Information and Software Technology 164 (2023) 107325

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

# The role of Reinforcement Learning in software testing

Amr Abo-eleneen, Ahammed Palliyali, Cagatay Catal *

*Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar*

ARTICLE INFO

*Keywords:*
Software testing
Machine learning
Reinforcement Learning
Artificial intelligence

ABSTRACT

**Context:** Software testing is applied to validate the behavior of the software system and identify flaws and bugs. Different machine learning technique types such as supervised and unsupervised learning were utilized in software testing. However, for some complex software testing scenarios, neither supervised nor unsupervised machine learning techniques were adequate. As such, researchers applied Reinforcement Learning (RL) techniques in some cases. However, a systematic overview of the state-of-the-art on the role of reinforcement learning in software testing is lacking.
**Objective:** The objective of this study is to determine how and to what extent RL was used in software testing.
**Methods:** In this study, a Systematic Literature Review (SLR) was conducted on the use of RL in software testing, and 40 primary studies were investigated.
**Results:** This study highlights different software testing types to which RL has been applied, commonly used RL algorithms and architecture for learning, challenges faced, advantages and disadvantages of using RL, and the performance comparison of RL-based models against other techniques.
**Conclusions:** RL has been widely used in software testing but has almost narrowed to two applications. There is a shortage of papers using advanced RL techniques in addition to multi-agent RL. Several challenges were presented in this study.

## 1. Introduction

Software mostly contains different types of errors and faults. Errors are generally unavoidable due to the inherent characteristics of software developers. This is an inescapable outcome because humans develop the code and sometimes make numerous mistakes. Requirements can be ambiguous or inaccurate, they can be misinterpreted, software components can be abused, developers can make silly mistakes while developing code, and hence, previously functioning programs may become inoperable. Software testing is a natural reaction to this issue. While the theory seems straightforward at first glance, testing software-intensive systems effectively is challenging. One of the primary reasons for this is the vast number of required tests for every nontrivial system, even for small-scale systems. Moreover, according to [1], each produced or changed software system must pass stringent testing to assure product quality. Moreover, designing test cases that check the correctness of the system is both time-consuming and expensive [2]. As presented in [3–6], software testing consumes almost half of the overall resources, a third of the entire working time, and more than 50% of the software development's overall charge. As such, test automation has been introduced to automate the process of test case generation. The authors in [2] showed the benefits of using automated testing against manual testing and concluded that both time and cost dimensions were improved.

However, for some applications where automated test case generation required more intelligence, machine learning-based approaches were introduced. Machine learning, mainly supervised and unsupervised learning, has gained a lot of attention to improve, ease, and automate the essential tasks of software validation and verification. These tasks include fuzzing, mutation, test input generation, and others. For complex scenarios such as the cases where software has a nearly infinite number of states [7], neither supervised nor unsupervised machine learning methods were adequate because the software testing task requires more intelligent and sophisticated actions. In this regard, reinforcement learning has been introduced as one of the promising areas to explore for software testing.

In this paper, a Systematic Literature Review (SLR) study is performed to show the current progress of using reinforcement learning (RL) in software testing. This paper is distinct from other types of literature review papers because we conduct a systematic search on electronic databases, retrieve related articles systematically, extract the required data, and synthesize them to address the research questions. The SLR process requires researchers to investigate the primary papers

* Corresponding author.
*E-mail addresses:* aa1405465@student.qu.edu.qa (A. Abo-eleneen), ap1304567@student.qu.edu.qa (A. Palliyali), ccatal@qu.edu.qa (C. Catal).

https://doi.org/10.1016/j.infsof.2023.107325
Received 13 August 2022; Received in revised form 12 June 2023; Accepted 30 August 2023
Available online 9 September 2023
0950-5849/© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

published on a certain subject in different electronic databases. For this purpose, we formulate several research questions and seek the literature to respond to them clearly. In addition, we discuss the results and present the gaps that need further investigation. To the best of our knowledge, this is the first SLR paper that surveys the use of reinforcement learning in software testing.

The remaining sections are organized as follows: An introduction to reinforcement learning basic concepts and motivation is presented in Section 2. Secondly, in Section 3, we formulate and design the research questions for the study and present the research methodology. In Section 4, we present the results followed by a discussion of the responses to the research questions in Section 5. Finally, we conclude the paper in Section 6.

## 2. Background and related work

### 2.1. Software testing

A test case is a collection of input data used to execute the program under test. An input to a program can cause different interactions. For functional testing, the function's arguments are the inputs. In the case of an Android application, the inputs are user interface events. If a network protocol is to be debugged, then the inputs would be the network packets. In addition, in the case of a word processing program, the inputs might be either user interactions or XML documents. To validate the observed behavior of the program under test, test cases must include these inputs and a test oracle that defines an expected output value or behavior that corresponds to the software's actual output. Test suite refers to a collection of test cases or simply a test set.

Since software testing is often conducted manually, it is preferable to automate this process. The advantages of such an action imply that the testing could be repeated indefinitely. Secondly, it eliminates the potential source of mistakes (i.e., humans) throughout repeated executions. Thirdly, it is significantly faster to run the test. A test case requires a test driver to automate execution. The test driver is responsible for getting the system under test (SUT) into the appropriate condition, applying the input and checking the output against the test oracle. A typical method of developing test drivers is through programming languages. For example, scripting languages such as Python are well-suited for developing complicated testing frameworks and utilizing the existing testing frameworks, particularly the xUnit frameworks, which is a common option. xUnit term is typically used to refer to unit testing frameworks for various programming languages such as JUnit or PyUnit for Java and Python, respectively.

Software testing and optimization are active research topics for a long time. The techniques discovered are presented in Fig. 1 and can be categorized into three major categories as follows:

- **Analysis testing**: Groups of tests that are used for analyzing the code and test cases. This category contains all code/specification coverage tests such as logical and graphical user interface coverage for testing a user interface (UI). In addition, mutation testing was also introduced to reinforce the quality of the generated test cases.
- **Test execution**: The group of techniques used for executing tests efficiently. Most of these techniques adopt the idea of automated testing, where performing the test cases, minimizing them, or re-running them are done automatically. Automated testing offers a reduction in both cost and time and is widely supported by different tools and frameworks. The key approach families include test case minimization, test case prioritization, and test case regression selection. All of these approaches reduce both the time and cost of human intervention.
- **Test Generation**: Creating test cases and then generating the input for the created test cases can be time-consuming since some parts of the source code are unreachable and, thus, untestable. Many researchers focused on this area by introducing random testing that incorporates randomness in creating both test cases and test inputs as in the case of the fuzzing technique. More intelligent methods adapted and strengthened the idea of random testing, such as mutation-based test generation. In addition to random testing, other methods were included to reduce the search space of valid inputs by studying the input space of each code partition and providing some inputs that can test that section such as equivalence partitioning and boundary value testing techniques. However, these methods were insufficient to cover some corner cases in the code that require specific input; thereby, more complex techniques such as symbolic execution were developed.

### 2.2. Machine learning

Large datasets and high-quality data can significantly improve the accuracy of machine learning models. Additionally, it is essential to use appropriate algorithms to tackle multiple issues, especially those involving various types of datasets. Machine learning is the general umbrella of various learning methods such as supervised, unsupervised, semi-supervised, and reinforcement learning, which is presented in Fig. 2.

In Supervised Learning (SL), a machine learning algorithm is provided with some training datasets, which contain input features and a prediction value. The algorithm aims to estimate a specific function (i.e., hypotheses function) that maps the input to the output. The SL categorizes the type of operation according to the output value. If the predicted value is a number, the process is called regression. On the other hand, if the predicted output is a class label, the operation is known as classification. Both regression and classification tasks can be addressed with many algorithms. For example, there are linear regression, polynomial regression, Lasso regression, and others under the regression family. Similarly, in classification, Naive Bayes [9], Support Vector Machines (SVM) [10], and Discriminant Analysis [11] are some examples. The problem with SL is that human intervention is crucial. Not only do people label the output for the training set, but they also select attributes, algorithms, and hyperparameters. Supervised learning is typically employed in domains where the human model requires specific knowledge and skill. However, the SL approach necessitates additional data processing for feature selection and anticipates parameter tweaking for optimal algorithm configuration.

In Semi-Supervised Learning (SSL), there are very few labeled data (e.g., 10%–20%) and the rest of the data are unlabeled. The idea in SSL is to use the unlabeled data together with labeled data because building a model with few labeled data is not feasible, and often leads to low-performance predictors, as such, in different iterations unlabeled data points receive some labels or values and the performance is improved using different iterations.

Unsupervised learning (UL), in contrast to SL, is a mathematical model that employs training datasets of input vectors and does not include a target variable. Typically, unsupervised learning provides the pattern of input variables and frequently presents various clusters formed from the input data. K-means clustering, Gaussian mixture models, and kernel density estimators are some examples of UL algorithms.

Recently, Deep Learning (DL) was introduced as a subset of machine learning that uses more advanced techniques than traditional shallow machine learning techniques. The concept of neurons and Multi-Layer Perceptron (MLP) topology existed before the DL was adopted widely. In DL networks, there are many hidden layers and different types of layers exist, such as a convolutional layer, pooling layer, and dropout layer. These new architectures led to better function approximations,

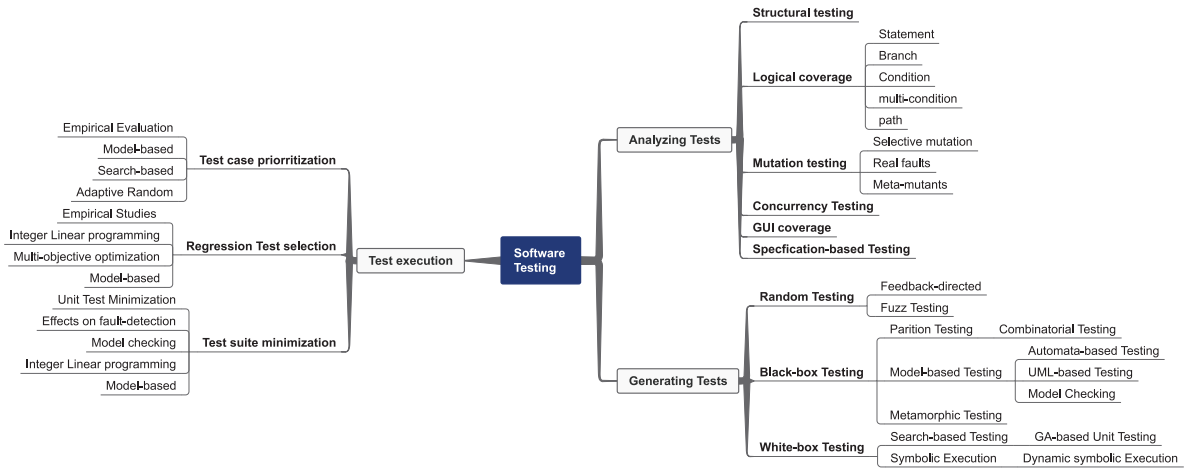


**Fig. 1.** Software testing operations, adapted and modified from [8].

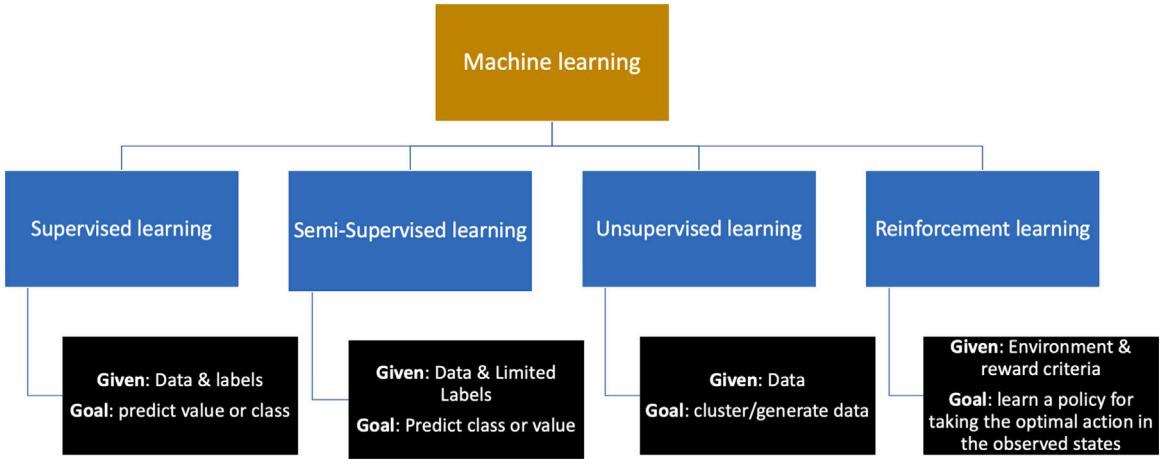


**Fig. 2.** Machine learning types.

enabling advanced functionality for DL. The paper presented by [12] shows a comprehensive overview of DL advances. Nowadays, the DL field includes many models and structures. These algorithms can be categorized into two main categories.

Firstly, discriminative models are used for optimal class discrimination for better data classification. This class includes Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). Secondly, Generative Models are the neural networks that focus on learning how the data was created to generate similar data. Below is a summary of the different algorithms used in each category above.

- **Convolutional Neural Network (CNN)** : CNN is mostly utilized for handling images and performing operations such as feature extraction and recognition. Numerous CNN versions exist, including visual geometry group (VGG) [13], AlexNet [14], Xception [15], Inception, and ResNet [16]. All of these variants were used in different applications and reusability was achieved by using transfer learning. In our context, CNN was complementary to RL to present the system changes in many articles.
- **Recurrent Neural Network (RNN)**: The RNN [17] algorithm is characterized by its capabilities to predict the target based on current and historical data. Since RNN suffered from vanishing gradients problem, which impacted its ability with long sequences of data, few other candidates were introduced, such as Long Short-Term Memory (LSTM) [18], Bi-directional LSTM (BI-LSTM), and Gated Recurrent Units (GRUs) [19]. These methods are usually strengthened by using an attention-based mechanism.
- **Generative models**: These models are usually used to generate data samples. It automatically detects and learns patterns or regularities in input data for the model to create new data samples from the original dataset. Generative Adversarial Networks (GAN) [20] are constructed from two neural networks: a generator (G) that generates new data with comparable features to the original data and a discriminator (D) that forecasts the likelihood that the following sample will be pulled from genuine data rather than the generated data. Thus, both the generator and discriminator are trained to work against one another in GAN modeling. While the generator attempts to deceive and confound the discriminator by providing more data that are genuine, the discriminator attempts to separate original data from G's created data. The family of generative models also includes Auto-encoders (AE) [21], Restricted Boltzmann Machines (RBM) [22], Self-Organizing Maps(SOM) [23], and Deep Belief Networks (DBN) [24].

Supervised and unsupervised learning techniques were successfully used in different tasks of the software testing research field. However, the problem with these two types of learning is that both SL and UL do not assume the dependence of events and thus, are not sufficient for interactive testing. For example, finding the optimal combination of actions to test an application's GUI requires interactivity that cannot be provided by SL and UL methods. Fortunately, Reinforcement Learning (RL) can solve such problems by learning in an interactive environment. In addition, RL promises adaptiveness and other features discussed in the next section.

### 2.3. Reinforcement learning

RL [25] is one of the machine learning approaches for an agent to gain knowledge. Unlike other learning techniques, RL is used to find the optimal decision in an interactive environment and does not require a dataset to learn from as it learns directly from the interaction between the agent and the environment. A basic illustration of the RL process is presented in Fig. 3. Conceptually, RL defines an actor (i.e., agent) and a simulated world/environment, in which the agent will interact. The agent runs within a simulated world. The way the world reacts to the agent's particular behavior is specified by a model that is not necessarily known to the agent. The agent may remain in one of the numerous states $s \in \mathcal{S}$ of the environment or may pick one of the multiple actions $a_t \in \mathcal{A}$ to transition between states. The transition probabilities between states determine the agent's final state (P). After applying an action, the environment provides feedback in the form of a reward $r \in \mathcal{R}$.

RL is usually applied in sequential choice tasks, where the selected action affects future states such as games, software testing, and others. RL relies on the Markov Decision Process (MDP) [26] and has other concepts such as policy and value functions discussed in this section. This paper presents a comprehensive systematic review of all recent software testing papers that includes RL as its primary focus. The fundamental components of an RL system in an MDP problem are ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $\gamma$), where $\mathcal{S}$ represents the list of possible states, $\mathcal{A}$ denotes the list of possible actions at state $S_t$, and $\mathcal{R}$ denotes discounted reward using a discounted factor $\gamma \in (0, 1]$. In the RL process, the next state $S_{t+1}$ is determined for each episode based on the following transition probability

$$P(s', r|s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a]$$

and the reward. In RL, the training process starts with the agent performing random actions $a_t \in \mathcal{A}$ at different state $s_t \in \mathcal{S}$ and receiving rewards $r_{t+1}$. During this training phase and by using the feedback reward, the agent builds a policy $\pi_t(a|s)$ that associates a specific action $\mathcal{A}_t$ with a specific state $\mathcal{S}_t$ that guarantees the highest accumulated rewards given by the following equation.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

This training phase keeps running until the agent exceeds specific timestamp threshold or the number of seconds per episode [27]. The agent is then trained for multiple episodes until converging to the best policy in terms of accumulative rewards.

To decide what action to perform at a given instant, the agent must understand how beneficial it is to be in a particular condition. The *value function* is a technique for determining this operation precisely for each state. It is defined as the total of anticipated rewards following the policy $\pi$ from the current state onward and is given by the symbol $V_\pi(s)$ and equation:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

A similar equation can also be used to provide the goodness of a specific action at a specific state $Q_\pi(s, a)$ given by the following equation.

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

During the training phase, the RL agent collects the interaction trajectories $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$ and learns from them. However, since the RL agent is built upon MDP, it relies only on the last interaction to get the next reward, which reduces the complexity of the problem.

Different RL algorithms may be used based on the type of the model. If the model of the system is known to the agent, Dynamic Programming (DP) can be adopted for finding the best solution to the environment, and the algorithms used for RL in such an environment are called model-based RL algorithms, examples include [28], and MBVE [29]. On the other hand, if the system model is unknown, the agent must use model-free algorithms that explicitly learn the statistical model of the environment. Model-free RL includes Monte Carlo (MC) and temporal difference (TD) as the most common approaches [27]. These two approaches have been adopted in multiple different learning agents, which can be grouped into three different groups as follows:

- **Value-based**: These methods use TD learning that updates the value function on each step according to a temporal error calculated between old and new value functions. Algorithms that fall into this category are SARSA, Q-learning, and Deep Q-network (DQN). These methods are characterized by lower variance, but more bias [30].
- **Policy-based**: Unlike value-based methods, these methods use MC learning that updates a policy at the end of the episode. Changes are done directly to the policy without demanding a particular metric. Examples include Proximal Policy Optimization (PPO) [31].
- **Actor–critic**: These methods harness the power of both value-based and policy-based learning and use MC and TD learning approaches to lower the bias and variance from the above-mentioned methods. Examples include Deep Deterministic Policy Gradient (DDPG) [32].

## 3. Research methodology

In this section, we explain the research questions, the design of our search process, the pipeline followed for study collection, the selection process, the inclusion and exclusion criteria, and the quality assessment. The pipeline followed is shown in Fig. 4. Later in this section, we discuss the final selected papers after applying the designed pipeline.

### 3.1. Primary study selection

For this systematic review, we formulated five research questions to extract insights into how different aspects of reinforcement learning could be applied to various parts of software testing. The following are the formulated research questions:

- **RQ-1**: What are the applications of reinforcement learning in software testing?
- **RQ-2**: What are the commonly used RL algorithms and architectures for learning?
- **RQ-3**: What are the challenges faced using RL within the software testing context?
- **RQ-4**: What are the pros and cons of using RL in software testing?
- **RQ-5**: What is the performance of RL compared to other techniques?

Several electronic databases were chosen to apply our initial study selection. The databases included ACM Digital Library, IEEE Xplore, Web Of Science, Scopus, DBLP, Science Direct, and Springer. These databases were selected based on their reputation, wide acceptance within the Computer Science community, and relevance to our research topic. These databases were chosen for their extensive collection of articles, book chapters, and conference papers in this research field. The selection of these databases aligns with practices in the SLR field and enables us to capture the critical existing publications.

We prepared queries based on a combination of keywords for querying the databases to limit the search to studies on reinforcement learning applied in software testing. The list of the queries used for each database is showcased in Fig. 5. We note that in some databases, the queries were different due to the different retrieving of results from each database, the custom queries per database were made based on the feedback from each database. In this study, defining search

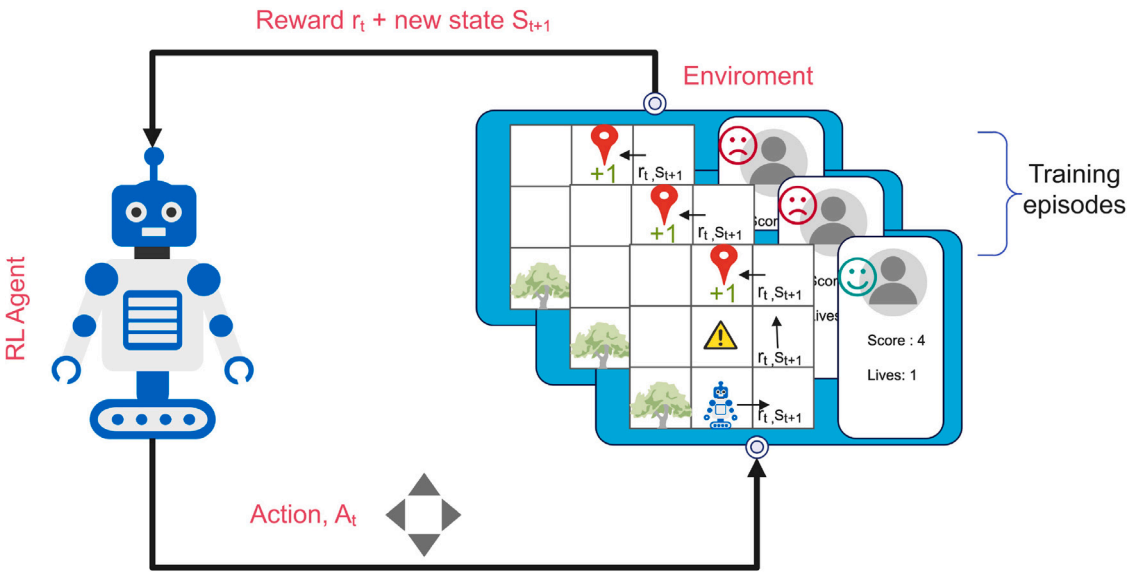


**Fig. 3.** Reinforcement learning cycle.


**Fig. 4.** Search strategy.

**IEEE Xplore**

- ("Reinforcement learning" OR "RL" OR "Multi agent" OR "Sarsa" OR "Q-learning" OR "Q-Learning" OR "QLearning" ) AND ("software testing" OR "software debugging" OR "app testing" OR "program testing" OR "API testing" OR "UI testing") AND ( ("static analysis" OR "dynamic analysis") OR ("Unit testing" OR "mutation" OR "system testing" OR "security testing" OR "Acceptance testing") )

**Scopus**

- ("Reinforcement learning" OR "RL" OR "Multi agent" OR "Sarsa" OR "Q-learning" OR "Q-Learning" OR "QLearning" ) AND ("software testing" OR "software debugging" OR "app testing" OR "program testing" OR "API testing" OR "UI testing") AND ( ("static analysis" OR "dynamic analysis") OR ("Unit testing" OR "mutation" OR "system testing" OR "security testing" OR "Acceptance testing") )

**Science Direct**

- ("reinforcement learning" ) and ("software Testing" or "Testing"))

**Web of Science**

- ("reinforcement learning" and "software Testing")

**Fig. 5.** Different queries used in the survey.

keywords was an iterative process that required careful consideration to ensure that our search was accurate. We began by brainstorming keywords such as technical terms, synonyms, acronyms, and alternative spelling, which are relevant to our research questions. We also used controlled vocabulary terms to ensure that our search is comprehensive. We used Boolean operators (AND, OR, NOT) to combine search terms and construct search strings. Truncation and wildcards were used to capture variations in spelling. We also consulted with subject matter experts to identify additional relevant keywords. To refine our search keywords, we added and removed keywords by adjusting search strings and trying different combinations of search terms.

Querying the different databases on the key relevant terms yielded a plethora of studies. To further select the most relevant papers and eliminate irrelevant or erroneously extracted studies, we defined a set of inclusion and exclusion criteria. It is important to develop clear eligibility criteria and to apply them consistently to all selected studies. This way we ensure that the included studies are relevant, reliable, and valid. Eligibility criteria include a set of criteria, which are used to decide which studies should be included and excluded. Eligibility criteria are mostly based on the research questions. The following types of eligibility criteria were used in this study. The first one is the study design, which is an important eligibility criterion because different study designs have different levels of bias. The second one is the population, which is a key eligibility criterion because the aim of the systematic literature review is specific to a particular population. The third one is the outcome, which is another key eligibility criterion because the research questions are specific to a particular outcome. The fourth one is the language because reviewers can only evaluate the languages they can understand. Last, but not least, is the publication date because some reviews cannot include all the studies published and

need to focus on a certain time frame. The following eligibility criteria were defined in this study:

1. Population: Studies involving reinforcement learning techniques in software testing.
2. Intervention: Studies that describe the integration of reinforcement learning with software testing approaches. Studies that investigate the effectiveness of reinforcement learning in improving software testing processes.
3. Comparator: Studies that compare the performance of reinforcement learning-based software testing approaches with traditional methods.
4. Outcomes: Studies that report empirical results. Studies that discuss challenges and limitations associated with the use of reinforcement learning in software testing.
5. Study Design: Empirical studies, including controlled experiments and case studies. Comparative studies evaluating the effectiveness of reinforcement learning-based software testing approaches.

The inclusion criteria defined in this SLR study are as follows:

1. The article has been published within the last 10 years.
2. The study involves "Reinforcement Learning" or the "Markov Decision Process" and applies it in testing any software.

The exclusion criteria are defined as follows:

1. The article is not related to our study.
2. The article is written in any language other than English.
3. Publication that is a duplicate or already retrieved from another database.
4. The article is inaccessible or the full text of the publication is not available.
5. The article is not a primary study paper.

We obtained a large number of papers upon applying the queries on the different data sources cumulatively. Before performing manual screening, we prepared a data pre-processing script based on some of the inclusion and exclusion criteria that could be automated. Criteria that could be automated included filtering by date, checking if a publication was a survey or a systematic review paper, and removing duplicates within individual databases and across the different databases. After filtering the studies with an automated process that we implemented, in the next section, we discuss the manual process applied to eliminate or include papers based on the remaining inclusion and exclusion criteria that were not automated in this step.

To reach a final set of selected papers, we manually went through each document after the initial automated screening. We first performed a high-level screening of the paper titles and the abstracts to categorize the papers to keep them for further screening, revisit, or eliminate them. Later, we went through each paper and collected source documents for further review. We had to eradicate a few papers in this step as they were not accessible. Next, we proceeded to go through each individual paper while simultaneously performing a quality assessment to arrive at the final set of papers. It is important to develop a transparent process for evaluating the quality of studies. This helps to ensure that the selected publications are of high quality. Evaluating the quality of studies is an important step in selecting publications for an SLR study. Key factors were considered when evaluating the quality of studies. Particularly, in this study, only high-quality databases were used for the identification of papers. Also, a quality assessment step was performed to make sure that the selected articles were high-quality. To streamline the process of title and abstract screening, we also developed a basic web application to visualize titles and abstracts and provide interactivity to select or eliminate papers based on them.

Selected questions shown in Table 1 were used effectively by many researchers who published SLR papers in the literature. The authors of this article also used them before and published several SLR articles. These questions are inspired by the questions proposed by well-established researchers in the SLR field. To ensure the selection of only high-quality papers, we applied a set of quality assessment questions (Table 1) and then started to score each paper according to the following.

- 2: if the paper satisfies the quality assessment question.
- 1: if the paper somewhat satisfies the quality assessment question.
- 0: if the paper does not satisfy the quality assessment question.

Whenever a conflict regarding the assigned scores occurred between authors, a separate discussion session was organized to reach a consensus. As we had a total of 6 quality assessment questions, with a maximum score of 2 points for each question, we established the average threshold level at 6. Consequently, we selected only the papers that scored above 6, indicating their better ability to address the quality assessment questions. The papers retrieval along with the quality assessment have resulted in 40 different papers being a part of our primary studies. The results of the quality assessment are presented in Fig. 6. While the $x$-axis of Fig. 6 displays the quality score intervals of the papers, the $y$-axis represents the count of papers that achieved the corresponding quality score. This figure demonstrates that a significant number of papers were eliminated due to their low-quality content.

In addition to the quality scoring approach, we implemented additional measures to ensure the quality of the selected papers. Firstly, for data extraction reliability, we engaged multiple reviewers who independently conducted data extraction and resolved any discrepancies through consensus. Secondly, a reliability analysis was performed to evaluate the consistency of ratings among the reviewers. Thirdly, the first two authors underwent training provided by the third author, who has previously published several SLR articles. Lastly, we took steps to mitigate various biases in the selection process, including journal bias, citation bias, positive outcome reporting bias, and geographic bias. We conducted a systematic and comprehensive search using multiple databases to mitigate the aforementioned biases.

### 3.2. Data extraction

After selecting the papers for further review, we prepared a data extraction form to extract key information from each selected paper. Data extraction requires careful planning and adherence to specified procedures. For data extraction, we followed the following steps: A standardized data extraction form was developed to collect all relevant data. Variables to be extracted were defined based on the research questions. The data extraction form was pilot-tested to make sure that it captures all relevant data. Data were extracted from all studies. The accuracy of the extracted data was verified. The designed data extraction form is presented in Table 2. Not all the information captured in the data extraction form is present for every selected paper. Therefore, when it comes to papers where certain elements are not covered, we omit only those elements instead of eliminating the paper. The assumption made here is that the subsets of papers are sufficient to answer the unique aspects separately. Furthermore, most papers did not mention challenges, so we had to infer what challenges they faced. In the next section, we discuss the data synthesis and reporting process used in the study.

### 3.3. Data synthesis and reporting

This section discusses the data synthesis applied to the papers selected for aggregation of information and reports on it after summarizing. Like the data extraction step, data synthesis also requires careful planning and attention to detail. For data synthesis, we followed the following steps: First, we identified the statistical methods to be

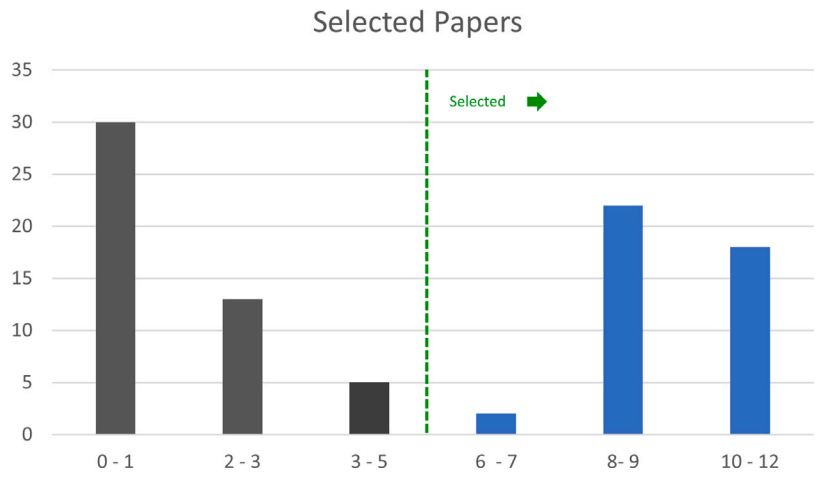


**Fig. 6.** Results of the quality assessment.

**Table 1**
The list of questions used for quality assessment.

| # | Quality assessment questions (scored 2-Yes, 1-Unclear/conflict, 0-No) |
|---|---|
| 1 | Is the scope of software testing clearly defined? |
| 2 | Does the article clearly state the objective? |
| 3 | Does the article compare its approach to industry standards? |
| 4 | Does the article report on the validity of threats? |
| 5 | Does the article formulate the problem in terms of RL? |
| 6 | Is the role of reinforcement learning key in contributing to their proposed approach? |

**Table 2**
Data extraction form.

| Captured elements | Components to capture |
|---|---|
| General Information | DOI, the title of the paper, author names, country, publication year and type |
| RL Algorithm | DQN, Q-learning, ... |
| RL Agent Type | Single-Agent, Multi-Agent |
| RL Environment/states | SUT, Control-Flow Graph, GUI |
| RL Reward criteria | Maximizing coverage, inputs, system faults, ... |
| Software Testing Type | UI Testing, Performance Testing, Security Testing, ... |
| RL Performance | How it compares to traditional approaches |

**Table 3**
The number of papers retrieved and filtered.

| Database | # initial number of papers retrieved | # of papers after exclusion criteria | After combining and filtering | After quality assessment |
|---|---|---|---|---|
| Science Direct | 77 | 16 | 90 | 40 |
| Scopus | 118 | 114 | | |
| ACM | 718 | 96 | | |
| Springer Link | 126 | 1 | | |
| DBLP | 2 | 2 | | |
| Web of Science | 77 | 16 | | |
| IEEE Xplore | 526 | 242 | | |
| Total | 1644 | 487 | | |

used. Second, data were pooled using statistical methods such as meta-analysis. Third, the results were interpreted in the context of the research questions. Last but not least, the quality of evidence was evaluated.

A combination or singled-out element from the data extraction form discussed in the previous section can be used to answer all of the five research questions. We aimed to answer RQ-1, RQ-3, and RQ-4 qualitatively while discussing RQ-2 and RQ-5 quantitatively and qualitatively in the next section.

## 4. Results

In this section, we present the general observations on the topic and individual research questions. The number of documents selected is shown in Table 3. We can observe from this table that we obtained a final number of 40 papers from a total of 1644 papers retrieved from electronic databases. By screening based on the exclusion and inclusion criteria and examining the titles, abstracts, and documents, we reached 90 studies. With the help of the final step called quality assessment, we obtained the final number of papers. Furthermore, the distribution of the selected papers over the last eight years is shown in Fig. 7. From the distribution of papers over the years, we can observe that the trend of applying reinforcement learning for software testing has spiked in the last 2 years. When analyzing the papers over the countries of the authors, we can observe that most of the contributions are from authors in China and the United States, as shown in Fig. 8. A summarized version of the data extracted from the papers is shown in Table 4. Going through each of the primary studies selected one by one, we prepared a mind map as shown in Fig. 9 based on the data extraction

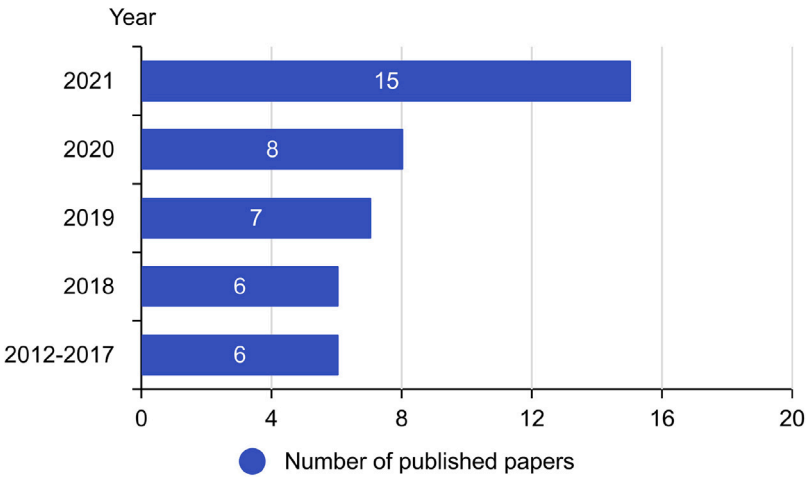


**Fig. 7.** The papers' distributions by years.

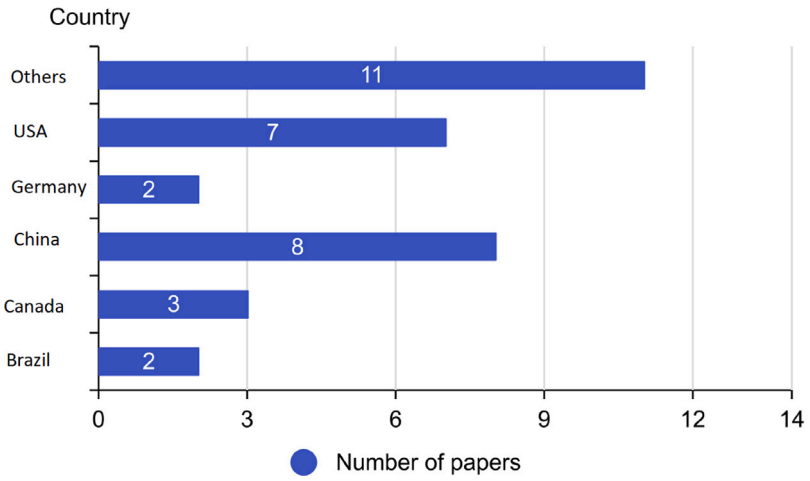


**Fig. 8.** The papers' distributions by countries.

form presented in Table 2. In the following sub-sections, we go through our observations concerning each of the research questions.

### *4.1. RQ-1 what are the applications of reinforcement learning in software testing?*

In the nodes listed under the application branch in the mind map (presented earlier in Fig. 9), we can observe different domains of software testing applications that were undertaken by the studies through applying reinforcement learning techniques either completely as their approach or as an integral part of their proposed approaches. However, after investigating the applications deeply, we see that the literature focused mostly on GUI-related applications such as mobile applications and web GUI testing [33–38]. In addition, test data generation and test case selection (e.g., test case prioritization and reduction) came in second and third place, respectively. Moreover, the validation for robotics-related software (e.g., autonomous vehicles, robots) was in fourth place. Finally, other unique usages presented in only one paper were combined in the others category [39–41]. The distribution of applications is illustrated in Fig. 10.

### *4.2. RQ-2: What are the commonly used RL algorithms and architecture for learning?*

Fig. 9 shows the mind map prepared based on the data extracted from the selected papers. The mind map looks at the core components of RL such as the types of environment, rewards, states, algorithms and applications [42]. The second and third level in the mind map showcases the unique approaches and applications found in the data extracted. The common algorithms used as observed in Fig. 9 fall under three main categories, which are value-based algorithms, actor–critic algorithms, and others. Value-based approaches are algorithms that make choices of actions that are optimal in the states [43]. Actor–critic algorithms utilize an actor-network and a critic network [44]. The actor's role is to decide on choosing actions to take, and the role of the critic network is to criticize and propagate the impact of the action and the necessary adjustments. The distribution of the different RL algorithms used by the studies is shown in Fig. 11. Our analysis showed that most of the studies utilized the relatively simple RL algorithm such as the Q-learning algorithm which is a value-based algorithm [33,35–40,45–57]. Other papers, however, have utilized the power of Deep Learning in empowering reinforcement learning which appears in the use of different advanced algorithms such as Deep Q-Network (DQN) [30], Deep Deterministic Policy Gradient (DDPG) [32], Advantage Actor–Critic (A2C) [58], Asynchronous Advantage Actor–Critic [58] (A3C), Soft Actor–Critic (SAC) [59] and others [34,60–68]. In some papers there was no indication about which RL algorithm was used, thus we use the keyword **OMITTED** to indicate this group of papers.

Furthermore, another aspect of the used RL algorithms is the agent type. The concept of reinforcement learning revolves around a set of the environment(s) and agent(s) capable of performing a set of actions.

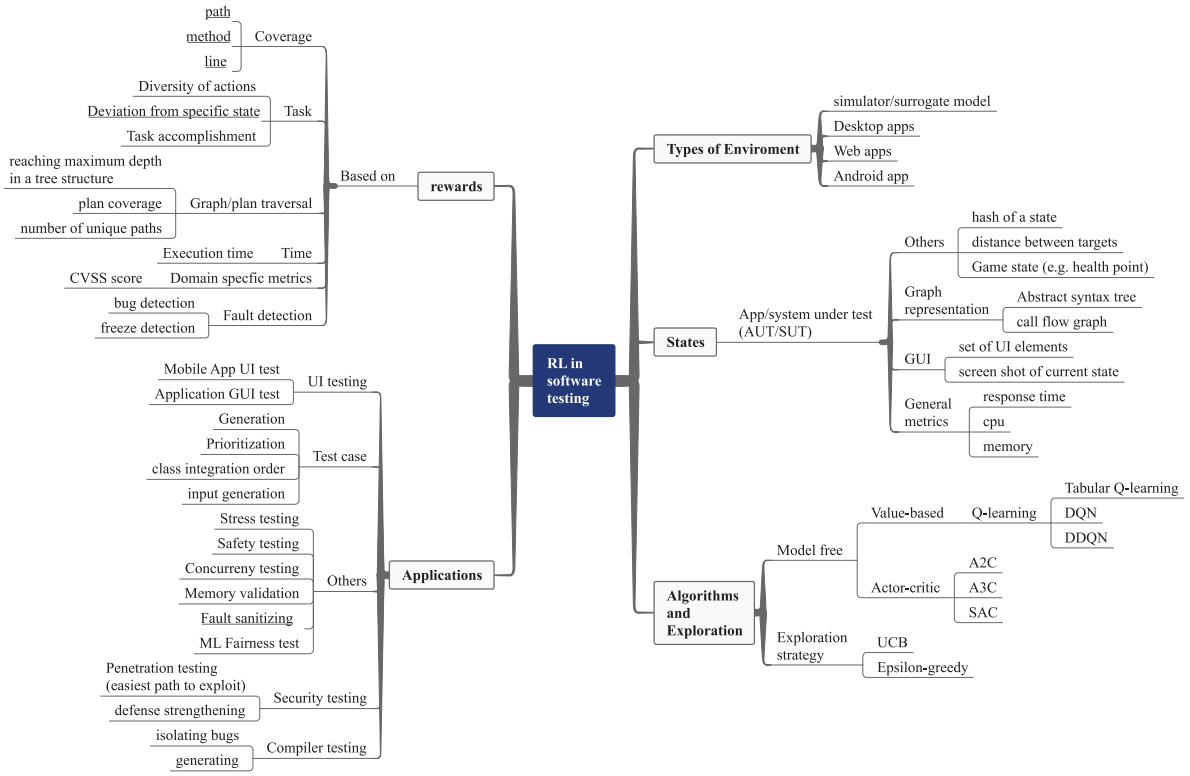


**Fig. 9.** RL in software testing.

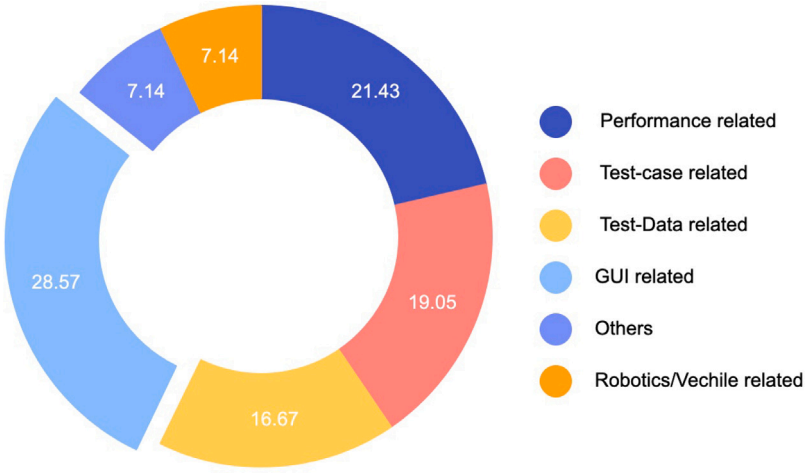


**Fig. 10.** The distribution percentage of RL applications in software testing.

RL algorithms generally are single-agent based where only one agent is present in the environment at a given instance or multi-agent based, where multiple agents are present in the environment with possibilities of agents influencing each other [69]. We observed that the majority of the studies implemented single-agent-based RL algorithms.

### *4.3. RQ-3: What are the challenges in using RL with software engineering and testing?*

One of the key aspects is to design an accurate abstraction of the systems [40]. In many cases, since reinforcement learning essentially needs to learn, it needs sufficient information to train. Having too little information, especially in black-box testing, can hinder the ability to apply reinforcement learning effectively. On the contrary, having too much information can imply a larger environment setup and states, leading to an exponential increase in the time required for [70]. Another challenge discussed is the inability to apply a trained RL model from one application to another without retraining agents. This implies that the time-consuming step of training has to be repeated when the need for re-using the same model on another application arises. Finally, the agent's performance in Reinforcement Learning models can be heavily impacted positively or negatively based on the configured hyper-parameters [71].

### *4.4. RQ-4: What are the pros and cons of using RL in software engineering?*

The primary studies claim several advantages to applying reinforcement learning on different software testing tasks, which can be summarized in the following points:

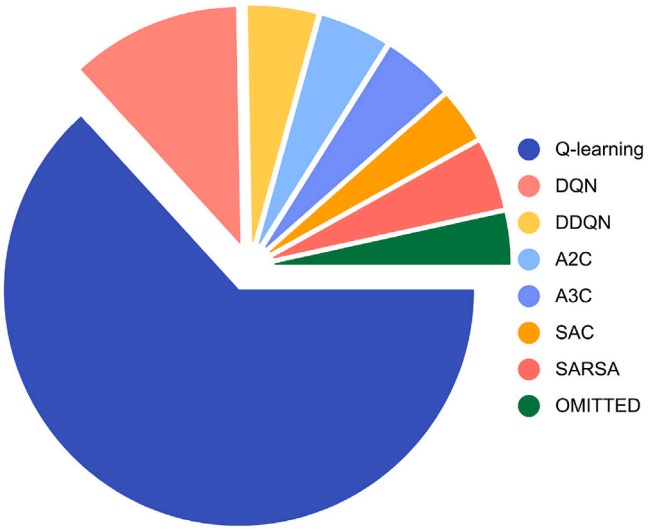


**Fig. 11.** The distribution of RL algorithm used.

- The ability to automate the exploration process for finding the best configurations (e.g., best test inputs or test cases with the highest coverage) without the need for exhaustive search [60].
- The RL agents minimize developers' effort in creating complex test cases [35].
- The adaptive nature of reinforcement learning helps with automating regression testing [72].
- Reinforcement learning can learn and recall adequate behavior for test cases and test input generations [60].
- The trained RL models are scalable and can scale as the application or SUT grows in size and complexity [61].

On the other hand, the cons can be summarized as follows:

- The complexity of designing the correct environment
- RL agents may require a long time or many iterations to train, especially in complex environments, making it unsuitable for software teams that rely on fast testing results.
- For some RL algorithms, hyperparameter tuning might be required to reach the best model with the best actions.

### *4.5. RQ-5: What is the performance of RL compared to other techniques and implementation?*

Fig. 12 showcases the different advantages of using reinforcement learning for software testing compared to other traditional techniques in terms of outcomes, implementation, or approaches. The most noticeable advantage reported by approximately 37% of the studies is that their proposed RL-based approaches yielded better coverage in their respective domains such as branch coverage and statement coverage [36–38,40,46,51,56,73,74]. Other noticeable advantages reported by more than one study include the reduced amount of time required to develop the RL model compared to manually developing test cases, obtaining higher accuracy and efficiency in terms of execution [49,50,56,64,75].

## 5. Discussion

### *5.1. General discussion*

Starting with the applications of reinforcement learning, the majority of the papers used RL in two main tasks. The first one is GUI testing for desktop, web, and mobile applications, while the second one is related to test data generation. The other uses of RL were split among different areas such as stress testing, concurrency testing, compiler testing, memory testing, and others. The use of RL for GUI applications is quite natural since the nature of GUI testing matches exactly the cycle of reinforcement learning presented earlier, and many tools for capturing the state of the GUI such as CNN algorithms exist. Many representations of the state space for app GUI testing were presented such as an image of the current state, the number of components that are present on the screen, and the set of visible components. For data generation, RL was used to create some input cases that increase the code coverage or cause an exception to the software. This explains the reward function focused on the code coverage and the number of exceptions for many papers. The rest of the papers introduced different techniques for using RL, however, many of these papers do not compare their RL problem formulations against other formulations, which can be considered as a drawback.

As a general observation among all papers, the most commonly used algorithm was Q-learning, while other papers selected different algorithms according to their specific needs. Considering that RL agents have been advancing over the years and new algorithms have been introduced, the use of Q-learning could have potentially impacted this research area adversely. This is because the Q-learning algorithm is considered simple and only sufficient for systems with a small number of states. Other algorithms, especially actor-critic-based algorithms, might provide much better results with faster convergence. In terms of RL agent type, almost all reviewed papers used single-agent reinforcement learning to achieve their own goal. While a single agent was sufficient in many papers, the use and impact of multi-agent RL on software testing remain to be investigated. While RL showed optimal or semi-optimal results in almost all reviewed papers, some challenges have been raised by many researchers. Most of these challenges come from the nature of RL. Examples include the difficulty of choosing an adequate state representative, in addition to the cost of training an agent related to the size of the system, the reusability of the model, and the sensitivity of many RL algorithms to hyperparameters and seeds used.

Reinforcement learning can be applied to many testing types. Each type of software testing has its challenges in designing, developing, and executing. Exhaustive testing is about testing a system completely by implementing an exhaustive set of tests covering an entire system. Heuristically prioritizing cases where not testing may have higher costs can help reduce costs in implementing tests. One of the advantages of reinforcement learning is its ability to learn meta-heuristic properties automatically, which allows for prioritizing and generating optimal test cases. Test input generation and test case generation or development require a heuristic understanding of the system. When a system needs to be black-box tested, it becomes even more challenging, as the system cannot be analyzed to develop tests efficiently. RL's exploratory nature and learning ability can be suitable for efficiently achieving high coverage when it comes to black-box testing. The scalable nature of RL models through their continued learning makes them suitable for regression testing as well. Automated GUI testing can become quite challenging when there are continuous changes. Innovative gestures and interactivity in modern GUIs only add to this challenge. Once test cases have been developed, maintaining complex GUI test suites becomes difficult.

Regression testing is another case where tests that have been developed need to continuously evolve, similar to maintaining complex GUI tests, this is also challenging. Systems with multiple users have complex states resulting from concurrent actions of the users where the system might fail. Developing tests that take into consideration all possibilities of a system that handles concurrent actions, especially large systems requires consideration of costs and return on investment. The advantage of RL in reducing the time required for development, and its ability to scale and automatically grow with the growth of the target systems solves the major challenge of maintainability of test cases. With automated generation capabilities of both data and test

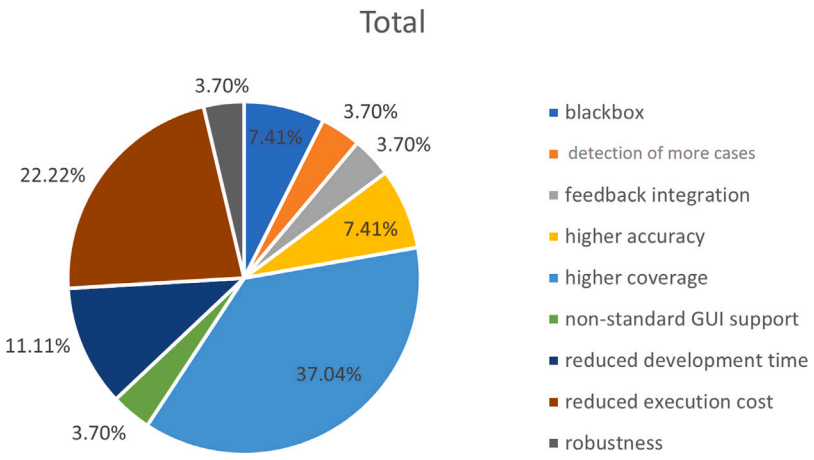


**Fig. 12.** Advantages of reinforcement learning over traditional approaches.

cases, well-designed RL models can prove to be an efficient toolset for software professionals for test development and setting up continuous integration pipelines.

Few studies used RL for some other non-traditional testing such as testing autonomous vehicles for safety, regression testing in games, and fairness of machine learning models. We believe that the research direction with an influx of studies on software testing using RL in the last few years is in the right direction. Since the majority of the studies focused on GUI testing or performance testing, more research needs to be done to exploit the full potential of RL in other testing domains which only a few studies covered such as autonomous vehicle testing, robotic testing, exploratory testing, and other tests.

Fig. 9 proposed a wide map of possible RL applications in the software testing domain using the selected primary studies. However, there are also some other aspects that are particularly important. Test case adaptation, test case scheduling, regression testing, and continuous integration and deployment are some of these aspects, which are briefly explained as follows:

- Test case adaptation: RL can be used to adapt test cases over time based on changes in the software and the environment. This can be done by training an RL algorithm to adjust test cases based on the expected impact on software quality.
- Test case scheduling: RL can be used to schedule test cases to maximize the expected impact on software quality. For example, an RL algorithm could be trained to schedule test cases in a way that minimizes the expected time to uncover defects.
- Regression testing: RL can be used to optimize regression testing, which is the process of testing software changes to ensure that they do not break existing functionality. RL can be used to select test cases that are most likely to uncover regression defects or to prioritize test cases based on their expected impact on software quality.
- Continuous integration and deployment: RL can be used to optimize continuous integration and deployment, which are software development processes that automate the integration and deployment of software changes. RL can be used to select the most appropriate testing techniques for each change and to schedule tests to optimize the expected impact on software quality.

RL has applications not only in white box testing but also in black box testing. RL can be applied to white box testing in various ways, such as optimizing the selection of test cases based on the expected impact on software quality, prioritizing test cases based on the expected impact on software quality, or generating new test cases. RL can also be applied to black-box testing in different ways, such as to optimize the selection of inputs and expected outputs based on the expected impact on software quality or to prioritize inputs and expected outputs based on the expected impact on software quality.

RL approaches are better suited to tasks that involve decision-making under uncertainty. A specific RL approach takes into account the complexity of the task, the availability of required knowledge, resources that are available for training, and the size of the state/action spaces. If there is a mathematical model describing the dynamics of the environment, model-based RL approaches can be better suited. In the case of larger state/action spaces where the environment modeling is difficult, model-free RL approaches suit better. If traditional RL algorithms cannot scale for tasks having high-dimensional state spaces, deep RL approaches are more suitable. When multiple agents interact in a dynamic environment, the multi-agent RL approach is better. As such, we can state that the choice of the RL approach is related to the constraints and requirements of the task.

Fig. 9 presents commonly used RL algorithms and architecture for learning. We observed that most studies used value-based approaches, particularly the Q-learning algorithm. A deep learning version of this algorithm, called Deep Q-Network (DQN), was also utilized in some studies. There are several improvements that have been built on the original DQN algorithm. Some of these networks are Dueling DQN, Double DQN, Prioritized Experience Replay, and Rainbow.

Fig. 9 shows that the majority of the studies implemented single-agent-based RL algorithms instead of multi-agent-based RL algorithms. There are several multi-agent reinforcement learning algorithms proposed in the literature. Independent Q-learning, multi-agent deep reinforcement learning, actor–critic methods, and joint action learning are some examples of multi-agent RL algorithms.

Fig. 12 presents the advantages of RL over traditional approaches. Concerning the performance comparison (RQ5), when comparing the performance of reinforcement learning (RL) with other approaches, there are several important factors that need to be considered.

- Study design: Factors such as the size of the sample, the length of the study, and the methods used to assess the performance are critical.
- Methods for comparing performance: A direct comparison made between RL and other approaches is different than comparing the performance of the RL model with a baseline or a control group.
- Types of applications: Application domains such as gaming and robotics can affect the analysis results.
- Results of the comparison: The magnitude of the effect size and the statistical significance of the results are also important.
- Implications for practice: The implications of the performance comparison for practice need to be evaluated.

### 5.2. Threats to validity

There are several limitations of this SLR study. Although study selection criteria and quality assessment procedures were applied, the quality of studies may have introduced bias into the results. While the authors aimed to make the interpretation of the findings objective, the assumptions and biases of the authors may have influenced this objectivity. Errors in data extraction or synthesis might have introduced bias or inaccuracies in the results. Although many databases were included, some papers might have been missed due to many other factors. Search keywords were iteratively identified, however, there might be some keywords that we have not considered in this study.

Internal validity indicates that the study design was able to appropriately extract relevant information to connect the studies to the research questions. We designed each of the research questions and data extraction forms and performed a further general assessment to appropriately inquire about the elements involved in applying reinforcement learning to software testing. Strict eligibility criteria were used to determine which studies should be included (i.e., study selection criteria), and the quality of the studies was assessed systematically and in a reliable way (i.e., quality assessment). The data extraction and synthesis methods were also transparent and systematic, therefore, the results are accurate and reliable.

The search strategy was comprehensive and transparent, including multiple sources and using appropriate keywords to capture relevant studies, and the study selection criteria were broad enough to capture a diverse range of studies.

The degree to which the conclusions we draw regarding relationships in our data are logical is known as conclusion validity. The conclusions drawn are based on discussion among the authors to reduce individual bias on interpretations. This also applies to the designing process of identification, screening of papers, and analysis presented in this paper. The results of the review were synthesized systematically and transparently.

## 6. Conclusion and future work

Software testing has become a crucial element in ensuring high-quality software development. For some complex scenarios, neither supervised nor unsupervised machine learning methods were adequate. In this SLR, we explored the use of reinforcement as one of the promising areas to explore for software testing. To the best of our knowledge, this SLR paper is the first one to systematically survey the role of RL in software testing. This study provides a lot of important observations. We first showed that RL has been widely used in software testing but has almost narrowed to two applications. Moreover, the literature showed a shortage of papers using advanced RL techniques in addition to multi-agent RL. Secondly, we showed the challenges of applying reinforcement learning, which comes from the nature of RL. Thirdly, the advantages of using RL were presented and showed optimal or semi-optimal results compared to other baselines. Finally, we raised some points that need further investigation. Future work needs to focus on exploring advanced RL techniques that show faster convergence than the typical Q-learning algorithm. In addition, useful RL techniques such as curriculum learning and imitation learning should be explored as they cut down training time. We also advise new researchers to compare different system states for the same problem. This will help future researchers choose the optimal configurations for training their RL agents.

## Funding declaration

Open access funding provided by the Qatar National Library.

## CRediT authorship contribution statement

**Amr Abo-eleneen:** Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing, Visualization. **Ahammed Palliyali:** Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing, Visualization. **Cagatay Catal:** Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing, Supervision.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## References

[1] P. Jalote, An Integrated Approach to Software Engineering, Springer Science & Business Media, 2012.

[2] D. Kumar, K. Mishra, The impacts of test automation on software's cost, quality and time to market, Procedia Comput. Sci. 79 (2016) 8–15, http://dx.doi.org/10.1016/j.procs.2016.03.003, Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[3] E. Kit, S. Finzi, Software Testing in the Real World: Improving the Process, ACM Press/Addison-Wesley Publishing Co., 1995.

[4] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, John Wiley & Sons, 2011.

[5] N. Oster, F. Saglietti, Automatic test data generation by multi-objective optimisation, in: International Conference on Computer Safety, Reliability, and Security, Springer, 2006, pp. 426–438.

[6] R. Ramler, K. Wolfmaier, Economic perspectives in test automation: balancing automated and manual testing with opportunity cost, in: Proceedings of the 2006 International Workshop on Automation of Software Test, 2006, pp. 85–91.

[7] D. Araiza-Illan, A.G. Pipe, K. Eder, Intelligent agent-based stimulation for testing robotic software in human-robot interactions, in: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, MORSE '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 9–16, http://dx.doi.org/10.1145/3022099.3022101.

[8] G. Fraser, J.M. Rojas, Software testing, in: Handbook of Software Engineering, Springer, 2019, pp. 123–192.

[9] K.P. Murphy, et al., Naive bayes classifiers, Univ. Br. Columbia 18 (60) (2006) 1–8.

[10] N. Cristianini, E. Ricci, Support vector machines, in: M.-Y. Kao (Ed.), Encyclopedia of Algorithms, Springer US, Boston, MA, 2008, pp. 928–932.

[11] T. Li, S. Zhu, M. Ogihara, Using discriminant analysis for multi-class classification: An experimental investigation, Knowl. Inf. Syst. 10 (4) (2006) 453–472.

[12] I.H. Sarker, Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions, SN Comput. Sci. 2 (6) (2021) 1–20.

[13] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2015, arXiv:1409.1556.

[14] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 25, Curran Associates, Inc., 2012.

[15] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1251–1258.

[16] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2015, arXiv:1512.03385.

[17] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724–1734, http://dx.doi.org/10.3115/v1/D14-1179.

[18] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.

[19] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, Association for Computational Linguistics, 2014, pp. 1724–1734, http://dx.doi.org/10.3115/v1/D14-1179, URL https://aclanthology.org/D14-1179.

[20] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, 2014, arXiv:1406.2661.

[21] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, 2021, arXiv:2003.05991.

[22] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted Boltzmann machines for collaborative filtering, in: Proceedings of the 24th International Conference on Machine Learning, ICML '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 791–798, http://dx.doi.org/10.1145/1273496.1273596.

[23] T. Kohonen, The self-organizing map, Proc. IEEE 78 (9) (1990) 1464–1480, http://dx.doi.org/10.1109/5.58325.

[24] G.E. Hinton, Deep belief networks, Scholarpedia 4 (5) (2009) 5947.

[25] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, Robotica 17 (2) (1999) 229–235.

[26] M.L. Puterman, Markov decision processes, in: Handbooks in Operations Research and Management Science, Vol. 2, Elsevier, 1990, pp. 331–434.

[27] R.S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, ACM Sigart Bull. 2 (4) (1991) 160–163.

[28] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, et al., Imagination-augmented agents for deep reinforcement learning, Adv. Neural Inf. Process. Syst. 30 (2017).

[29] V. Feinberg, A. Wan, I. Stoica, M.I. Jordan, J.E. Gonzalez, S. Levine, Model-based value estimation for efficient model-free reinforcement learning, 2018, arXiv preprint arXiv:1803.00101.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.

[32] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv preprint arXiv:1509.02971.

[33] L. Harries, DRIFT: Deep reinforcement learning for functional software testing, 2020, CoRR abs/2007.08220, URL https://arxiv.org/abs/2007.08220.

[34] E. Collins, A. Neto, A. Vincenzi, J. Maldonado, Deep reinforcement learning based android application GUI testing, in: Brazilian Symposium on Software Engineering, Association for Computing Machinery, New York, NY, USA, 2021, pp. 186–194, http://dx.doi.org/10.1145/3474624.3474634, URL https://dl.acm.org/doi/10.1145/3474624.3474634.

[35] L. Mariani, M. Pezze, O.R.M. Santoro, AutoBlackTest: Automatic black-box testing of interactive applications, in: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, IEEE, pp. 81–90, http://dx.doi.org/10.1109/ICST.2012.88.

[36] Z. Wu, E. Johnson, W. Yang, O. Bastani, D. Song, J. Peng, T. Xie, A reinforcement learning based approach to automated testing of android applications, A-TEST 2018 - Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, Co-located with FSE 2018 (2018) 31–37, http://dx.doi.org/10.1145/3278186.3278191.

[37] D. Adamo, M.K. Khan, S. Koppula, R. Bryce, Reinforcement learning for android GUI testing, in: A-TEST 2018 - Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, Co-located with FSE 2018, Association for Computing Machinery, Inc, 2018, pp. 2–8, http://dx.doi.org/10.1145/3278186.3278187.

[38] S. Carino, J.H. Andrews, Dynamically testing GUIs using ant colony optimization, in: Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Institute of Electrical and Electronics Engineers Inc., 2016, pp. 138–148, http://dx.doi.org/10.1109/ASE.2015.70.

[39] H. Dai, Y. Li, C. Wang, R. Singh, P.-S. Huang, P. Kohli, Learning transferable graph exploration, in: Proceedings of the 33rd International Conference on Neural Information Processing Systems, Curran Associates Inc., Red Hook, NY, USA, 2019.

[40] S. Mukherjee, P. Deligiannis, A. Biswas, A. Lal, Learning-based controlled concurrency testing, Proc. ACM Program. Lang. 4 (2020) http://dx.doi.org/10.1145/3428298, URL https://dl.acm.org/doi/10.1145/3428298.

[41] N. Pfeifer, B.V. Zimpel, G.A. Andrade, L.C. dos Santos, A reinforcement learning approach to directed test generation for shared memory verification, in: 2020 Design, Automation & Test in Europe Conference & Exhibition, DATE, IEEE, pp. 538–543, http://dx.doi.org/10.23919/DATE48585.2020.9116198.

[42] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 2018.

[43] F.S. Melo, Convergence of Q-learning: A simple proof, Tech. Rep, Institute of Systems and Robotics, 2001, pp. 1–4.

[44] I. Grondman, L. Busoniu, G.A. Lopes, R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, IEEE Trans. Syst. Man Cybern. C(Applications and Reviews) 42 (6) (2012) 1291–1307.

[45] G. Czibula, I.G. Czibula, Z. Marian, An effective approach for determining the class integration test order using reinforcement learning, Appl. Soft Comput. 65 (2018) 517–530, http://dx.doi.org/10.1016/J.ASOC.2018.01.042.

[46] Y. Zheng, Y. Liu, X. Xie, Y. Liu, L. Ma, J. Hao, Y. Liu, Automatic web testing using curiosity-driven reinforcement learning, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering, ICSE, IEEE, pp. 423–435, http://dx.doi.org/10.1109/ICSE43902.2021.00048.

[47] M. Buzdalov, A. Buzdalova, I. Petrova, Generation of tests for programming challenge tasks using multi-objective optimization, in: GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference Companion, 2013, pp. 1655–1658, http://dx.doi.org/10.1145/2464576.2482746, URL http://twitthere.wordpress.com.

[48] M.H. Moghadam, Machine learning-assisted performance testing, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1187–1189, http://dx.doi.org/10.1145/3338906.3342484, URL https://dl.acm.org/doi/abs/10.1145/3338906.3342484.

[49] X. Zhang, M. Lin, D. Zhang, A learning strategy for software testing optimization based on dynamic programming, in: Proceedings of the Fourth Asia-Pacific Symposium on Internetware, Association for Computing Machinery, New York, NY, USA, 2012, http://dx.doi.org/10.1145/2430475.2430483, URL https://dl.acm.org/doi/10.1145/2430475.2430483.

[50] M. Esnaashari, A.H. Damia, Automation of software test data generation using genetic algorithm and reinforcement learning, Expert Syst. Appl. 183 (2021) http://dx.doi.org/10.1016/J.ESWA.2021.115446.

[51] Y. Wu, Y. Chen, X. Xie, B. Yu, C. Fan, L. Ma, Regression testing of massively multiplayer online role-playing games, in: 2020 IEEE International Conference on Software Maintenance and Evolution, ICSME, 2020, pp. 692–696, http://dx.doi.org/10.1109/ICSME46990.2020.00074.

[52] J. Wu, C. Zhang, G. Pu, Reinforcement learning guided symbolic execution, in: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, pp. 662–663, http://dx.doi.org/10.1109/SANER48275.2020.9054815.

[53] C.-Y. Chen, J.-L. Huang, Reinforcement-learning-based test program generation for software-based self-test, in: 2019 IEEE 28th Asian Test Symposium, ATS, 2019, pp. 73–735, http://dx.doi.org/10.1109/ATS47505.2019.00013.

[54] K. Viswanadha, S. Shivaji, R. Shanmugam, S. Song, S. Choi, N. Kumar, ATARI: Autonomous testing and real-time intelligence - A framework for autonomously testing modern applications, in: 2019 IEEE International Conference on Artificial Intelligence Testing, AITest, 2019, pp. 52–54, http://dx.doi.org/10.1109/AITest.2019.000-8.

[55] K. Böttinger, P. Godefroid, R. Singh, Deep reinforcement fuzzing, in: 2018 IEEE Security and Privacy Workshops, SPW, IEEE, pp. 116–122, http://dx.doi.org/10.1109/SPW.2018.00026.

[56] D. Araiza-Illan, A.G. Pipe, K. Eder, Intelligent agent-based stimulation for testing robotic software in human-robot interactions, in: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, Association for Computing Machinery, New York, NY, USA, 2016, pp. 9–16, http://dx.doi.org/10.1145/3022099.3022101.

[57] S. Feng, Y. Feng, H. Sun, S. Bao, Y. Zhang, H.X. Liu, Testing scenario library generation for connected and automated vehicles, part II: Case studies, IEEE Trans. Intell. Transp. Syst. 22 (9) (2021) 5635–5647, http://dx.doi.org/10.1109/tits.2020.2988309.

[58] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, 2016, arXiv:1602.01783.

[59] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018, arXiv:1801.01290.

[60] T. Ahmad, A. Ashraf, D. Truscan, A. Domi, I. Porres, Using deep reinforcement learning for exploratory performance testing of software systems with multi-dimensional input spaces, IEEE Access 8 (2020) 195000–195020, http://dx.doi.org/10.1109/ACCESS.2020.3033888.

[61] S. Huurman, X. Bai, T. Hirtz, Generating API test data using deep reinforcement learning, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Association for Computing Machinery, New York, NY, USA, 2020, pp. 541–544, http://dx.doi.org/10.1145/3387940.3392214, URL https://dl.acm.org/doi/abs/10.1145/3387940.3392214.

[62] A. Nguyen, B. Le, V. Nguyen, Prioritizing automated user interface tests using reinforcement learning, PervasiveHealth: Perv. Comput. Technol. Healthc. 7 (2019) 56–65, http://dx.doi.org/10.1145/3345629.3345636.

[63] J. Chen, H. Ma, L. Zhang, Enhanced compiler bug isolation via memoized search, in: 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE, IEEE, pp. 78–89.

[64] C. Degott, N.P. Borges, A. Zeller, Learning user interface element interactions, in: ISSTA 2019 - Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Association for Computing Machinery, Inc, 2019, pp. 101–111, http://dx.doi.org/10.1145/3293882.3330569.

[65] A. Groce, A. Fern, J. Pinto, T. Bauer, A. Alipour, M.E.C. Lopez, Lightweight automated testing with adaptation-based programming, in: 2012 IEEE 23rd International Symposium on Software Reliability Engineering, IEEE, pp. 161–170, http://dx.doi.org/10.1109/ISSRE.2012.1, URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6405364.

[66] J. Kim, M. Kwon, S. Yoo, Generating test input with deep reinforcement learning, in: Proceedings of the 11th International Workshop on Search-Based Software Testing, Association for Computing Machinery, New York, NY, USA, 2018, pp. 51–58, http://dx.doi.org/10.1145/3194718.3194720, URL https://ieeexplore.ieee.org/document/8452812.

[67] C. Paduraru, A. Stefanescu, B. Ghimis, Testing multi-tenant applications using fuzzing and reinforcement learning, in: LANGETI 2020 - Proceedings of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing, Co-located with ESEC/FSE 2020, Association for Computing Machinery, Inc, 2020, pp. 1–6, http://dx.doi.org/10.1145/3416504.3424333.

[68] S. Reddy, C. Lemieux, R. Padhye, K. Sen, Quickly generating diverse valid test inputs with reinforcement learning, in: Proceedings - International Conference on Software Engineering, Vol. 12, IEEE Computer Society, 2020, pp. 1410–1421, http://dx.doi.org/10.1145/3377811.3380399.

[69] G. Neto, From single-agent to multi-agent reinforcement learning: Foundational concepts and methods, Learning Theory Course 2 (2005).

[70] S.D. Whitehead, L.-J. Lin, Reinforcement learning of non-Markov decision processes, Artif. Intell. 73 (1–2) (1995) 271–306.

[71] B. Zhang, R. Rajan, L. Pineda, N. Lambert, A. Biedenkapp, K. Chua, F. Hutter, R. Calandra, On the importance of hyperparameter optimization for model-based reinforcement learning, in: International Conference on Artificial Intelligence and Statistics, PMLR, 2021, pp. 4015–4023.

[72] T. Shi, L. Xiao, K. Wu, Reinforcement learning based test case prioritization for enhancing the security of software, in: 2020 IEEE 7th International Conference on Data Science and Advanced Analytics, DSAA, IEEE, pp. 663–672, http://dx.doi.org/10.1109/DSAA49011.2020.00076.

[73] J. Čegiň, K. Rástočný, Test data generation for MC/DC criterion using reinforcement learning, in: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, pp. 354–357, http://dx.doi.org/10.1109/ICSTW50294.2020.00063.

[74] D. Karunakaran, S. Worrall, E. Nebot, Efficient statistical validation with edge cases to evaluate highly automated vehicles, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC, IEEE, pp. 1–8, http://dx.doi.org/10.1109/ITSC45102.2020.9294590, URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9294590.

[75] C. Paduraru, M. Paduraru, A. Stefanescu, Optimizing decision making in concolic execution using reinforcement learning, in: 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, 2020, pp. 52–61, http://dx.doi.org/10.1109/ICSTW50294.2020.00025.