y. Gao, C. Tao, H. Guo, and J. Gao, "A Deep Reinforcement Learning-Based Approach for Android GUI Testing," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM-2022) Joint International Conference on Web and Big Data*, Nanjing, China, Aug. 2022, pp. 262–276, https://doi.org/10.1007/978-3-031-25201-3_20.

[43] H. Yasin, S. H. A. Hamid, and R. J. R. Yusof, "DroidbotX: Test Case Generation Tool for Android Applications Using Q-Learning," *Symmetry*, vol. 13, Feb. 2021, Art. no. 310, https://doi.org/10.3390/sym13020310.

[44] T. Gu *et al.*, "AimDroid: Activity-Insulated Multi-level Automated Testing for Android Applications," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Shanghai, China, Sep. 2017, pp. 103–114, https://doi.org/10.1109/ICSME.2017.72.

[45] "Write automated tests with UI Automator," *Android Developers*. [Online]. Available: https://developer.android.com/training/testing/other-components/ui-automator.

[46] "F-Droid - Free and Open Source Android App Repository." [Online]. Available: https://f-droid.org/.

[47] "JaCoCo Java Code Coverage Library." [Online]. Available: https://www.eclemma.org/jacoco/.

[48] "coverage: Code coverage measurement for Python." [Online]. Available: https://github.com/nedbat/coveragepy.

[49] "EMMA: a free Java code coverage tool." https://emma.sourceforge.net/.

[50] M. Pan, A. Huang, G. Wang, T. Zhang, and X. Li, "Reinforcement learning based curiosity-driven testing of Android applications," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, Apr. 2020, pp. 153–164, https://doi.org/10.1145/3395363.3397354.