UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

DeepRLGUIMAT: Deep Reinforcement Learning-based GUI Mobile Application Testing Approach

Eliane Figueiredo Collins Ribeiro

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Eliane Figueiredo Collins Ribeiro

DeepRLGUIMAT: Deep Reinforcement Learning-based GUI Mobile Application Testing Approach

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. José Carlos Maldonado
Co-advisor: Prof. Dr. Arilo Dias-Neto

USP – São Carlos
February 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

F475d

Figueiredo Collins Ribeiro, Eliane
DeepRLGUIMAT: Deep Reinforcement Learning-based GUI Mobile Application Testing Approach / Eliane Figueiredo Collins Ribeiro; orientador José Carlos Maldonado; coorientador Arilo Dias Neto. -- São Carlos, 2022.
122 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2022.

1. TESTE E AVALIAÇÃO DE SOFTWARE. 2. AUTOMAÇÃO DE TESTE DE SOFTWARE MÓVEL. 3. APRENDIZADO DE MÁQUINA POR REFORÇO. I. Maldonado, José Carlos , orient. II. Dias Neto, Arilo, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Eliane Figueiredo Collins Ribeiro

DeepRLGUIMAT: Abordagem de Aprendizado de Máquina Profundo por Esforço Aplicado a Testes de GUI de Aplicações Móveis

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. José Carlos Maldonado
Coorientador: Prof. Dr. Arilo Dias-Neto

USP – São Carlos
Fevereiro de 2022

“Dedico este trabalho à minha mãe Ermelinda,
que não pôde se formar professora mas ensinou aos filhos o
gosto pelos estudos. Às minhas finadas avós, que em sua época
não lhes era permitido ir além de ler e escrever, e às
gerações de mulheres que lutaram para que
hoje este sonho fosse possível.”

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr José Carlos Maldonado, for the opportunity to be his student and the teachings along this journey. I thank my co-supervisor Arilo Dias Neto, who opened the doors for this realization to be possible. Thanks to Professor Auri Vincenzi for his attention and support in my research.

Special thanks to the Institute of Mathematics and Computer Sciences (ICMC) for training, staff and teachers.

I thank my family, mother, and brothers who together and patiently supported this incredible journey of doctoral research. To my husband, José Luiz, who contributed so much and supported me in the most challenging moments with caring and pride. And I thank my friends from LABES, highlighting Jorge Prates, who helped me keep going strong on this journey.

“To me programming is more than an important practical art. It is also a gigantic undertaking in the foundations of knowledge.”

(Grace Hopper)

RESUMO

COLLINS, E. **DeepRLGUIMAT: Abordagem de Aprendizado de Máquina Profundo por Esforço Aplicado a Testes de GUI de Aplicações Móveis**. 2022. 120 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

Contexto: Os constantes avanços nas tecnologias de computação móvel e a demanda do mercado por novos produtos e aplicativos que atendam a um público cada vez mais amplo representam uma oportunidade e a necessidade de refletir sobre como garantir a qualidade desses aplicativos móveis, sendo também o teste de software para esses aplicativos importante e fundamental. A pesquisa nesta área tem sido cada vez mais necessária porque serve como evidência de qualidade do sistema. No entanto, aplicativos móveis têm algumas características e limitações, como a quantidade de memória a ser usada, a vida útil da bateria, a quantidade de dados de entrada, o tamanho da tela do dispositivo móvel e os diferentes sistemas operacionais. As ferramentas de teste disponíveis são ainda limitadas nas estratégias e critérios de de teste que apóiam. Considerando o crescimento do uso desses aplicativos e os desafios que a automação de teste enfrenta, como muitas combinações de operações, transições, cobertura de funcionalidade, mudanças de elementos de interface e reprodução de falhas, pesquisas para contornar essas dificuldades são encorajadas. Com isso, estudos de técnicas de Inteligência Artificial, como Aprendizado de Máquina por Reforço, surgem como uma oportunidade de aprimorar esta área para geração de casos de teste por meio da exploração de aplicativos. Objetivo: Este trabalho propõe a abordagem DeepRLGUIMAT que utiliza a técnica de aprendizado por reforço profundo, com o algoritmo Deep Q-Network para gerar casos de teste por meio da exploração do aplicativo móvel utilizando tentativa e erro, produzindo uma variedade de dados de teste de entrada de acordo com o método de Testes Funcionais Sistemáticos e seguindo a probabilidade distribuição para satisfazer o propósito de cobrir funcionalidades da aplicação. Método: Foi conduzida uma investigação na literatura técnica por meio de um mapeamento sistemático para conhecer os principais estudos na área. Com base nas informações adquiridas a abordagem foi elaborada e desenvolvida por meio de uma ferramenta para prova de conceito para executar a estratégia elaborada em aplicações Android. Por fim foram conduzidos experimentos empíricos em 30 aplicações móveis e feita comparação com abordagens similares na literatura (ferramentas com abordagem randômica Monkey, abordagem Model-based com Aprendizado de Máquina Droidbot, e as abordagens que usam aprendizado de máquina por reforço DroidbotX e Q-testing) em termos de métricas de cobertura de código, falhas encontradas e cobertura de funcionalidades. Resultados: Foram observados que a abordagem proposta atingiu maior valor para cobertura de código em instrução, branches, linhas de código e métodos em comparação com as ferramentas de estado da arte. Em termos de falhar e travamentos encontrados, a ferramenta obteve resultado igual ao das ferramentas de estado da arte. Em cobertura de funcionalidade, a abordagem

proposta exercitou mais operações funcionais em comparação com as ferramentas comparadas. Conclusão:A abordagem proposta mostrou resultados promissores sendo mais efetiva para navegação e em executar operações nas aplicações, gerando testes úteis e efetivos que exercitam a variações de entrada de dados que facilitam a cobertura de funcionalidades de aplicações móveis e com maior possibilidades de indentificar a presenção de erros/fallas.

Palavras-chave: Teste de Aplicações Móveis, Aprendizado de Máquina por Reforço, Automação de testes.

ABSTRACT

COLLINS, E. **DeepRLGUIMAT: Deep Reinforcement Learning-based GUI Mobile Application Testing Approach**. 2022. 120 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

The constant advances in mobile computing technologies and the market demand for new products and applications that serve an increasingly broad audience represent an opportunity and a need to reflect on how to guarantee the quality of these mobile applications, as well as software testing important and fundamental for these applications. Research in this area has been increasingly needed because it serves as evidence of system quality. However, mobile applications have some characteristics and limitations such as the amount of memory to use, battery life, amount of input data, mobile device screen size and different operating systems. The testing tools available are still limited in the testing strategies and criteria they support. Considering the growth in the use of these applications and the challenges that test automation faces, such as many possible combinations of operations, transitions, functionality coverage, interface element changes and fault reproduction, research to overcome these difficulties is encouraged. Thus, studies of Artificial Intelligence techniques, such as Reinforcement Learning, emerge as an opportunity to improve this area for the generation of test cases through the exploration of applications. Objective: This work proposes the DeepRLGUIMAT approach, which uses the technique of deep reinforcement learning, Deep Q-Network algorithm to generate test cases through the exploration of the mobile application using trial and error, producing a variety of input test data according to the Systematic Functional Testing method and following the probability distribution to satisfy the purpose of covering application functionality. Method: An investigation was carried out in the technical literature through a systematic mapping to know the main studies in this area. The approach was designed and developed based on the information acquired, built a tool for concept proof to execute the elaborated strategy. Finally, empirical experiments were carried out in 30 mobile applications. A comparison was made with similar approaches in the literature (tools with Monkey random approach, Model-based approach with Droidbot Machine Learning, and approaches that use DroidbotX reinforcement machine learning and Q-testing) in terms of code coverage metrics, found flaws, and feature coverage. Results: It was observed that the proposed approach achieved greater value for code coverage in instruction, branches, lines of code and methods compared to state-of-the-art tools. The tool achieved the same result as state-of-the-art tools in terms of failures and crashes encountered. The proposed approach exercised more functional operations than the compared tools in functionality coverage. Conclusion: The proposed approach showed promising results, being more effective for navigation and executing operations in applications, generating useful and effective tests that exercise data entry variations that facilitate the coverage of mobile application functionalities

and with greater possibilities of identifying the presence of errors/failures.

Keywords: Mobile Application Testing, Reinforcement Learning, and Automated Testing.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Context

Mobile devices are an increasingly important part of our daily life, beyond simple communication and messaging to carry out many essential tasks, such as business, health and banking transactions systems. It is necessary to ensure software quality, reliability, and security on these devices as a large audience shall access them. Moreover, because of the fierce market competitiveness, the software industry must quickly release its products to the public at a low cost. Failures in mobile applications (apps) cause dissatisfaction and losses for software companies. In this sense, software app testing is necessary to prevent software failures from being found by the end-user. Manual app testing is the most popular technique for testing. However, it is more expensive, tedious, and error-prone (GAO *et al.*, 2014).

For such a reason, researchers are studying different aspects related to the automation for mobile applications (or simply mobile app testing). It becomes crucial and challenging since applications can run on other platforms, devices and operational systems. In the case of functional testing in mobile apps, they must help ensuring that the functionalities are following the requirements specified by the customer. In addition, they must behave correctly in the limited environment of a device that generally has memory constraint, processing, and data inputs but operates with a variety of standards of communication networks and services (3G,4G, GPS, NFC). For this reason, the validation should focus mainly on the application' behavior, flow navigation, mobile services, mobile web APIs (Application programming interface), transactions, and the user interface (GAO *et al.*, 2014).

Test automation in this scenario is necessary because it is possible to execute in a few second tests that would require many people, many devices, and hours to be manually performed, reducing costs and time. There are two critical aspects to the effectiveness of automated mobile app testing. Firstly, mobile apps should be cheaper than traditional software

for the web, and desktop (GAO *et al.*, 2014). On the other hand, they must be reliable and functionally correct. Automation is undoubtedly among the essential means to keep the testing cost low while ensuring an adequate degree of reliability. Secondly, current errors occur due to interface changes, interoperability issues between the application, framework, the operating system, and the hardware layers, as well as the influence of external conditions that affect the functionality of the application, such as notification interruptions and wireless data network (HALLER, 2013).

To reach these aspects is essential to work using support solutions that can enable automated testing of mobile apps so that their execution becomes more adaptive to changes, reliable and less dependent on human supervision and manual performance. Research in different strategies for mobile apps test case generation and execution is done using random, model-based, search-based, Machine Learning (ML) tools in state of the art. The disadvantages of the existing tools are coverage of functionalities, the tests are hard to be reproduced, and faults are hard to be localized (KONG *et al.*, 2019). One point to note is that these tools do not concern themselves with an important factor in software testing, the variation of test data input. There are several techniques and criteria in the literature as Equivalence Partition, Boundary Value Analysis, Systematic Functional Testing, and so on (DELAMARO MARIO JINO, 2016) about test data input selection, and according to the method used, the test cases can be more efficient to find failures.

In recent studies like (ADAMO *et al.*, 2018), (KOROGLU; SEN,), (PAN *et al.*, 2020) Reinforcement Learning (RL) is applied to app testing with promising results exploring app navigation and finding crashes. RL is the ML approach in which the agent learns through the environment interaction, exploration and trial-error (SUTTON; BARTO, 2018).RL has been used successfully to solve tasks as Atari games, Mario Bros game, or as complex as Go and Dota achieving superhuman performance. These studies in the field of test generation do not provide test cases to exercise the input variations, so the rules of input requirements are not covered. In this context, having many opportunities since the apps and environments have become more complex and demanding solutions on a large scale.

In this sense, this research proposes the DeepRLGUIMAT approach to generate test cases using the RL method Deep Q-Network (DQN) to perform functional tests, cover application functionalities, reveal failures, crashes and create test cases with a variety of input following Systematic Functional Testing. The tool DeepRLGUIMAT was developed to perform the proof of concept of this approach in Android mobile applications. Android platform was chosen because of the large market share and it is an open source platform supported by a large community.

1.2 Motivation

The technical literature presents some studies to understand how the industry has applied test automation activities of software. The results highlighted the benefits and limitations of automated software testing, such as the main benefits of test automation are reusability, repeatability, and effort saved in test executions (KIRUBAKARAN; KARTHIKEYANI, 2013). The limitations found are the high initial cost in designing the test cases, cost of changes in maintainability, buying a test automation tool, and training the staff.

Therefore, manual testing is adaptive when the app interface changes because the testers perform it, but automated tests must be rewritten when the software is modified. Software modifications are made rapidly in agile development environments hence rewriting the tests takes a large portion of the development time. Nevertheless, manual testing is also very time-consuming. It motivates the research of novel testing methods that are both adaptive and automated (KIRUBAKARAN; KARTHIKEYANI, 2013).

The mobile GUI (Graphical User Interface) automated testing is the process of using software tools to perform app testing with a graphical user interface to ensure correct behaviour and state of the GUI (KROPP; MORALES, 2010). It has two main challenges: i) Testing whether different devices provide an adequate data rendering, and ii) Testing whether native applications on different devices are correctly displayed. It is essential to make these test tasks automatic to avoid repetitive manually interacting with the GUI, which is time-consuming and costly. On the other side, they must be reproducible, reliable, and correct (KIRUBAKARAN; KARTHIKEYANI, 2013).

Regarding these challenges and limitations found in test automation of mobile apps, test case generation techniques have been studied mainly using model-based, search-based and random (Kong *et al.*, 2019). RL is the first Artificial Intelligence field to seriously address the computational problems that arise when learning from interacting with an environment to achieve goals (SUTTON; BARTO, 2018). So, RL techniques have been studied to be applied to test generation, initially showing promising results. Although the current solutions focus on app navigation and found crashes, they have limitations such as reproducing the failures; it takes a lot of manual efforts to inspect the results to find out the issues (WANG *et al.*, 2018); the tools performs the same actions generating repeated tests and lacks test input variation that imposes a challenge to create test cases to satisfy requirements (SAID *et al.*, 2020).

Considering the exploration and the various types of actions that can be combined in the interaction with applications, this research studies the RL technique DQN. It presents a good performance in continuous spaces and has mechanisms that avoid overfitting accessing past actions (memory replay) (PASZKE, 2017a). DQN does not need a database and learns through trial and error; the training journey of this algorithm already contributes to the exploration of the application. To RL efficiently vary many input types, it is not enough to perform several random

data inputs but to be guided by test techniques and criteria for competently input test data. In this sense, the Systematic Functional Test stands out for combining other techniques and has shown good results in mutation testing (LINKMAN; VINCENZI; MALDONADO, 2003). Thus, allowing the RL agent to generate tests feasible for the tester to attend app functionalities.

1.3 Objective

Based on this context, motivation and shortcomings presented previously, the main research question that guides this work is: How can Deep Reinforcement Learning technique be incorporated into automated testing of mobile applications to contribute to the greater code coverage, test input variation and detection of failures ?

This research aims to analyze the DQN Reinforcement Learning method and propose an approach, DeepRLGUIMAT, to create test cases through exploring the mobile application, identifying failures, and covering functionalities through test input variation, contributing to improving the test automation environment.

1.3.1 Specific Objectives

Following the specific objectives for this research.

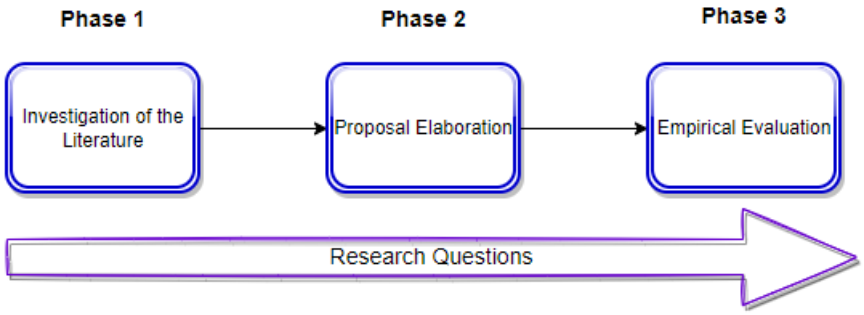
- Identifying approaches to allow the implementation of automated generation of test cases available in the technical literature;
- Analyzing the context patterns for RL in automated black-box testing of mobile applications;
- Defining the criteria to define a set of input actions of UI elements in mobile applications able to be automated;
- Creation of the approach that generates test cases in mobile applications
- Defining a method of evaluating the RL technique to identify metrics that prove the effectiveness of the proposed test tool

1.4 Methodology

The development of this research follows three phases as shown the figure 1. The first phase refers to a investigation of the literature, followed by a proposal approach in phase 2. Phase 3 to evaluate the proposal, was carried out through empirical evaluation.

Despite the methodological structure being demonstrated in sequential phases, the stages of this research, several times, take place in parallel, given that the bibliographic review is always

Figure 1 – Methodology



revisited and updated, from the beginning to the end of the work. Given this general perspective, the details of each phase of the research plan will be described in the following sections.

1.4.1 Investigation of the Literature

In this phase, the investigation of the literature was performed to discover the works that had a similar proposal. The systematic mapping (PETERSEN *et al.*, 2008) of machine learning in mobile application testing was conducted to identify the main trends in this area and the similar approaches used in functional test case generation for mobile applications. Thus, this type of review addresses a specific issue, uses explicit and transparent methods to conduct a thorough literature search and critical assessment of individual studies, and concludes what we currently know and don't know about a particular issue or topic. The results are detailed in chapter 3.

1.4.2 Proposal Elaboration

The preparation of the thesis proposal was carried out based on the knowledge acquired from the investigation of the literature, where the references for improving the testing automation were extensively analyzed and compared with each other. However, the detailing of the strategy for using the references that formed the conceptual basis of the work will be detailed along with the presentation of its proposal in Chapter 4.

1.4.3 Empirical Evaluation

In this phase we perform the empirical evaluation in which results are derived by observation or experiment instead of theory (CARBON; CIOLKOWSKI, 2007). The same evaluation pattern performed in the literature was followed, with the execution of a controlled experiment and comparison with similar solutions present in the literature, the detail of the evaluation is described in Chapter 6.

1.5 Organization

This thesis is organized as follows: Chapter 2, is presented the main conception of mobile applications, software testing, machine learning, and reinforcement learning trends in this field. In Chapter 3, a systematic Mapping Machine Learning based Mobile Application Testing is showed to provide a context of these fields and the related works found. In Chapter 4, the proposal approach DeepRLGUIMAT and the implementation aspects are described in. Chapter 5 presents the evaluation of the tool. Finally, in chapter 6, the conclusion and future work.

CHAPTER

2

BASIC CONCEPTS AND TERMINOLOGY

2.1 Initial Considerations

In this chapter, an overview of the topics that underlie the research of this thesis is presented. The organization of the chapter will be as follows. Section 2.2 presents a theoretical basis for contextualizing the study of software testing, a general introduction on the key elements of functional testing and main techniques are presented. In section 2.3 presents the basis for contextualizing the study of mobile applications; a general introduction on the key elements of mobile application is presented and the Android operating systems. Section 2.4 shows the mobile application testing is presented, highlighting the testing approaches, the types of mobile testing, testing tools, and the failures factors that make mobile testing different from the traditional test approach; the challenges and particularities present in this activity are also summarized. Section 2.5 presents the main concepts of Machine Learning; a general introduction to machine learning techniques is presented.

2.2 Software Testing

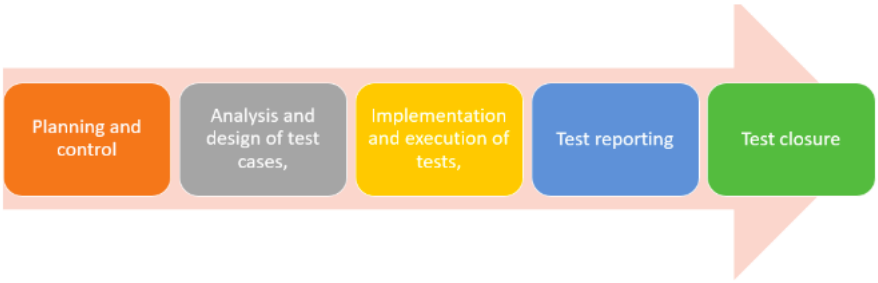
According to (MYERS, 2004) (2004), software testing is a process, or a series of processes, designed to ensure that computer code does what it was designed to do and does not do anything unintended. Software testing has become essential to companies to ensure the quality of the products regardless of whether the development methodology. The common terminology of software testing:

- Mistake: A human action that produces an incorrect result;
- Fault (or Defect): An incorrect step, process, or data definition in a program;

- Failure: The inability of a system or component to perform its required function within the specified performance requirement;
- Error: The difference between a computed, observed, or Measured value or condition and the true, specified, or theoretically correct value or condition;

The test process, in general, consists of the following activities throughout the software development: planning and control, analysis and design of test cases, implementation and execution of tests, test reporting, and test closure, Figure 2 (VEENENDAAL; GRAHAM; BLACK, 2008).

Figure 2 – Test Process



Source: Elaborated by the author.

Test managers or test leaders work with customers to establish the test objectives in the test planning phase. Then, a test plan document is generated to describe strategies, test scope, resources, methods, test techniques, completion criteria, and schedule of activities during the project. The test plan can be based on the standard format established by the (829-2008. . . ,). The test manager is responsible for the test team, ensuring the control of the test tasks established in the testing process.

In the analysis and design of test cases, the test objectives are turned into tangible test cases and test conditions. Test cases must be complete, reproducible, and independent. A test case contains identification, prerequisites, steps, and expected output. In addition, they must be designed to discover unexpected software failures. In this activity, test cases are created in a document called Test Case Specification (VEENENDAAL; GRAHAM; BLACK, 2008).

The environment must be configured to allow coding test scripts in the implementation phase. Then the test execution must be run, recording the results in a Test Execution Report and communicated to the project team. Once the test activities meet the exit criteria in the test plan, the activities such as results, logs, test documents related to the project are archived and used as a reference for future projects.

However, in addition to the test activities, the software testing also has different levels to which the software must be submitted: unit test, integration, system test, and acceptance test.

Thus, it allows focusing on testing the software under various development aspects, detecting several types of failures according to the technique used in each phase as described in Figure 3.

Figure 3 – Test Levels, (VEENENDAAL; GRAHAM; BLACK, 2008)

Level	Description
Unit Testing	Verify the functionality of a specific section of code at functional level.
Integration Testing	Tests the interfaces between component against a software design.
System testing	Test a completely integrated software system and verifies that it satisfies the requirement.
Acceptance Testing	It is performed as a part of hand-off process between any two phases of software development process.
Regression Testing	Tests the defects that are occurred after a major code change i.e. tests new functionality in a program.

2.2.1 Testing Design Techniques

This section briefly describes the main techniques that are used for software testing design.

2.2.1.1 Structural Testing

The structural testing technique is also called white box testing. It validates the software at the implementation level, testing the software's logical paths, conditions, loops, and the use of variables. Some criteria to use this technique are classified as following:

- Complexity-based criteria: The information about the program's complexity is used to derive test cases as the basic path complexity cyclomatic; this metric provides a quantitative measure of the difficulty of conducting the tests and an indication of reliability (DELAMARO MARIO JINO, 2016);
- Control flow-based criteria: control flow analysis is used as the source of information to derive the test cases. It determines how to test logical expressions (decisions) in computer programs. Decisions are considered as logical functions of elementary logical predicates (conditions) and combinations of conditions'values are used as data for testing of decisions Vilkomir, Kapoor and Bowen (2003);
- Flow-based criteria: These criteria derive test cases using the Def-Usage Graph concepts, which is the extension of the control flow graph. The information about the data flow

of the program is added, characterizing associations between points of the program in which a value is assigned to a variable and points where this value is used (DELAMARO MARIO JINO, 2016);

2.2.1.2 Black Box Testing

It is a technique used to design test cases evaluating the system as a black box and verify if the inputs and outputs conform to the customer requirements. In general, the system must addresses all possible data inputs. However, it can cause the test suites to be huge and even infinite(exhaustive test). Therefore, some functional test criteria have been defined to evaluate the system in the best possible way. Among the main functional test criteria are highlighted (VEENENDAAL; GRAHAM; BLACK, 2008):

- Equivalence Partitioning: In this technique, software inputs are divided into groups(partitions) that are expected to be similar behavior. The partitions identified can be valid data or invalid data to achieve the testing coverage goals. This technique can be applied to all levels of testing;
- Boundary Value Analysis: This technical analysis the behavior of the software at the edge of each equivalence partition. The maximum and minimum values are its boundary valid and invalid values to be tested. This technique also can be applied to all test levels;
- Decision Table: The decision table is used to analyze the conditions and actions of the system. Each table of table corresponds to a business rule that defines a combination of conditions resulting from the actions associated with that rule. The test cases are designed to cover each column, exercising combinations of conditions;
- State Transition: In this technique, a state transition diagram is built to analyze the aspects of the system, the inputs, events that are triggering changes (transitions). A state table shows the relationship between the states, inputs, and the valid or invalid transitions. It is used in embedded systems and also suitable for internet applications;
- Use Case Testing: The test cases are derived from use cases, their flows, regular scenarios, and alternative scenarios. It is useful to model the acceptance testing and can combine it with other techniques.

2.2.1.2.1 Systematic Functional Testing (SFT)

SFT attempts to combine the functional testing criteria (Equivalence Partitioning and Boundary Value Analysis). Once the input and output domain has been partitioned, STF requires at least two test cases of each partition to minimize the problem of co-incident errors masking faults. It also requires the evaluation at and around the boundaries of each partition (LINKMAN;

VINCENZI; MALDONADO, 2003). This approach presents guidelines for input data of various types of functions, input and output domains, as follows (VIDAL, 2011):

1. For discrete numerical values, input and output data must be considered and test cases are generated for each class.
2. For numerical value ranges, input and output data must also be considered, and the test cases are derived contemplating the limits and an internal value of each range.
3. Test cases with different values from the expected and with special cases must be generated, both for the input data and the output data.
4. Test Cases that exploit illegal values are needed to show that the software under test handles deviations from the success path. Thus, considering values outside the maximum and minimum limits of an interval, for example, are relevant test cases.
5. When creating test cases that include real numbers, specificity must be observed, this type of data does not have an exact limit. Normally, real numbers are entered in the base often, stored in a base of two, and finally retrieved in the base often, thus generating inconsistent data. Thus, test cases that address this situation must be created, adopting an error accuracy interval, with different limit values. In addition, test cases must be created that consider very small real values and the value zero.
6. For variable range values that depend on one or more variables, test cases must be created that cover all possible combinations of possible values.
7. When the information to be tested involves vector or data matrix, test cases that evaluate the data must be created: matrix size and matrix data. The size of the array must be tested, in all dimensions and all possible combinations, in its minimum, maximum and intermediate values. For array data values, the issues raised above should be considered.
8. In case of texts or data strings, it is necessary to validate the variation in size and validity of each character, considering the alphabet of characters only or the alphanumeric or even the punctuation only.

2.2.1.3 *Experience-based Testing*

In experience-based techniques, the primary source of information is the experience and knowledge of testers. Depending on the tester's approach and experience, these techniques may achieve widely varying degrees of coverage and effectiveness. Theses techniques are considered complementary to specification-based testing (ISTQB, 2018):

- Error guessing technique creates a list of possible errors, defects, and failures, and design tests that will expose those failures and the defects that caused them. These errors, defects

can build failure lists based on experience, defect, and failure data or common knowledge about why software fails;

- Exploratory testing is informal (not pre-defined) tests designed, executed, logged, and evaluated dynamically during test execution. The test results are used to learn more about the component or the system, and to create tests for the areas that may need more testing;

- Checklist-based testing, the testers design, implement and execute tests to cover test conditions found in a checklist. As part of the analysis, testers create a new checklist or expand an existing checklist, but testers may also use an existing checklist without modification;

2.2.2 Software Testing Automation

Software Testing Automation refers to the process of use some standard software solutions to control the execution of test-cases on the Software Under Test (KUMAR; MISHRA, 2016).It involves programming skills and knowledge in automation tools and frameworks. The investment in test automation is justified by the high cost and time-consuming manual test execution of large test suites. Other advantage is the increase of test coverage because multiple testing tools can be used at once allowing for parallel testing of different test scenarios. The main disadvantages include effort to choose and evaluate test tools and the cost of proprietary tools and code maintenance can be expensive.

There are approaches to automate test cases, among them, the use of Graphical interface-based tools that have the ability to record actions and execute them stands out. These are the tools known as rec-and-play (Record and Playback). In this approach, the tool interacts directly with the application, simulating a real user. As the application is being executed manually, the tool offers recording support: it captures the actions and transforms them into scripts. Another way widely used of automating test cases is Script Programming. In this approach, for unit tests, from largest to the smallest portions of code are tested: functions, methods, classes, components. In black-box testing, the UI elements code of the application is called, and the interaction with the graphical interface is programmed using a library or framework tool. The maintenance of the code is simpler and more productive. The code is often improved (Refactoring) and standardized. Professionals with knowledge of code and programming are required to create automated scripts (MEIRELES *et al.*, 2015).

Software testing automation tools can be divided into different categories as follows (KUMAR; MISHRA, 2016):

- Unit Testing Tools: It involves testing the most basic units of code, Developers write unit test cases in a programming language and the test cases can be executed automatically. Example of unit testing tools: JUnit, NUnit, and TestingNG.

- Functional Testing Tools: The functional testing tools receive input and examine the obtained output in comparison with the specified test oracle in the given test case. Some examples of Functional Testing Tools: Selenium, TestComplete, and Watir.

- Code Coverage Tools: It measures the number of lines, statements, or blocks of code tested using automated test suites. Code coverage testing is an essential metric to understand the quality of Quality Assurance. Some examples of tools: Cobertura, CodeCover, Jacoco, and EMMA.

- Test Management Tools: These tools are used to automate test activities such as test case specification, defect tracking, test report, and others. It also helps teams manage projects easily by providing a searchable and maintainable placeholder for test activates. Some examples of tools include Testlink, Mantis, and Jira.

- Performance Testing Tools: These tests tools aid to determine how the software will perform in terms of responsiveness and stability under various conditions and workloads. Example of tools: JMeter, Rational Performance Tester, HP LoadRunner.

Software testing automation frameworks (test automation frameworks) provide the basic set of software tools and services that can aid testers as they develop automated test cases. A good test automation framework should be general enough to provide functions that help a tester create automated tests for all the different components of the delivered software system (CERVANTES, 2009). There are different types of test automation frameworks, the author (KUMAR; MISHRA, 2016) considered the following main types:

- Linear Automation Framework: That is used to test user interface (UI). The tester records each step action such as navigation, user input, or checkpoints, and then playback to conduct the test in a sequential order without the need to write code to create functions.

- Modular Based Framework: A test script is created for each module of the software and then combined to build larger tests in a hierarchical approach. These larger sets of tests will begin to represent various test cases.

- Library Architecture Framework: It identified similar functionalities within the software that need to be tested and grouped them by function instead of dividing the software under test into modules to be tested in isolation each with its test scripts.

- Data-Driven Framework: In this approach the test data are separated from script logic and stored externally to an external data source, such as Text Files, Excel Spreadsheets, CSV files, SQL Tables, or ODBC repositories, thus allowing testers to test the same function or feature of software multiple times with different sets of test data.

- Keyword-Driven Framework: With this approach, similar to data-driven but keywords are also stored along with their associated objects in an external data source, making them independent from the automation tool being used to execute the tests. Keywords are part of a script representing various actions being performed to test the software.

- Hybrid Testing Framework: The hybrid framework combines two or more frameworks types set up to get the best practices of different frameworks suitable for a software project needs.

2.3 Mobile Applications

Smart devices as smartphones or tablets are in people's daily lives. According to the International Data Corporation (IDC) the researchers pointed to more than 1.4 billion mobile devices in circulation in the year 2015 and estimates an increase of 1.9 billion over the next four years (GAO *et al.*, 2014). Best known for smart devices, tablets, e-readers, and wearables, these devices have led software development to a new application class, mobile applications (MARTIN *et al.*, 2016). These applications performed on mobile devices require operating systems as Android(Developers, 2011) and Apple iOS (DEVELOPERS, 2021).

A mobile application (or mobile apps) is software designed to run on smartphones, cellphones, tablets, and other mobile devices considering the contextual input information. In mobile computing, an application is considered mobile if it runs on an electronic device that can move. The software for mobile computing has some constraints: limited resources, security, vulnerability, performance variability, reliability, and finite energy source. In the computing context, a mobile application is sensitive to the computing environment in which it is executed. It should adapt or react according to its user, environment, or time context. The contextual information can be categorized into human factors and physical environments (DELAMARO MARIO JINO, 2016).

According to (MASI *et al.*, 2012) (2012), mobile applications can be classified into three categories:

- Native Applications: it is software executed in the device's operational system (OS). These applications can access the APIs (Application Programming Interface) of the OS. The source code is developed using a development toolkit provided by the vendor, it is compiled, and the executable program must be embedded in the device Silva, Jino and Abreu (2010). The development and maintenance cost is high;

- Web Applications: It is related to apps that run on browsers. These applications have low-cost and resources, but they are limited and need internet connections to be executed;

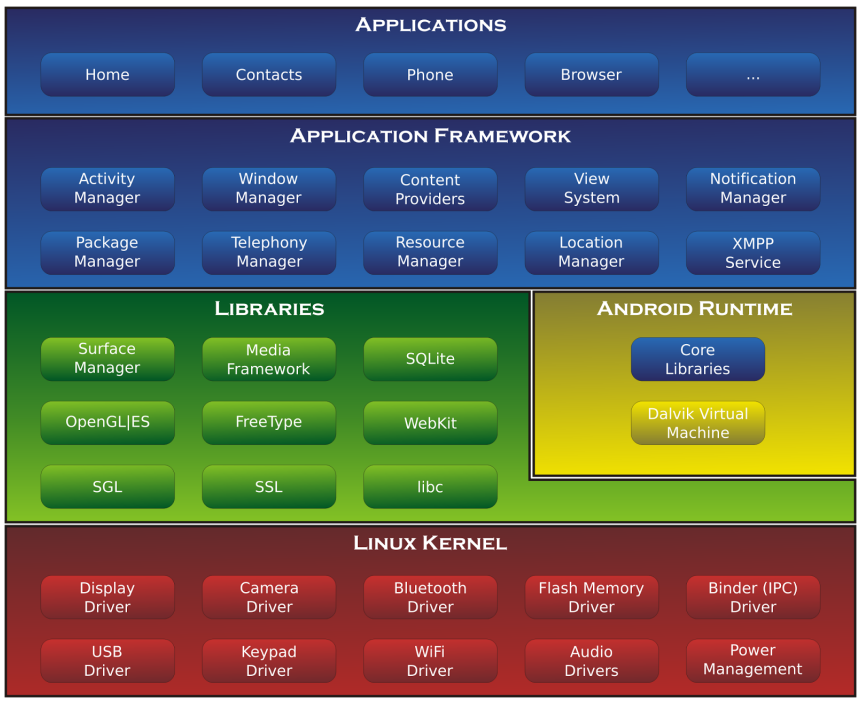
- Hybrid Applications: these apps combine native and web technologies.They often are written in HTML, JavaScript, CSS, and media files packaged and stored locally on the device, they have a lightweight interface, are portable, and can run offline in the device;

2.3.1 Android

The Android is an open-source operating system based on the Linux Kernel. Initially, it was developed by the company Android Inc acquired by Google in 2005. This operating system can be customized for hardware from different manufacturers like Samsung, LG, Asus, Motorola, etc. This platform attracts many developers and manufacturers, being since 2013 the leading platform in the mobile device market (ANDROID, 2021).

The Android user interface is based on direct manipulation, using touch input gestures that correspond to real-world actions to manipulate objects on the screen. It has internal hardware such as accelerometers, gyroscopes, and proximity sensors used by some applications to respond to additional user actions such as adjusting from portrait to landscape, depending on how the device is oriented. Android allows users to customize the home screen with application shortcuts and widgets, which allow users to view live content such as emails and weather information. According to an April 2017 StatCounter report, Android overtook Microsoft Windows to become the most popular operating system for total Internet usage. The Android architecture is component-based, as show the Figure 4.

Figure 4 – Android, source (ANDROID, 2021)



Like Linux for computers, the Kernel manages security, memory, and processes, in and out of files and network, and device drivers. The Linux kernel for android specifically

manages power, memory sharing, low memory killer, interprocess communication, etc. The Librarians have native libraries, Surface Manager, Display Manager, Audio / Video Framework, browser engine, graphics engine, and database. Android Runtime has two main components: Core java libraries and Dalvik Virtual Machine that runs the android applications. The Dalvik Virtual Machine is designed to reproduce the low-memory environment, slow CPU, and limited battery life. Application Frameworkkeeps control of application packages on the device, common interface, lifecycle and navigation, notifications, and so on (DEVELOPERS, 2021).

2.4 Mobile Application Testing

Mobile Application Testing refers to testing native mobile apps, device testing, mobile web application, and hybrid applications. Native application testing aims to validate the quality of mobile applications downloaded and executed on select mobile platforms on different mobile devices. The focus is on functionality, behavior, QoS (quality of service) requirements, usability, security, and privacy. Web application testing aims to validate the quality using different Web browser's diverse mobile devices. These applications usually provide users with a thin mobile client to access functions online from the server. Hybrid applications are combinations of native and web applications. They can run on devices offline and are written using web technologies like HTML5and CSS. The main testing approaches for mobile applications include (GAO *et al.*, 2014):

- Emulation-based Testing: this approach involves using a mobile device emulator in the computer simulating the mobile device. This approach is not expensive, but it has limitations as not possible to use all possibilities of gestures, the specific hardware characteristics, and the limited scale for testing QoS;
- Device-based Testing: It is a very costly approach since it is necessary to invest in the testing laboratory and purchase real mobile devices. It has many advantages, as testing mobile network, verify hardware interaction, functions, and environment. The disadvantages include the rapid changes of device models, not possible testing large scale of QoS because it depends on of many devices distributed what it is not feasible for the majority of enterprises;
- Cloud Testing: This approach is offered by many vendors. Basically, a mobile device cloud is built that can support testing services on a large scale. It also allows different mobile users to provide testing services cost-effective;
- Crowd-based Testing: A crowd-based testing infrastructure is built using freelance or contracted testing professionals. The testing process is managed in an ad hoc way with minimal mobile test automation tools. It is cost-effective, but the risk includes an uncertain quality level and schedule;

Mobile application testing is applied at each application level as in traditional software testing (unit testing, integration testing, system testing, and acceptance testing). As for test types just like traditional testing, structural and functional tests use the same test criteria.

However, taking into account the target device, approaches can be highlighted to assess the behavior of the software upon interruptions in the operating system or other applications (messages, notifications, calls, etc.), as well as tests with different types of connections (4G, WIFI, Airplane Mode), location (GPS), connectivity with other devices (Bluetooth communication, approximation) among other aspects (GAO *et al.*, 2014).

The authors (KIRUBAKARAN; KARTHIKEYANI, 2013) emphasize that apps are expected to receive inputs from different context providers (example, users, sensors, and connectivity devices). The inputs can vary from the different contexts the mobile device can move toward. The testing may lead to unpredictability and high variability of the application behavior is potentially receiving.

2.4.1 Supporting Tools for Mobile Application Testing

A wide range of studies has developed techniques to help mobile application developers improve mobile application testing, particularly attempting the improvement of UI and system testing coverage. One of the biggest challenges that researchers face in their current line of research on automated tests for mobile apps is that they cannot achieve high code coverage. Many problems are caused by a lack of engagement in proper regression tests, which causes frequent crashes, as not being able to install and start the software and buggier new versions. Device tests and automated testing can address the regression tests improving the application quality and customer satisfaction (NAGAPPAN; SHIHAB, 2016).

Automated mobile app testing raises two further issues: the lack of standardization in mobile test infrastructure, scripting languages, and connectivity protocols between mobile test tools and platforms; and the lack of unified test automation infrastructure and solutions that cross platforms and browsers mobile devices. In test automation for functional system testing in mobile applications, it is important to automatically execute scripts captured during the user interaction and replayed and modified even on different devices. It is necessary to make this task automatic to avoid manually interacting with the GUI that is a time-consuming task (KIRUBAKARAN; KARTHIKEYANI, 2013).

The automated unit testing for unit testing in mobile applications, there are supporting tools for each platform, such as Android Test, which is a framework that includes several levels of unit testing is an integral part of the Android development environment (DEVELOPERS, 2021).

The tool Robolectric is also a unit testing framework specific to Android. It runs the tests inside the JVM and allows you to load native Android features so that real device tests

are performed (ROBOLECTRIC, 2021). Another framework used in several platforms is xUnit allows performing unit tests to determine if the sections of the code are processing expectedly in diverse circumstances and are available for the main languages like ASP, C++, C, Delphi, Java, Perl, and PHP (JUNIT.ORG, 2021).

Table 1 – Automated Mobile Test Tools

Tools	Platform	App Type	Environment	Record & Replay	Cloud Service	Free
Appium	Android/IOS	Native/Web/ Hybrid	Device/ Emulator	no	no	yes
Calabash	Android/IOS	Native/Hybrid	Device/ Emulator	yes	no	yes
EggPlant	Android/IOS	Native/Web/ Hybrid	Emulator	no	yes	no
Espresso	Android	Native	Device/ Emulator	no	no	yes
MITE	Android/IOS	Web	Device/ Emulator	yes	no	no
Monkey	Android	Native/Hybrid	Device/ Emulator	no	no	yes
Monkey Talk	Android/IOS	Native/Web/ Hybrid	Device/ Emulator	yes	yes	no
Perfecto Mobile	Android/IOS	Native/Web/ Hybrid	Emulator	yes	yes	no
Robot Framework	Android/IOS	Native/Web/ Hybrid	Device/ Emulator	no	no	yes
Robotium	Android	Native/Hybrid	Device/ Emulator	no	no	yes
SeeTest Mobile	Android/IOS	Native/Web/ Hybrid	Device/ Emulator	yes	no	no
Selendroid	Android/IOS	Native/Web/ Hybrid	Device/ Emulator	yes	no	yes
Sikuli	Android/IOS	Native/Web/ Hybrid	Emulator	yes	no	yes
UI Automator	Android	Native	Device/ Emulator	no	no	yes

The test automation for functional testing is a challenge in mobile apps since it is difficult on all possible devices. The existing simulators cannot simulate real-world phones with their sensors, GPS, among other internal characteristics. Many tools for simulating functional testing use the technique capture-and-replay for recording, executing, and replay contextual inputs selected during the test election phase. It is important to consider the increased success of multiple platforms and now many cross-platform applications. Additionally, all the platforms need to run on different hardware with different versions of the OS. Thus even if the application is automated tested on one device, there is no guarantee that it may work on another device.

However, these problems are not new. A study by (JOORABCHI; MESBAH; KRUCHTEN, 2013) describes a tool, CHECKCAMP, that tests for inconsistencies between iOS and Android versions of mobile apps using extracted abstract models. Currently, automated tools should consider this scenario.

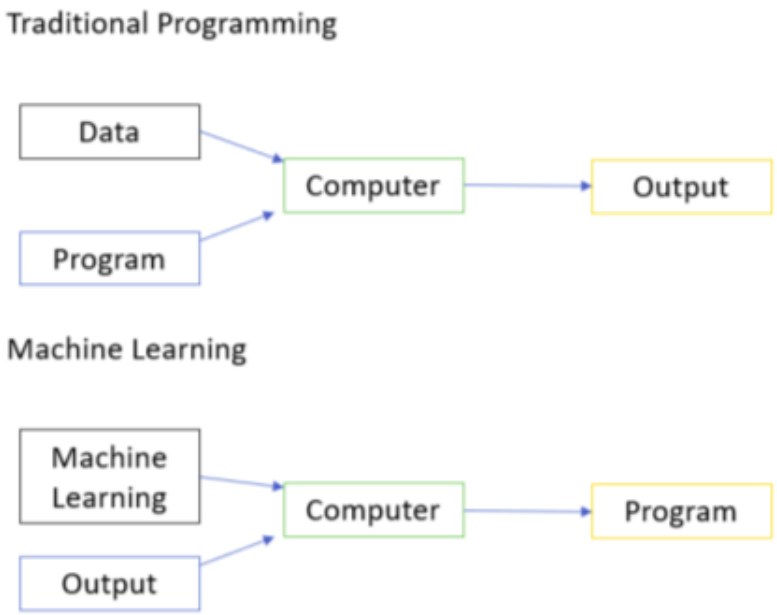
Table 1 shows some popular test tools and their characteristics of these tools as Gui-based function, which is related to create test scripts to validate the application user interface. The native attribute is about tools that enable automated testing of native applications. The Web is regarding the apps which can run using browsers and network connection. The Record and Play relate to record the interaction steps of the tests and execute them after the script is created. The Environment attribute is about execution in Devices and Emulators. Cloud service is about tools that offer to test in the cloud. Free or Open Source is the tools that are open source.

These key attributes to choose the tool must be considered especially when it is a proprietary tool. In the case of open-source tools, the company has the option to customize them to suit your needs better. There is a growth of tools that use cloud computing for its execution in the market, also offering services of tests in the cloud. This trend is driven by the need for scalability in testing and various models, brands, and platforms for mobile devices.

2.5 Machine Learning

According to (MITCHELL, 1997)(1997), "Machine Learning (ML) is the ability to improve performance in executing some task through experience." It means computers are

Figure 5 – Machine Learning differentiates from Programming



Source: Elaborated by the author.

programmed to learn from past experiences. It is used a principle of inference called induction, in which one obtains generic conclusions from a particular set of examples. Figure 5 illustrates how ML differentiates from traditional programming.

The Machine Learning algorithms aim to learn from a data subset called a training set, a model or hypothesis able to relate input attribute values with output attributes. An important requirement is the capability to work with imperfect data. It means data noises, redundancy, or absent data. Its turn, the usage of pre-processing data techniques is important to identify the problems. The inductive biases important, since when the algorithm is learning from a subset of data training, it is looking for a hypothesis between a space of possible hypotheses, able to describe the relationship of objects that best match with data training (FACELI *et al.*, 2011).

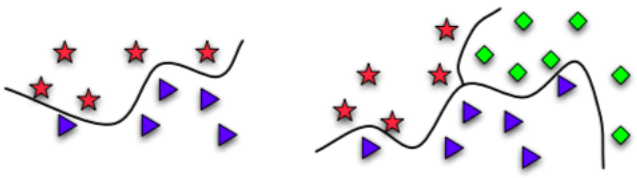
Learning is a wide domain. Consequently, machine learning has branched into several subfields dealing with different types of problems and learning tasks. The four parameters along which learning types can be classified are described below.

2.5.1 Supervised Learning

This is called supervised because a " teacher knows the output." The goal is to learn a general rule that maps inputs to predict outputs. The teacher evaluates the ability of the inducted hypothesis to predict the output for new samples. The tasks of Supervised Learning (SL) can be Classification and Regression (MITCHELL, 1997).

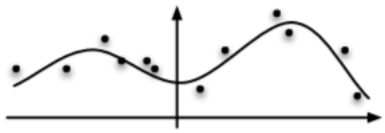
In classification, the goal is to predict what class an instance of data should fall into, discrete output. The Classification can be binary, the Multiclass Classification is the logical extension of binary classification demonstrated in Figure 6 (SMOLA A. J.; VISHWANATHAN, 2020 (accessed June 3, 2020)).

Figure 6 – Left: binary classification. Right: 3-class classification (SMOLA A. J.; VISHWANATHAN, 2020 (accessed June 3, 2020))



In Regression, the goal is the prediction of a numeric value. The output is continuous. The goal is to estimate a real-valued variable, Figure 7. An example is a system that can predict the price of a used car. Inputs are the car attributes brand, year, engine capacity, mileage, and other information that we believe to affect a car's worth. The output is the price of the car (MITCHELL, 1997).

Figure 7 – Regression (SMOLA A. J.; VISHWANATHAN, 2020 (accessed June 3, 2020))



SL has several algorithms, among the most popular, are highlighted below (Singh et al., 2016):

- Naïve Bayes classifier is based on Bayes' theorem with strong (naive) independence assumptions between the features. It analysis each attribute and assumes that all of them are independent;
- Support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite-dimensional space. It considers achieving a good separation when the hyperplane has the largest distance to any class's nearest training data point. In general, the larger the margin, the lower the generalization error of the classifier;
- Logistic Regression: the dependent variable is categorical. Situations where the dependent variable has more than two outcome categories may be analyzed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression;
- Decision tree classifier is based on the decision tree to go from observations as the decision node, branches, and leaves. The mechanism is transparent and the structure can be followed to see how the decision is made.
- K-nearest neighbors (k-NN) classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.
- Neural Network Multilayer Perceptron (MLP) is a class of feedforward artificial neural network (ANN), consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.

2.5.2 Unsupervised Learning

In Unsupervised Learning (UL), there's no label or target value given for the data. There is no evaluation of the accuracy of the structure that is output by the relevant algorithm. The goal of this ML type is learning to group similar items together. It is a task called clustering. In UL, the statistical values that describe the data are found. This is a task known as density estimation. Another task of UL is reducing the data from many features to a small number to visualize it

properly in two or three dimensions (SMOLA A. J.; VISHWANATHAN, 2020 (accessed June 3, 2020)).

The main algorithms of UL are described below (MITCHELL, 1997):

- Clustering K-means aims to partition n observations into k clusters so that each observation belongs to the cluster with the nearest mean, serving as a cluster prototype;
- Hierarchical Clustering aims to find groups such that instances in a group are more similar to each other than in different groups. It has two strategies, Agglomerative (bottom-up) and Divisive (top-down);
- Anomaly detection aims to identify the items, events, or observations which do not conform to an expected pattern or other items. It is used mainly in applications for intrusion detection, fraud detection, and fault detection;
- Neural Network - Generative adversarial networks generative network learns to map from a latent space to particular data distribution. In contrast, the discriminative network discriminates between instances from the generator's true data distribution and candidates.

2.5.3 Semi-supervised Learning

Semi-supervised Learning is suitable to tackle training sets with large amounts of unlabeled data and a small quantity of labeled data. The goal is to learn a predictor that predicts future test data better than the predictor learned from the labeled training data alone. The acquisition of self-labeled data follows an iterative procedure, aiming to obtain an enlarged labeled data set, in which they accept that their own predictions tend to be correct (GONZÁLEZ *et al.*, 2018).

The brief of main methods for semi-supervised learning is presented below (CHAPELLE; SCHOLKOPF; ZIEN, 2009):

- The Generative models assume that the distributions take some particular form are parameterized by a vector. If these assumptions are incorrect, the unlabeled data may actually decrease the accuracy of the solution relative to what would have been obtained from labeled data alone;
- Low-density separation This method aims to label the unlabeled data such that the decision boundary has a maximal margin over all of the data;
- The Graph-based methods use a graph representation of the data. It has a node for each labeled and unlabeled example. The graph may be constructed using domain knowledge or the similarity of examples;

- Heuristic approaches are commonly used with a supervised learning algorithm trained based on the labeled data only. Then, this classifier is applied to the unlabeled data to generate more labeled examples as input for the supervised learning algorithm.

2.5.4 Reinforcement Learning

In this ML type, there is, for example, a decision-maker called the agent that is placed in an environment. The environment is in a certain state that is one of a set of possible states. Typically, the environment is formulated as a Markov decision process. The decision-maker has a set of actions possible, and once an action is chosen and taken, the state changes. The solution to the task requires a sequence of actions. It gets feedback in the form of a reward (ALPAYDıN, 2011).

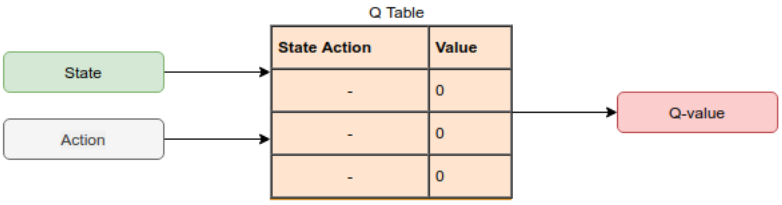
Typically in RL, the environment is formulated as a Markov decision process (PUTERMAN, 1990), which is defined by (SUTTON; BARTO, 2018):

- A set of possible states: S
- A reward function for the next state given a (state, action) pair: $R(st, at, st+1)$
- A transition probability i.e distribution over the next state given a (state, action) pair: $T(st+1|st, at)$
- A discount factor $\gamma \in [0,1]$, where lower γ emphasizes more on immediate rewards

The decision-maker has a set of possible actions; once an action is chosen and taken, the state changes. The solution to the task requires a sequence of actions. It gets feedback, in the form of a reward (SUTTON; BARTO, 2018). RL has been used successfully to solve tasks as Atari games, Mario Bros game, or as complex as Go and Dota achieving superhuman performance. Q-Learning is a model-free RL algorithm to learn a policy telling an agent which action to take under certain circumstances, as shown in Figure 8. The main idea is $Q^* : State \times Action \rightarrow \mathbb{R}$ taking action in a given state, then its possible to construct a policy that maximizes the rewards (WATKINS; DAYAN, 1992). It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards without requiring adaptations. For finite Markov decision process (FMDP), Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward overall successive steps, starting from the current state. This algorithm can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy (MELO, 2001).

Deep Q-Learning algorithm can be cast as an extension of the Q-learning algorithm that uses a deep neural network to approximate the action-value function. DeepMind developed the DQN (Deep Q-Network) algorithm in 2015. It solved a wide range of Atari games by combining reinforcement learning and deep neural networks at scale. The algorithm was developed by

Figure 8 – Q-Learning.



enhancing a classic RL algorithm called Q-Learning with deep neural networks and a technique called experience replay. In DQN, decision-making regarding which action to take involves a fundamental choice. Exploitation makes the best decision given the current information, and Exploration gathers more information and explores possible new paths. In exploration, ε-Greedy policy allows the AI agent to take random actions from the action-space with a certain probability ε (FAN *et al.*, 2020).

DQN uses a technique called Experience Replay; it uses replay memory to store the trajectory of the Markov decision process (MDP). At each iteration of DQN, a mini-batch of states, actions, rewards, and next states are sampled from the replay memory as observations to train the Q-network, which approximates the action-value function (FAN *et al.*, 2020).

Figure 9 – Pseudocode Deep Q-Network.

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise state s_t
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Set $s_{t+1} = s_t$
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
  end for
end for

Figure 9 shows the DQN algorithm. The *episode* means one a sequence of states, actions, and rewards, which ends with a terminal state (a terminal state can be a completed mission or a failure). For each action selected and performed, the reward and the next state are observed. The replay memory stores the state, action, reward, and the next state and samples at random from D when performing updates. This approach is limited since the memory buffer is overwritten by recent transitions due to the finite memory size N (PASZKE, 2017b).

2.6 Final Considerations

The Software Testing area has adapted to the new software platforms, called mobile application Testing. In general, research in this area presents some techniques and supporting tools for different types of testing in mobile apps. However, there is a tendency to adapt strategies and tools developed for other platforms. As has been observed, mobile app testing has challenges and difficulties like different requirements, types, and techniques. This area also has several subdivisions, some of which are little explored. Many challenges and barriers to overcome in mobile app testing and Artificial Intelligence techniques such as Machine Learning are studied as an excellent resource to the evolution of software testing support tools. The next chapter will present the systematic mapping of machine learning and mobile app testing.

CHAPTER

3

MACHINE LEARNING-BASED MOBILE APPLICATION TESTING MAPPING

3.1 Initial Considerations

The popularity of smartphones and the market demand are challenging to assure the quality of the mobile applications (apps). Manual test in a variety of devices and operating systems is costly. For this reason, research in test automation and artificial intelligence techniques (ML) are highlighted. To find the related works in the context of ML-based app testing, we conducted a Systematic Mapping (SM) to find the trends and challenges in this area. The results were categorized as ML type, ML algorithm, app testing level, app type, operational system, and app testing research challenges. We analyzed 31 primary studies addressed to ML-based app testing. Supervised Learning is the most ML type used to defect detection, vulnerability and cost prediction, test case prioritization, and GUI error recognition. Investigations in the system test level are highlighted mainly in GUI test generation. Studies indicate promising results in failure prediction, detection, and test generation. However, research has been restricted to native Android apps. The limitations of ML include performance, availability of large amounts of data, and lack of test oracle.

3.2 Systematic Mapping

A systematic mapping (SM) can be defined to build a classification scheme and structure for a software engineering field of interest. The analysis of results focuses on frequencies of publications for categories within the scheme (PETERSEN *et al.*, 2008). This research aims to set out to review the state of the art of how ML has been explored to automatically streamline mobile application testing and provide an overview of the research intersection of those two fields by conducting a systematic mapping review.

The methodology for retrieving state of the art comprises four planning phases; the definition of research questions as part of the objective, the definition of selection criteria, the creation of a search string, and the determination of reference databases. The research question can be reflected through the representation of PICOC (PETERSEN *et al.*, 2008), described in Table 2 below:

Table 2 – PICOC

Population	Mobile Application Testing
Intervention	Machine Learning
Comparison	N/A
Outcomes	machine learning based mobile application testing
Context	Reviews all empirical studies of machine learning in the domain of mobile application testing

3.2.1 Research Questions

The Research Questions (RQ) emphasizes the literature classification so that the community of researchers and professionals gives them insights into how ML has been addressed to mobile app testing. According to the goal of our study, the research questions of this study are shown in Table 3.

Table 3 – Research Questions

ID	Question
RQ1	What type of ML techniques have been used to cope with mobile app. testing? **RQ1.1** Which ML Algorithms?
RQ2	Which testing levels are automated by ML algorithms? **RQ2.1** Which Testing techniques and types?
RQ3	Which Mobile application Type and Operational System ?
RQ4	Which mobile app. testing challenges are treated by ML?
RQ5	What advantages and limitations are found on ML based mobile app. testing?

The main idea of RQ1 is to identify the ML types that have been applied in mobile app testing. RQ1.1 complements with information of which ML algorithm is the most applied and how. RQ2 focuses on identifying which mobile app test level are automated by ML, in RQ2.1 extend to know the testing techniques and test types, important information for testing professionals. RQ3 identifies the mobile app type where ML is applied, such as native, web, or hybrid, and the Operational system (Android or IOS) and, for consequence, to know the platform limitation of the studies.RQ4 classifies the test challenges that have been studied with the application of ML, and RQ5 shows the advantage and limitations observed by the studies, which can help researches.

3.2.2 Search Process

In this phase, automated searching using five digital reference libraries was performed. According to the relevance in the field and to obtain good results, the libraries are:

1. Elsevier's Scopus
2. ScienceDirect
3. IEEE Xplore Digital Library
4. ACM Digital Library
5. Web of Science

The search protocol was created considering the combination of keywords of three main aspects: machine learning, mobile applications, and testing. Therefore, the search string has three parts, as shown: The first part is explicitly related to terms of ML linking them using the operator OR. Furthermore, the mobile application terms and software testing terms and variations were included. The three parts are linked with the operator AND.

- *(("machine learning" OR "supervised learning" OR "unsupervised learning" OR "reinforcement learning" OR "q-learning" OR "natural language" OR NLP OR "Text mining" OR predict OR prediction OR classification) AND ("mobile application" OR "mobile app" OR "mobile software" OR "Android" OR "IOS") AND (test OR testing))*

This string could be applied entirely in IEEExplore, ACM Digital Library, Scopus, and Web of Science. For Science Direct, the search has a limitation of 8 boolean connectors, so we divided into two search strings to perform the search and not to miss any potential result:

1. *(("machine learning" OR "supervised learning" OR "reinforcement learning" OR "q-learning" OR "predict") AND ("mobile app" OR "Android" OR "IOS") AND Test)*
2. *(("machine learning" OR "natural language processing" OR "Text mining" OR classification) AND ("mobile app" OR "mobile software" OR "Android" OR "IOS") AND Test).*

In Scopus, Science Direct, and Web of Science, we filtered to exclude results of areas not linked, such as Medicine, Biomedicine, Humanities, Healthy, Psycology, Mechanics, Neuroscience, Hardware and other related.

These criteria formed the homogeneous selection of studies during selection phases. The research did not restrict the selection regarding the publication date to identify all available papers based on the definition criteria until the execution of this search in early 2021.

The search process was conducted using the string created in the automated search of selected libraries. The automated search had no restriction to year. For libraries ScienceDirect, Web of Science, and Scopus, the Computer Science area filtered the results.

The search string returned a total of 3049 results. Table 3 shows the results returned separated by the library.

Table 4 – Search Results per Library

Library	Papers
ACM Library	162
IEEE	571
ISI Web of Science	305
Science@Direct	648
Scopus	1363

The technique snowballing backward and forward were applied in order to identify similar studies in the references of selected papers and search for new studies through citations. Snowballing is an approach for selection of papers, it requires a set of initial articles to start the process. There are two approaches: backward snowballing, which selects papers referenced by the initial set of papers. The other approach is forward snowballing, which selects papers that do cite the set of papers. The selected papers from an iteration of the snowballing algorithm compose the group of articles, which will be the seeds for the following iteration. The process continues until a stopping criterion is satisfied (BADAMPUDI; WOHLIN; PETERSEN, 2015).

3.2.3 Primary Study Selection Process

This section presents the inclusion (IC) and exclusion criteria (EC) used in the secondary study. The inclusion criteria were specified as follows:

- (IC1) The result must be fully available
- (IC2) The result must be in English
- (IC3) A concrete approach is described or a survey about machine learning and mobile application testing is provided.

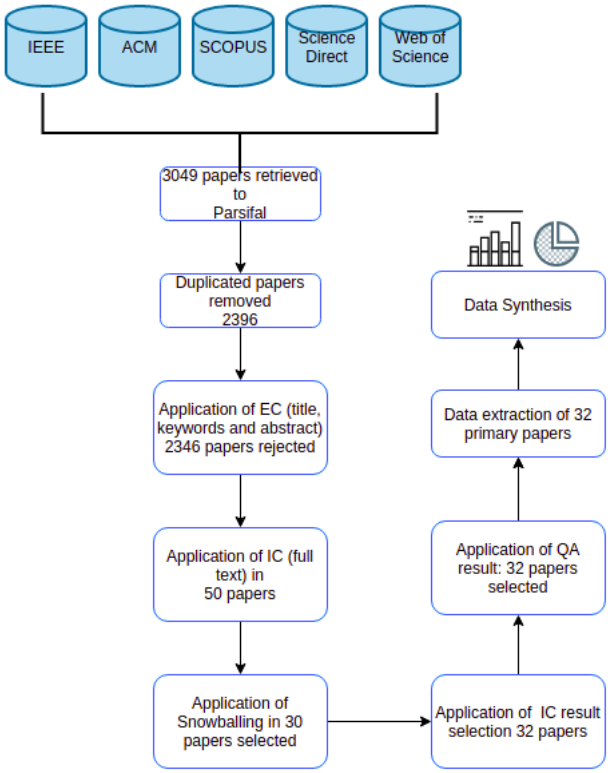
The exclusion criteria were also specified as follows:

- (EC1) The result is not in the context of mobile application testing.
- (EC2) The result is in the context of hardware, network security and operational system security.

- (EC3) Studies related to testing embedded systems in general, and not running on mobile devices.
- (EC4) Review papers and studies such as surveys, SLR, and SMS related to the topic.
- (EC5) The result is not in the context of mobile application testing.
- (EC6) The paper has only the character of a (advertisement) brochure without details.
- (EC7) The paper is sketchy or under reported, such as a one page or a vision paper.

The *Selection Process* had the following stages: First, the duplicated papers are removed, then the EC is applied to filter the studies by title, keywords, and abstracts. The papers accepted are verified according to the IC. The papers selected were examined to cope with quality assessment, reading them entirely. Figure 10 shows this selection process in detail.

Figure 10 – Selection Process



3.2.4 Study quality assessment

We evaluated the quality assessment (QA) of the studies after the selection process. The following checklist was used to assess the reliability and thoroughness of the selected papers.

- Q1: Does the study clearly present the application of ML in mobile app testing?

- Q2: Does the study clearly present the ML type and technique defined?
- Q3: Does the study clearly propose a solution to mobile app testing issue?
- Q4: Does the study present evaluation strategy and results ?

Each of the questions was answered as “Yes” “Partially,” or “No”. We attributed the punctuation value for each answer Yes corresponds to 1.0, Partially is 0.5, and 0 for No. We considered a study as partial in cases where the methodological details could have been derived from the text, even if they were not clearly described. Its quality score was computed by summing up the punctuation of the answers to all four questions. The quality classification level into High (score value=4), Medium (score value from 2 to 4), and Low (score value less than 2).

3.2.5 Data Extraction

In this study, to answer the RQs, a data extraction form was created that includes the fields designed to bring together publication information, such as title, affiliation, year of publication, and the fields included to understand and answer the RQs. The data extracted were stored in a spreadsheet and checked to ensure that information is valid for analysis. The data items (DI) are described in Table 5.

3.3 Results and Data Synthesis

The data synthesis activity involves compiling the data extracted from each primary study included in the SM.The visual synthesis and classification of the selected studies were summarized to answer each RQs described in section 3.2.1.

According to the inclusion and exclusion criteria, the studies were selected using the tool Parsifal (PARSIFAL, 2020 (accessed June 3, 2020)). In the total of 3049 papers returned, 653 duplicated papers were removed. 2396 papers were rejected by title and abstract, and 50 papers were accepted to be fully analyzed. According to the selection criteria, 30 papers were selected. The snowballing returned 2 studies (S18, S22) added, and a total of 32 papers were fully accepted. The Table 5 shows the primary studies selected.

Regarding the year of the publications, in Figure 11, the year 2018 had the highest number of publications in this field of study. In 2019, there was a drop in publications. In 2020 a modest growth.

Despite the popularity, from 2018, there were no big changes in the performance of ML algorithms. They still demand a learning curve for professionals to combine testing and ML, computational processing, and good hardware resources be executed, which can hinder the development of tests tools.

[0] *Snowballing results

Table 5 – Data Items

	Title	**Description**	**RQ**
DI1	ID	identification of paper	RQ1
DI2	TITLE	title of paper	RQ1
DI3	AUTHOR	author of the paper	RQ1
DI4	YEAR	publication year	RQ1
D15	COUNTRY	publication country	
DI6	PUBLICATION SOURCE TYPE	publication source type such as conference of software engineering and testing or journal of Artificial Intelligence area	RQ1
DI7	EVALUATION TYPE	type of study such as study case, experiment, experience report, systematic review	RQ1-4
DI8	ML TYPE	machine learning technique classification such as supervised, not supervised, reinforcement learning and deep learning	RQ1
DI9	ML ALGORITHM	the ml algorithm used such as support vector machine, decision three, neural network, q-learning etc.	RQ1
D10	TESTING LEVEL	the software testing level such as system test, unit testing, test integration	RQ2
DI11	TESTING TECHNIQUES	software testing techniques e.g. functional, non-functional, structural.	RQ4
DI12	TESTING TYPES	software testing types , e.g. black box testing, security testing, compatibility testing.	RQ2.1
DI13	TESTING APPROACH	approach used to testing applications, e.g. GUI testing, code features and attributes.	RQ2.1
DI14	TESTING CHALLENGES	research challenges in software testing, test case generation, defect prediction, test input generation etc	RQ5
DI15	APPLICATION TYPE	type of mobile application: native, hybrid or web	RQ3
DI16	OPERATIONAL SYSTEM	android, ios or other	RQ3
DI17	ADVANTAGES	Advantages of the studies and results	RQ4
DI18	DISADVANTAGES	disadvantages, weakness of the study	RQ4

Figure 12 shows the distribution of research papers by country. Most of the studies are from China (11), followed by the USA (5), India (3), Italy (3), Japan (2), Turkey (2), Indonesia (1), Germany (1), Netherlands(1), Taiwan (1) and Malaysia (1).

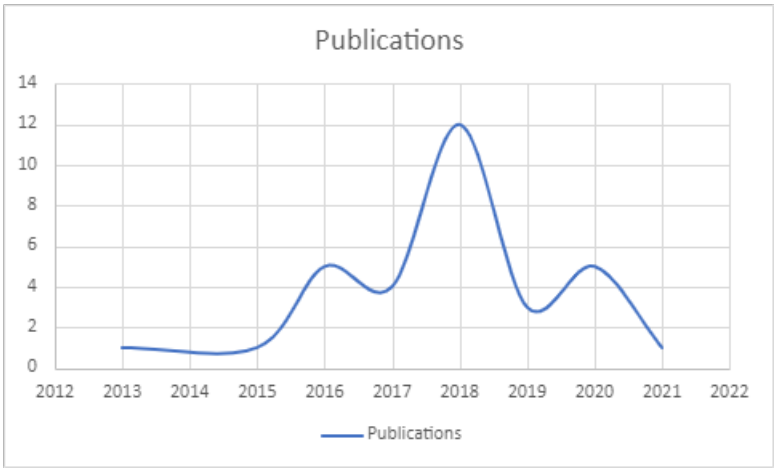
Table 7 presents the publication sources of the selected papers, 19 are conference papers, 2 are workshops, and 11 are journal papers. The source of publications was classified by results from Testing and Software Engineering related conferences, Testing, Software Engineering related journals, Software Engineering related Workshops, Mobile Software related conferences, Artificial Intelligence related journals, Computer Science related journals, Mobile software related Conferences, Eletronic Journals, Eletronic Conferences, Security related Conferences, Security related Journals and Multidisciplinary journal. Figure 13 shows the distribution of studies by publication source type. Regarding evaluation type, 30 studies presented empirical

Table 6 – Primary Studies

ID	Title	Authors	Year
S1	QBE: QLearning-Based Exploration of Android Applications	Y. Koroglu et al.	2018
S2	Reinforcement learning for android GUI testing	Adamo, David & Khan, Md & Koppula, Sreedevi & Bryce, Renée.	2018
S3	A reinforcement learning based approach to automated testing of android applications	Thi Anh Tuyet Vuong and Shingo Takada.	2018
S4	Humanoid: A deep learning-based approach to automated black-box android app testing	Y. Li, Z. Yang, Y. Guo and X. Chen	2019
S5	Combining Automated GUI Exploration of Android apps with Capture and Replay through Machine Learning	Domenico Amalfitano, Vincenzo Riccio, Nicola Amatucci,Vincenzo De Simone, Anna Rita Fasolino	2018
S6	Automation of Android applications functional testing using machine learning activities classification	A. Rosenfeld, O. Kardashov and O. Zang	2018
S7	AIMDROID: Activity-insulated multi-level automated testing for android applications	T. Gu et al.	2017
S8	Guided GUI testing of android apps with minimal restart and approximate learning	Wontae Choi, George Necula, and Koushik Sen.	2013
S9	UIChecker: An Automatic Detection Platform for Android GUI Errors	M. Ji	2018
S10	Semantic analysis for deep Q-network in android GUI testing	Vuong, Thi & Takada, Shingo.	2019
S11	AppFlow: Using Machine Learning to Synthesize Robust, Reusable UI Tests	Gang Hu, Linjie Zhu, and Junfeng Yang.	2018
S12	Deep Learning-Based Mobile Application Isomorphic GUI Identification for Automated Robotic Testing	T. Zhang, Y. Liu, J. Gao, L. P. Gao	2020
S13	Guiding App Testing with Mined Interaction Models	Nataniel P. Borges, Maria Gómez, and Andreas Zeller.	2018
S14	Automatic Text Input Generation for Mobile Testing	P. Liu, X. Zhang, M. Pistoia, Y. Zheng, M. Marques and L. Zeng,	2017
S15	Learning to Prioritize GUI Test Cases for Android Laboratory Programs	Cheng-Zen Yang, Yuan-Fu Luo, Yu-Jen Chien, and Hsiang-Lin Wen.	2016
S16	Prioritizing Test Cases for Memory Leaks in Android Applications	Qian, J., Zhou, D.	2016
S17	Cross-Project Just-in-Time Bug Prediction for Mobile Apps: An Empirical Assessment	G. Catolino, D. Di Nucci and F. Ferrucci	2019
S18	*Mobile application software defect prediction	M. Y. Ricky, F. Purnomo and B. Yulianto	2016
S19	Software fault prediction based on change metrics using hybrid algorithms: An empirical study	Wasiur Rhmann, Babita Pandey, Gufran Ansari, D.K. Pandey	2020
S20	An empirical framework for defect prediction using machine learning techniques with Android software	Ruchika Malhotra.	2016

ID	Title	Authors	Year
S21	Application of machine learning on process metrics for defect prediction in mobile application	Kaur, Arvinder & Kaur, Kamaldeep & Kaur, Harguneet.	2016
S22	*High-Frequency Keywords to Predict Defects for Android Applications	Y. Fan, X. Cao, J. Xu, S. Xu and H. Yang	2018
S23	Defect Prediction in Android Binary Executables Using Deep Neural Network	Feng Dong, Junfeng Wang, Qi Li, Guoai Xu, and Shaodong Zhang.	2018
S24	Compatibility Testing Service for Mobile Applications	T. Zhang, J. Gao, J. Cheng and T. Uehara	2015
S25	Vulnerability Detection on Mobile Applications Using State Machine Inference	W. van der Lee and S. Verwer	2018
S26	Dynamic Loading Vulnerability Detection for Android Applications Through Ensemble Learning	T. Yang, H. Cui and S. Niu.	2017
S27	New deep learning method to detect code injection attacks on hybrid applications	Ruibo Yan, Xi Xiao, Guangwu Hu, Sancheng Peng, Yong Jiang.	2018
S28	A Novel Android Application Penetration Analysis Method	Zengshuai, Hao & Leizi, Meng & Xiong, Zhan & Jie, Wang & Jianbo, Yu.	2017
S29	Reinforcement learning-based curiosity-driven testing of Android applications	Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, Xuandong Li	2020
S30	Functional test generation from UI test scenarios using reinforcement learning for android applications	Yavuz Koroglu Alper Sen	2020
S31	Automated Test Selection for Android Apps Based on APK and Activity Classification	L. Ardito, R. Coppola, S. Leonardi, M. Morisio and U. Buy	2020
S32	DroidbotX: Test Case Generation Tool for Android Applications Using Q-Learning	Husam N. Yasin, Siti Hafizah Ab Hamid Raja Jamilah Raja Yusof	2020

Figure 11 – Papers published by year



evaluation results, 2 papers of case studies.

3.3.1 RQ1 - What type of ML techniques have been used to cope with mobile application testing?

The studies were classified by ML techniques described in section II: Supervised Learning, Unsupervised Learning, Semi-supervised learning and Reinforcement Learning. The results in Figure 14, indicate Supervised Learning is the most used ML type present in 22 papers

Figure 12 – Countries

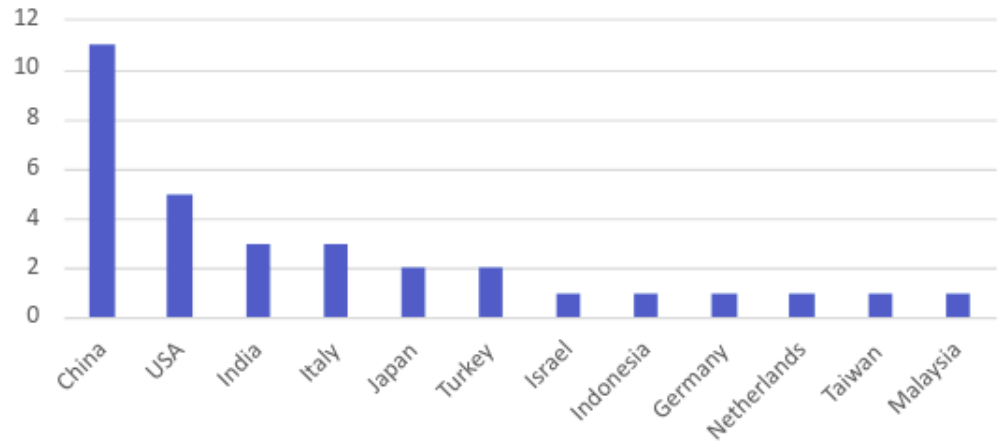
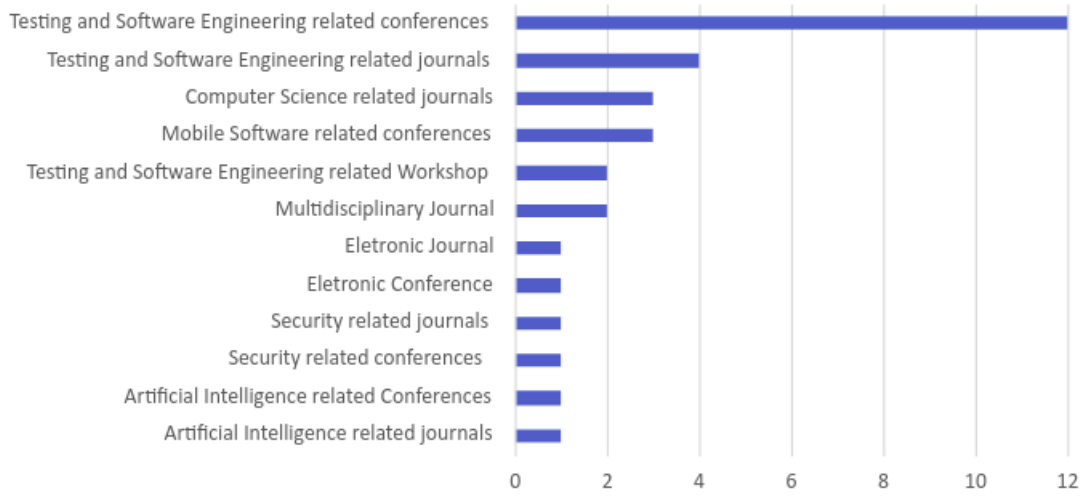


Figure 13 – Papers quantity by Publication Source Type



(68,75%), followed by RL with 8 studies (25%), and UL in 2 studies selected (6,25%). Supervised Learning approaches require labeled data, and in general, there are test tools that generate a lot of data from test analysis, code analysis, app logs, and historical execution. This facilitates the generation of labeled datasets to use Supervised Learning approaches. However, this approach has the drawback of often having too much cost to collect large amounts of data, and process the data to form a quality dataset.

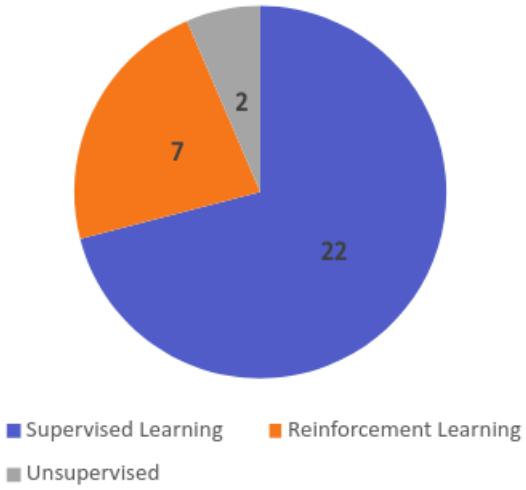
The research question RQ1.1 Which ML Algorithms? Figure 15 shows 21 different ML algorithms. The RL algorithm Q-learning is the most cited, found in 6 studies (S1, S2, S3, S7, S29, and S32), followed by DNN in 5 studies, then Active learning, NB and SVM got 2 citations. All other algorithms used were found in one study each. Q-learning has been explored for testing interfaces. This approach has a Q-function implemented as a table of states and actions (Q-values for each s, a pair are stored there). It uses the Value Iteration algorithm to update the values as the agent accumulates knowledge directly. In testing, the states are the application screen and the actions the steps applied in UI elements. The variations of this algorithm Double

Table 7 – Publication source

Publication source	Indexed by
Testing and Software Engineering related conferences	
IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)	IEEE
Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019	IEEE/ACM
International Conference on Software Maintenance and Evolution(ICSME)	IEEE
ACM SIGPLAN international conference on Object oriented programming systems languages & applications (OOPSLA '13)	ACM
9th International Conference on Software Engineering and Service Science (ICSESS)	IEEE
Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE	IEEE
26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)	ACM
39th International Conference on Software Engineering (ICSE)	IEEE/ACM
IEEE Symposium on Service-Oriented System Engineering (SOSE)	IEEE
42nd Annual Computer Software and Applic. Conference (COMPSAC)	IEEE
IEEE Symposium on Service-Oriented System Engineering	IEEE
ISSTA 2020: Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis	ACM
Testing and Software Engineering related journals	
Information and Software Technology	Science Direct
Applied Soft Computing Journal	Science Direct
Journal of Systems and Software,	Science Direct
Software Testing Verification Reliability	IEEE
Mobile Software related conferences	
IEEE/ACM International Conference on Mobile Software Engineering and Systems	ACM
Artificial Intelligence related conferences	
International Conference on Artificial Intelligence and Robotics and the International Conference on Automation, Control and Robotics Engineering (ICAIR-CACRE '16)	ACM
Artificial Intelligence related journals	
Advances in Intelligent Systems and Computing	Scopus
Computer Science related journals	
Journal of Computer Science and Technology	Scopus
Journal of King Saud University - Computer and Information Sciences	Scopus
IEEE Computer Society	IEEE
Electronic Conference	
Conference: 2017 2nd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2017)	Web of Science

Electronic Journal	
Chinese Journal of Electronics	IEEE
Security Conferences	
IEEE European Symposium on Security and Privacy Workshops	IEEE
Security Journals	
Wireless Personal Communications: An International Journal	Science Direct
Multidisciplinary Journals	
IEEE Access	IEEE
Simetry	Science Direct

Figure 14 – ML Types

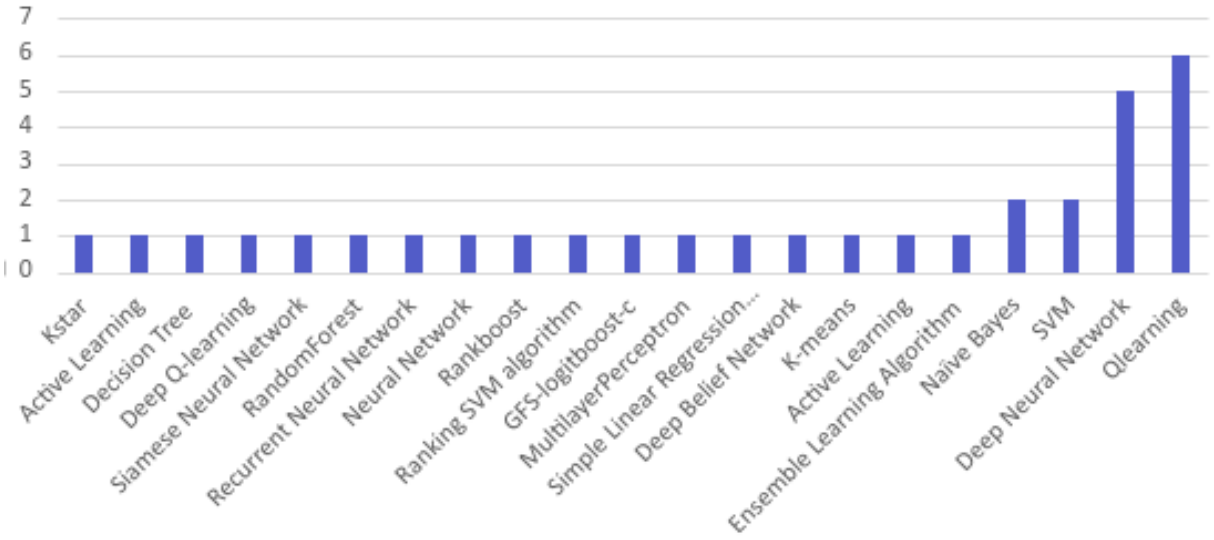


Q-learning and Deep Q-learning, are also present in the studies S10 and S30. They demanded more computational power but presented the advantage to infer large search spaces, useful for large applications. The RL algorithms don't need a dataset, so the learning journey of the algorithm is useful to explore the GUI app finding new states and app failures. Most of the studies focused on the reward policy in identify new states to increase the coverage. Still, no discussion was found regarding the quality of the training phase of this approach to guarantee better performance, and the non-deterministic nature is a limitation to reproduce the results.

DNN is the second most used, present in S4, S12, S27, and S31. This algorithm is used in an image recognition context and intended to predict the class labels of the observed data for pattern analysis and classification. The strategy in S4, S12, and S31 attempt to use image recognition to guide test actions without the dependency of app code. The need for a large-quality dataset is an important point to this approach be well succeed.

The SL algorithms Naïve Bayes, Kstar, Active Learning, Decision tree, Logistic Regression, RandomForest, Rankboost, Ranking SVM, GFS-logitboost-c, MultilayerPerceptron, Simple Linear Regression, Deep Belief Network, Ensemble Learning are present in studies S8, S9, S11, S13, S15, S16, S17, S18, S19, S20, S21, S22, S25, S26, and S28 in different strategies.

Figure 15 – ML Algorithms



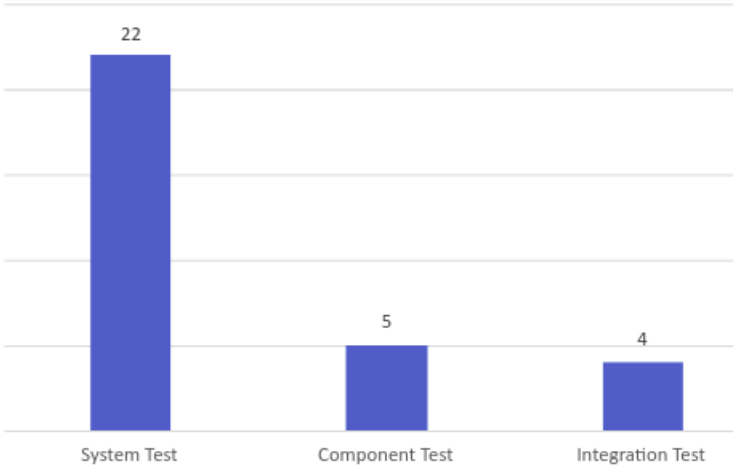
These algorithms have been used learning patterns from a dataset of historical data, or software code metrics and software characteristics main for prediction and classification.

3.3.2 RQ2: Which mobile application testing levels are automated by ML algorithms?

Regarding test levels, Figure 16 shows the system test is the most appearing in 23 studies (71,87%)(S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S24, S25,S28, S29, S30, S31 and S32). Component test is observed in 6 studies (18,75%) (S17,S18, S19, S20, and S21), and integration tests appeared in 4 studies (12,5%)(S22, S23, S26, and S27). Through GUI, system test is costly for mobile apps because of the user input limitation, context sensitive information, variety of device models and versions. So, the research in these conditions is more targeted. The figure also shows which ML type is more applied to Testing Level. We observe SL is the most applied to system test followed by RL and UL in the minor. In the integration test and component test, just SL approaches are applied. The component test studies evaluated SL algorithms for defect prediction using labeled datasets, achieving high accuracy. The integration test studies used SL to verify vulnerabilities through the code in android apps.

Regarding RQ2.1, Which Testing techniques and types? Table 8 shows classification according to ISTQB syllabus 2018 (ISTQB, 2018) as Testing types and sub-types. Most of the studies (62,50%) selected are Functional testing with sub-type GUI Testing. Seven studies are non-functional testing (21,87%), one testing type Compatibility Testing (S24). Six studies regard security testing (S22, S23, S25, S26, S27 e S28), these studies used the approach attack paths, code injection, penetration test, OWASP, and keyword from Abstract Syntax Trees (WYSOPAL *et al.*, 2006) respectively. Three studies (S18, S20, and S21) were classified as structural testing (9,37%); they used basis path testing and the approaches McCabe Object-Oriented, and McCabe

Figure 16 – Testing Level



code metrics (MCCABE. . . ,). Finally, two studies(S17 and S19) were classified as Change-related testing (6,25%). It is a technique designed to verify that a change to the software does not have any adverse effects when code changes (VEENENDAAL; GRAHAM; BLACK, 2008). These studies are in regression testing to predict defects through code change measures (component test level).

Table 8 – Testing types and Sub-types|

Testing Types	Testing Sub-types	Papers
Functional Testing	GUI Testing	S1, S2, S3, S4, S5, S6, S7, S8, S9,S10, S11,S12, S13, S14, S15, S16, S29, S30, S31, S32
Non- Functional Testing	Compatibility Testing	S24
	Security Testing	S25, S27, S28, S22, S23, S26
Structural Testing	Basis Path Testing	S18, S20, S21
Change related Testing	Regression Testing	S17, S19

3.3.3 Which Mobile application Type and Operational System?

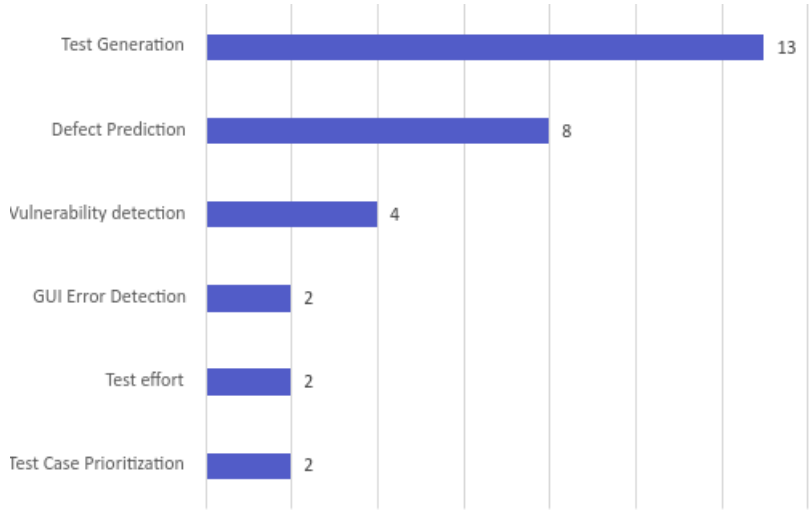
To define this classification, it was considered that the selected studies could be applied to Android, IOS, or hybrid apps. The experiments described were analyzed to arrive at this classification. Most of the studies selected are solutions to the apps for android platform (96,87%), one study, S14, is a solution applied to IOS, and S27 attends both android and IOS. From results, 28 present solutions for native apps(87,5%), four studies (S5, S14, S17, and S27) for hybrid apps. Android platform is open source; this facilitates the access of implementation and research tools.

But hybrid apps are becoming popular (**??**), so test solutions in that context are also needed. The research is challenging due to the development technology of these apps not being native to the operational system, so native tools from the Android platform cannot easily read UI elements or collect logs. They need to be tested and monitored in different operational systems.

3.3.4 RQ3: Which mobile application testing challenges are treated by ML?

The selected studies present researches to solve testing challenges for mobile applications. Figure 17 shows the testing challenges identified as follows. Test Generation appears as the most testing challenge present in the research, followed by Defect Prediction, Vulnerability detection, Test Case Prioritization and selection, GUI Error Detection and Identification, and Test Effort. Figure 18 shows the ML algorithms applied to each test challenge in the studies selected.

Figure 17 – Test Challenges



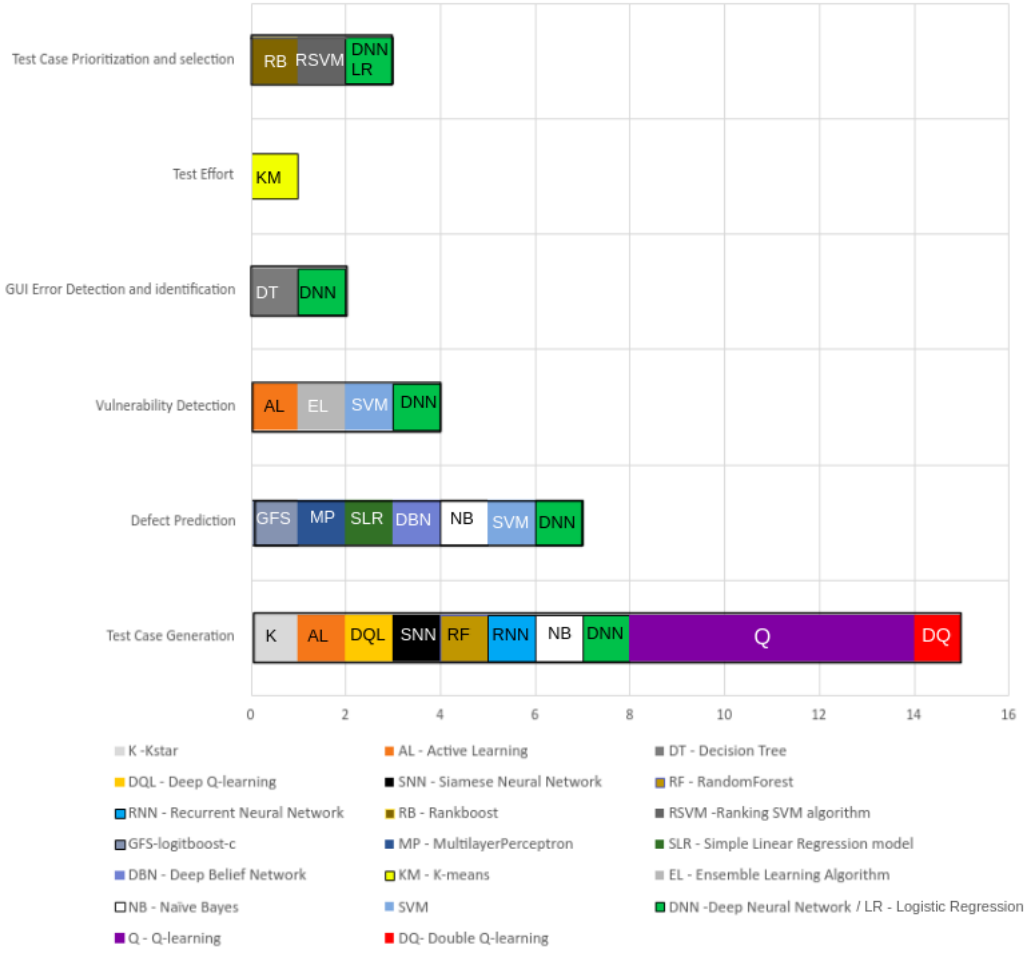
The test design is a labor-intensive task in software testing. It also has a strong impact on the effectiveness and efficiency of testing. For these reasons, test generation has been one of the most active research topics, resulting in many different approaches and tools (ANAND *et al.*, 2013). Regarding this topic, the studies S1, S2, S3, S7, S29, and S32 used the RL algorithm Q-Learning , they achieved better results in code coverage metrics than test case generation tools from literature such as Monkey, Sapienz, Puma, and others; The studies S10 and S30 also got higher code coverage using the variation of RL algorithms Deep Q-learning and Double Q-learning. The Studies S6, S8, S13 used different strategies with SL algorithms kstar, Active Learning, RandomForest, respectively, using a labeled dataset of GUI to guide the tests. S14 used a different strategy applying the unsupervised deep learning approach RNN (recurrent neural network) to create text input to mobile application tests covering use case context, this study performed better than state of art automatic text input tools.

In defect prediction, the ML should learn from a dataset of code metrics to identify the modules that are defect prone and require extensive testing (ARORA; TETARWAL; SAHA, 2015). Mobile apps are often run on different versions of the operating system or different hardware, making the accuracy of the prediction more complex. We found different strategies highlighted from papers that address this topic: The study S17 used the dataset of metric feature selection technique for mobile apps, and the Naive Bayes model achieved the best results outperforming some well-known ensemble techniques. The study S18 used open datasets of static code analysis. This study found that SVM achieved higher accuracy of 83%, performing better than other ML models. The study S19 employs the code change metric to predict the faults instead of just historical data or static analysis metrics, and this strategy was applied verifying GitHub apps repository and achieved good performance with the GFS-logitboost-c algorithm. Study S20 proposes a framework of defect prediction using object-oriented metrics for predicting defective classes. The algorithm Multilayer Perceptron achieved a better score of 0.7 of AUC (area under curve metric). In S21, the authors created a dataset of process metrics to predict defects, Simple Linear Regression model obtained the best prediction than other regression approaches, and the study confirmed that this kind of SL method achieved the best performance using process metric. S22 and S23 used SL algorithms Deep Belief Network and Deep Neural Network to predict security defects in mobile apps. The first one used the strategy to learn the functional and semantic information to predict whether files contain defects. The second one trains the algorithm with extracts both token and semantic features of the defective files in apks to predict them.

A software vulnerability can be seen as a flaw, weakness, or even an error in the system that an attacker can exploit to alter the system's normal behavior (JIMENEZ; MAMMAR; CAVALLI, 2009). Detection of vulnerabilities in mobile apps is important to assure reliability and avoid security failures, such as invasion of apps, attack paths, and code injections. In study 25, this topic is addressed using active learning (**??**) in combination with algorithms that discover attack paths in the learned state machine. The detection techniques attempt to discover a path that exploits a given class of vulnerabilities specified in the OWASP Top10. If such a malicious path exists in the inferred model, the application must contain a specific vulnerability. In study 26, to detect the vulnerability, the static analysis is used to determine the location information of the loading point. It extracts the feature vector for each loading procedure and the classification of the extracted feature vector through constructed multilabel classification ensemble learning algorithm. The results obtained that the detection method can detect vulnerabilities of dynamic loading effectively and is more comprehensive. Study 27, construct a novel deep learning network, Hybrid Deep Learning Network (HDLN), and use it to detect these attacks. This study extracts more features from the Abstract Syntax Tree (AST) of JavaScript. It employs three methods to select key features and train the DL to distinguish vulnerable applications from normal ones achieving 97.55% accuracy. In S28, the SVM algorithm is used to identify vulnerabilities in Android apps that allow penetration invasion. A sample set of apps under test was created after

summarizing defect collection, and an attacking API sequence was extracted, achieving above 81% of accuracy applied in three different kernel functions.

Figure 18 – ML algorithm by Testing Challenge



Test case prioritization is useful to reduce the cost of regression testing. Testers may prioritize their test cases so that those which are more important, by some measure, are run earlier in the testing process (ELBAUM *et al.*, 2004). For mobile apps, the prioritization of test cases that have the likelihood to reveal important failures in the early development stage is important due to the various testing types to be executed in this context. Study S15 proposes a learning framework based on the RankBoost learning-to-rank approach to facilitate the verification task by learning to prioritize GUI test cases. The results show that this study can effectively rank the test cases if there are enough pairwise priority relations in the training test cases. S16, proposes a novel approach to prioritize test cases according to their likelihood to cause memory leaks in android apps using Ranking SVM algorithm, It firstly builds a prediction model to determine whether each test can potentially lead to memory leaks. Then, for each input test case, it partly run it to get its code features and predict its likelihood to cause leaks prioritizing to run first. Study S31 presents a novel testing framework that allows the tester to write scripts, creates a model of the app's GUI, identifying activities, an app classifier that determines the type of apps,

and activity classifier, and a test adapter that executes test scripts that are compatible with the specific app. Using a DNN for App classification and LR for activity classification, this study adapted high-level test cases to 28 out of 32 apps. It reduced the LOCs of the test scripts by around 90%.

In GUI error detection, ML is applied to examine some image details that are hard to identify by testing tools, such as wrong icon position, field format breaks, and other unpredictable interface issues that make these apps more likely to have interface problems. Study S9 proposed an automatic detection platform for GUI errors of mobile apps related and image-related widget error classification model through decision tree algorithm, which detects the error of widgets, achieving f 98.06% for text-related and 95.44% for image related. In S12, used a DNN end-to-end trainable model to determine the similarity between GUIs and identify isomorphic GUIs in robotic testing of mobile apps. In this study, The camera captures GUI information and monitors test execution, and the robotic arm interacts with mobile devices like the test performer. This method achieved 97.6% of accuracy, demonstrating effectiveness.

Test effort is related to testing costs. In S26, a statics method is applied to cluster mobile devices with similar configurations and appliances to avoid redundant testing reducing the test cost in compatibility testing using an unsupervised learning k-means algorithm to generate an optimized compatibility test sequence. The results obtained showed potential application and effectiveness.

3.3.5 What advantages and limitations are found on ML based mobile application testing?

Mobile app testing is specially challenged in the software testing area. Finding good tools and good coverage of test scripts are considered difficult to achieve due to mobile app limitations as described in chapter 2. Different ML types have been studied to fit human interaction simulation, defect prediction, and classification. Researches using SL algorithms have been present in various app testing challenges such as test generation, test case prioritization, defect prediction, and GUI error detection. The SL algorithms NB, SVM, Decision Tree, LR, GFS-logit boost-c, Multilayer Perceptron, and Simple Linear Regression are applied successfully through a dataset of app code metrics, recognizing the app patterns. The limitation of the presented research is the dependence on a large dataset of code metrics. SL also showed promising results with DNN for app GUI error identification. It can search, correlate and combine data to get faster learning. However, imperfections in the training dataset make them vulnerable. The known limitation of this approach is the need for a large amount of annotated data. Depending on the display, image quality, luminosity conditions, or overlap of screens, the detection can fail.

Reinforcement Learning, specifically Q-learning algorithm is the most used in test case generation for GUI testing. In S3, the author highlighted the Q-Learning algorithm has the

capability of finding the optimal way to reach a certain state of the environment, it is used to guide the exploration to reach states that reveal application's functionalities or hard-to-reach states. The empirical evaluation of studies S1, S2, S3, S7, S10 and S32 showed q-learning has the advantage to explore mobile applications and generate more test scenarios based on GUI state, improved the coverage and found more crashes than other approaches. The research evaluations presented as limitation the lack of the dataset to probability distribution since different apps may require quite different sequences of actions to satisfy the same objective. Researches in SL have been widely used in various mobile app testing challenges such as test case generation, test case prioritization, test input generation, defect prediction, GUI error detection and identification. Defect Prediction in unit testing level is the most applied SL algorithm based on test metrics or code metrics dataset. The limitation of the presented research is the dependence of a large dataset of history mobile app data with metrics used for prediction.

Some deep learning approaches presented in S4, S12 and S31 showed promising results for application GUI identification for input testing generation, GUI error diagnosis, UI components detection and GUI isomorphic relation. Deep learning can search, correlate and combine data in order to get faster learning, however, imperfections in the training dataset of DNN make them vulnerable. The known limitation of this approach is the need of a large amount of annotated data. Depending on the display, image quality, illuminosity conditions or overlap of screens, the detection can fail.

The availability of larger mobile app dataset, history data, quality of data provided (treatment and preprocessing) are the barriers highlighted by the most studies observed. It is important to note that it can be difficult for the adoption of ML in mobile app testing in the industry. The need to invest in a good computational resource for processing the ML algorithms and the investment in training and allocation of test professionals to work with the treatment and creation of data and in understanding the algorithms and your results returned.

We also observed that most of the studies only focused on android native apps, limiting the employment in hybrid apps tests. The barriers observed in the studies are highlighted large datasets, quality of data, and hardware structure. To apply in the industry requires investment in hardware resources and qualified professionals.

3.3.6 Threats to Validity

This mapping considered the main studies of ML and mobile application testing, to guarantee the corrected analysis of the main concepts of these two research areas, the results classification obtained were reviewed according to the authors consensus in the technical literature and internally in order to avoid the Researcher bias.

In order to capture relevant results to answer the RQs, the research was conducted using the main databases: IEEE, ACM, Scopus, Web of Science and Science Direct. One threat to

validity is related to the possibility of several relevant studies missing during the conduction of the automated search in these databases because of different types of search in each database. To avoid it, the search using the string was performed in the ACM library join search using the string to capture by Title, Abstract and Keywords. In Science direct, due to limitations of the number of keywords and operations to perform the automated search, the string were adapted and a join was performed observing the results returned when run the main keywords (taken from RQs) of each area. Another action taken to avoid this risk of missing relevant studies was to conduct searches on Google Scholar and perform the snowballing to dig for relevant papers that might not be returned on the date of the search.

A difficulty found during results analyses is certain papers did not describe in detail the ML algorithm and the type of dataset used and the processing of data, to achieve this information, the references were consulted to obtain a clue to apply the correct classification. Another barrier was to find the reference in the papers of mobile applications type, most of the studies just mention the operational system. Finally, in few studies were found a section describing the difficulties or concerns about the ML techniques applied in mobile application testing.

One Internal validity identified is related to the possibility of several relevant studies missing during the conduction of the automated search in these databases To avoid it was to perform snowballing to dig for relevant papers that might not be returned on the date of the search. The external validity found was that certain papers did not describe the ML algorithm, dataset information missing, and few studies describing the difficulties or concerns about the ML-based mobile app testing. The mitigation applied was by consulting the references to obtain a clue to apply the correct information. The conclusion validity during the study is to find different definitions of test types and test techniques in the literature. Therefore, the definition by ISTQB was used.

3.4 Final Considerations

This SM pointed that mobile app testing and ML have increasing attention over the last few years. Concerning our research questions, SL is the most ML type observed in the studies. It has successfully performed defect prediction using static code data. Moreover, these studies presented the dependence of the large dataset to perform the prediction efficiently. This same challenge is observed in vulnerability detection and GUI error detection. Q-learning is the most cited algorithm to solving test generation and exploration through GUI apps regarding the ML algorithms. The challenge of this approach is to create a sequence of actions of test cases automatically and create the test oracle to cover the app's functionality. Due to the high cost, test generation through the app' GUI at the system test level is the most researched in the studies.

The advantages and limitations presented by the study (RQ5) highlighted that ML algorithms could improve performance over time but depends on storage and computational capability.

Concerns in the industry involve few studies applied to hybrid apps, resource investment for hardware, and qualified professionals to create and maintain these systems. However, the creation of tools to support the mobile app challenges reported is in the initial stages of maturity. The SM results can be useful to researchers to obtain information to identify research directions for performing a rigorous evaluation of their work also in industry. The future of this work will identify ML-based app testing solutions in real-world companies.

CHAPTER

4

PROPOSAL APPROACH DEEPRLGUIMAT

4.1 Initial Considerations

The advances in mobile computing and the market demand for new products that meet an increasingly public represent the importance of ensuring the quality of mobile applications. In this context, automated GUI testing has become highlighted in the research. However, studies indicate that there are still limitations to achieving many possible combinations of operations, transitions, functionality coverage, the bottleneck in manual test design, and fault reproduction (KONG *et al.*, 2019).

Several techniques are applied to overcome these limitations, such as Random, which produces pseudo-random events to test. However, it has no guidance may get stuck when dealing with complex transitions. Model-Based it extracts test cases from navigation models built by static or dynamic analysis. Their performance is limited when the amount of space and actions increase. Search-based using evolutionary algorithms as Genetic Algorithms has shown excellent results since a specific coverage target guides them. However, they do not take advantage of past exploration successes to learn the most compelling exploration strategy dynamically, it needs computational power, and they do not scale well with the complexity (Kong *et al.*, 2019).

Machine Learning techniques are popular in many commercial applications, and it is now accessible technology. However, supervised approaches need a large amount of data to get learning. RL is a technique that the agent learns through environment interaction and trial-error in this context. It does not need a dataset. According to the literature, the RL algorithm Q-learning is the most used to generate test cases for GUI testing. It learns the value of an action in a particular state. The "Q" refers to the algorithm's function computes the expected rewards for an action taken in a given state. This approach has the limitation of applying only to discrete action and small state spaces, being inefficient in a large amount of action-state combinations. Another difficulty is that the agent Q-learning cannot estimate the value to other states (PASZKE, 2017a).

So, the current Q-learning approaches do not consider significant variations of input test data.

The RL algorithm DQN uses a deep neural network instead of a Q-table to approximate values to overcome the Q-learning limitation. The mechanism of Experience Replay accelerates learning. The agent DQN learns leading with deferred rewards learning to link to actions that cause them. So, DQN is suitable in this research focused not only on app exploration and finding crashes but on producing useful test cases finding failures exercising the test input variation.

The test input variation in software testing is important for efficiently finding faults, as exhaustive testing is impractical (ISTQB, 2018). In this sense, testing techniques and criteria help select test inputs, as explained in chapter 2, covering a range of input values and may reveal failures that violate input requirements. Systematic Functional Testing (SFT) combines the main input criteria (equivalence partition and Boundary value analysis), is successful in functional and structural tests and has good results in mutation testing (LINKMAN; VINCENZI; MALDONADO, 2003). Therefore, this work is based on SFT guides, seeking the agent to perform combinations that can be efficient to reveal functional failures. It is a novel aspect for test generation tools.

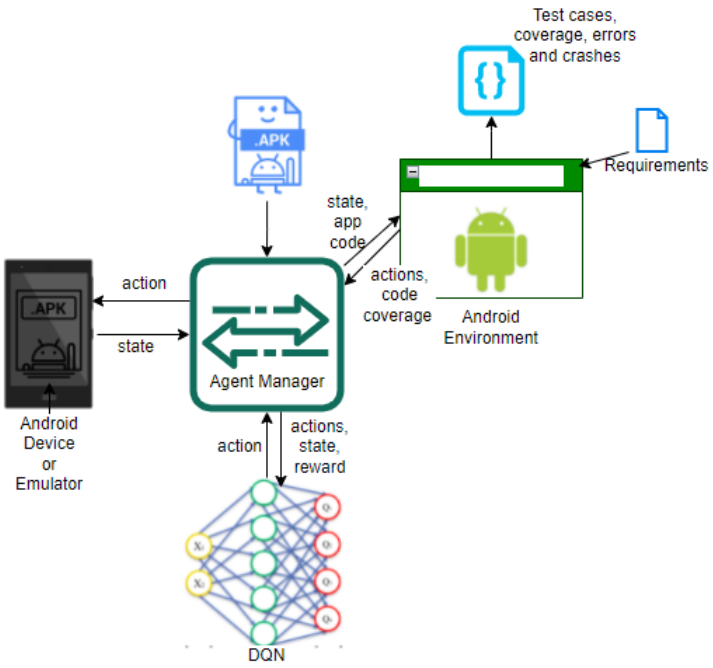
This chapter describes the approach DeepRLGUIMAT to test case generation for mobile apps and the implementation of the tool applied to Android for proof of concept. Section 4.2 shows the approach modules details Environment, DQN and Agent Manager. The implementation environment and Operational use details are described in 4.3,4.4 and 4.5.

4.2 DeepRLGUIMAT

Figure 19 represents the workflow of the approach DeepRLGUIMAT.It contains three modules: Agent Manager, DQN and the Environment. Agent Manager is the RL agent which interacts with the RL environment (Android Environment) and sends the information of actions, rewards and states to module DQN. The environment is the world through which the agent moves and responds to the agent, in our case the representation of Android app. The environment takes the agent's current state and action as input and returns its reward and its next state as output.

The DeepRLGUIMAT input is the mobile app (android apk) only. The module agent manager calls the environment to install and launch the application on an emulator or device. It reads the state of app GUI and generates a vector of GUI possible input actions. Agent manager calculates the reward and sends them to DQN, which processes state, actions, and reward value to select the best action back to the agent manager to be addressed to the device. The agent can perform the tests in two ways: without entry requirements, the agent explores the app using general actions with default input types; and with entry requirements to guide the app exploration.

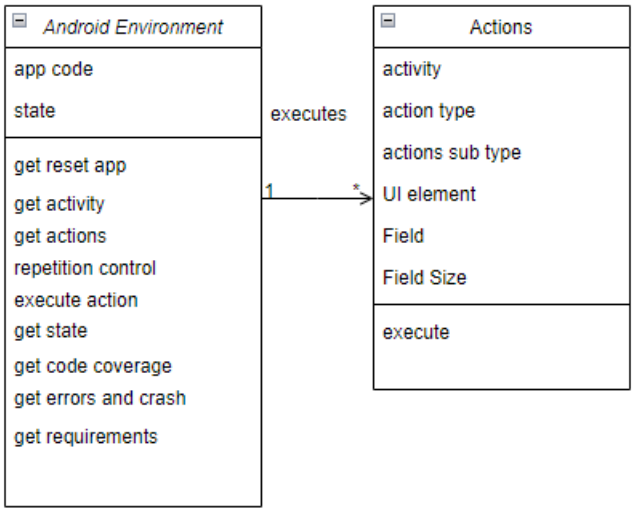
Figure 19 – DeepRLGUIMAT Workflow



4.2.1 Environment Module

There are several RL environments available to try RL algorithms in OpenAI gym (GYM, 2020). The Environment in DeepRLGUIMAT allows to connect to an Android device or emulator, install and launch the app, send actions and collect the state (app screen), the activity, the code coverage, and the test results. Figure 20 shows the Android Environment attributes and operations.

Figure 20 – Android Environment classes

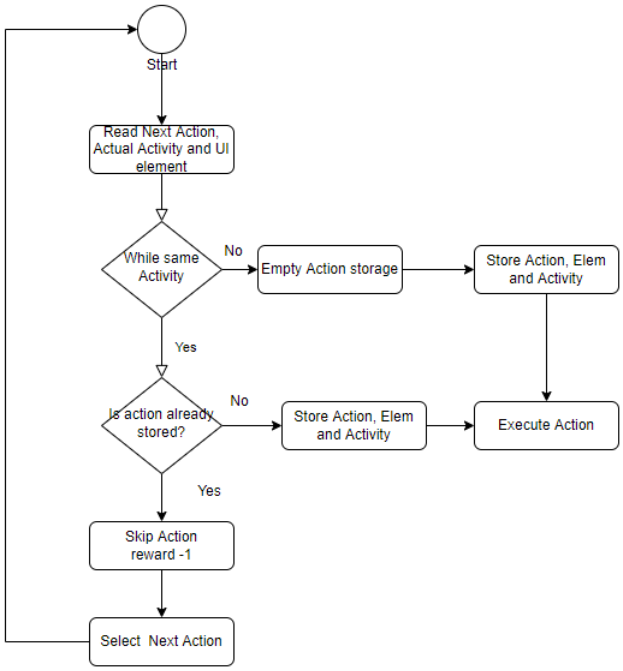


The operation *get reset* app uses commands to install and launch the application under

test in a target device or emulator; *get activity* returns the name of the app screen launched. In *get actions* all GUI elements of the screen are read, and a vector of possible actions for the GUI elements is created and returned to the Agent manager module.

The *repetition control* (Figure 21) is a mechanism to prevent the tool from getting stuck in an action execution bias. As training is carried out, if there is no variety of actions for the agent to learn, biases characterized by repetition of actions on the same screen may occur. It regulates the actions performed at the same screen. It does not prevent from repeating actions, it can repeat, but in a limited way in the same screen. While the action is in the screen, this mechanism encourages attributing a smaller reward to return the actions not yet performed in the screen context (all new actions performed are stored in a vector) until the agent leaves the screen, the Action stored vector is empty and the actions of other state are stored and verified again.

Figure 21 – Repetition Control



In *get state*, it takes the screenshot of the device and store in a states directory. The operation *get code coverage* is a feature that collects the code coverage. The function *get errors* and *get crash* monitor the app, identifying if an action provokes an error or a crash. An app crash is an unexpected exit caused by an unhandled exception. The method execute actions, has as a parameter the action in *Actions class*; this function sends the screen name, the code coverage after the action performed if the action caused the failure or crashes. In *get requirements*, the requirements file is read to store the information to be guide the exploration.

The *Action Class* has the attribute action types, and its can be GUI app: Click, Scroll, Set Text, Select, Long Click, Check and Android device actions: Home, Back, Volume, Rotate and Android menu. Table 9 shows the GUI actions in the first column, the action subtypes in the second column, and from columns 3 to 7, the parameters. This set of standard actions were

inspired by the SFT guidelines for numbers and text, when there are no system requirements, we consider the default value limits generated through a random function were adopted with sizes ranging from 1 and 2 for small text, 10 to 20 for medium text and 100 to 200 for large text. The same for numbers, symbols and mixed (mixed text and number values) and we applied the SFT guidelines to offer actions that provide tests with extreme values (small text, small number, large text, large number), acceptable values (medium text, medium number), special cases and illegal cases (empty, symbols, mixed) . In case of field size requirements provided, then inputs are generated to cover them, threshold values and disallowed values.

Table 9 – GUI Actions

GUI Actions	**SubActions**	**Param 1**	**Param 2**	**Param 3**	**Param 4**	**Param 5**
Click	-	x	y	-	-	-
LongClick	center	x	y	-	-	-
	top left	x	y	tl	-	-
	botom	x	y	br	-	-
Text	text medium	x	y	string	(size 10 to 20)	-
	text large	x	y	string	(size 100 to 200)	-
	text small	x	y	string	(size 1 to 2)	-
	number medium	x	y	number	(size 10 to 20)	-
	number large	x	y	number	(size 100 to 200)	-
	number small	x	y	number	(size 1 to 2)	-
	symbols	x	y	symbols	(size 10 to 20)	-
	mixed	x	y	string, and number	(size 10 to 20)	-
Scroll	swipe down	x1	x2	y1	y2	-
	swipe up	x1	x2	y1	y2	length
	swipe right	x1	x2	y1	y2	-
	swipe left	x1	x2	y1	y2	length
Device Actions	**Parameter**					
Back	-					
Home	-					
Menu	Android menu itens					
Volume	up / down					
Rotate	landscape / portrait					

The action long click has 3 variations center, top left, and bottom. These variations facilitate to access long-click menu in mobile apps. The Text action has 8 sub-types, and they vary according to the data types letter, number and symbols, and size (column Param 4 the size

in character number). The action sub-types of Scroll are Up, Down, Right, and Left; the duration value default is 10 steps. The device actions include Back, Home, Menu (random click on menu items), Volume (up and down), Rotate (landscape and portrait).

The test events generation is a useful feature since the manual input of automated test scripts is a bottleneck in the development process. DeepRLGUIMAT registers the actions performed in text files called test events to debug a scenario. The file contains the screen name, the sequence of actions performed red underlined in the picture (click, type, check, etc.), input value, the location of GUI element (bounds), element name, the screenshot of the device, and if it has an error.

Figure 22 – Test Case

```
1 Test Event com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity
2
3 typed in com.blogspot.e_kanivets.moneytracker:id/etCategory value: YUhMmRuobJLpiBiRNIFFpCASRnTddCyaxDdUCJMDCdZLxKSaOCP
4
5 Expected max size 50 Actual value: YUbMmRuobJLpiBiRNIFFpCASRnTddCyaxDdUCJMDCdZLxKSaOCP  Screen: state_20210614-111731.png
6
7 clicked android.widget.ImageButton com.blogspot.e_kanivets.moneytracker:id/fabDone bounds:[594,1310][692,1381] Screen: state
8
9 typed in com.blogspot.e_kanivets.moneytracker:id/etTitle value: RfgWtKQtZdZGHWJhPcAsiryJSWHrjsXKhxAvlPjQWnvHsnjDDen
10
11 Expected max size 50 Actual value: RfgWtKQtZdZGHWJhPcAsiryJSWHrjsXKhxAvlPjQWnvHsnjDDen  Screen: state_20210614-111750.png
12
13 typed in com.blogspot.e_kanivets.moneytracker:id/etPrice value: 3  Screen: state_20210614-111756.png
14
15 clicked  android.widget.ImageButton (Navegar para cima) bounds:[0,55][98,153]  Screen: state_20210614-111803.png
16
17 Go to next activity: com.blogspot.e_kanivets.moneytracker/.activity.record.MainActivity
18
19 Expected activity: com.blogspot.e_kanivets.moneytracker/.activity.record.MainActivity
20
21 Test Passed
```

The Figure 22 shows a test event file generated by DeepRLGUIMAT. This file is formed in the first line by the name of the Activity where the events occurred. In the next lines, we see the actions performed on which type of UI element (button, text, image button, ...) the identifier of the element if it exists (resource id or element text), the entered value (in the case of a text), the location of the action (bounds) and the screenshot of the screen. In the figure, we highlight the actions underlined in red, and in blue, we underline the expected result information. In the circle, we highlight the text of an element used for its identification. Finally, we have if the test result is following the requirements, Test Passed. Otherwise, the result would be Test Failed.

4.2.1.1 Actions

In the DeepRLGUIMAT tool, the generation of actions vector is done by extracting elements from the app's interface to derive the sub-types of action (Table 9). In case there are requirements, these are used to generate more actions according to the field specification, using the systematic functional testing technique for test inputs to cover the requirements. Having the requirements is to explore the application more efficiently and generate test events with defined expected results.

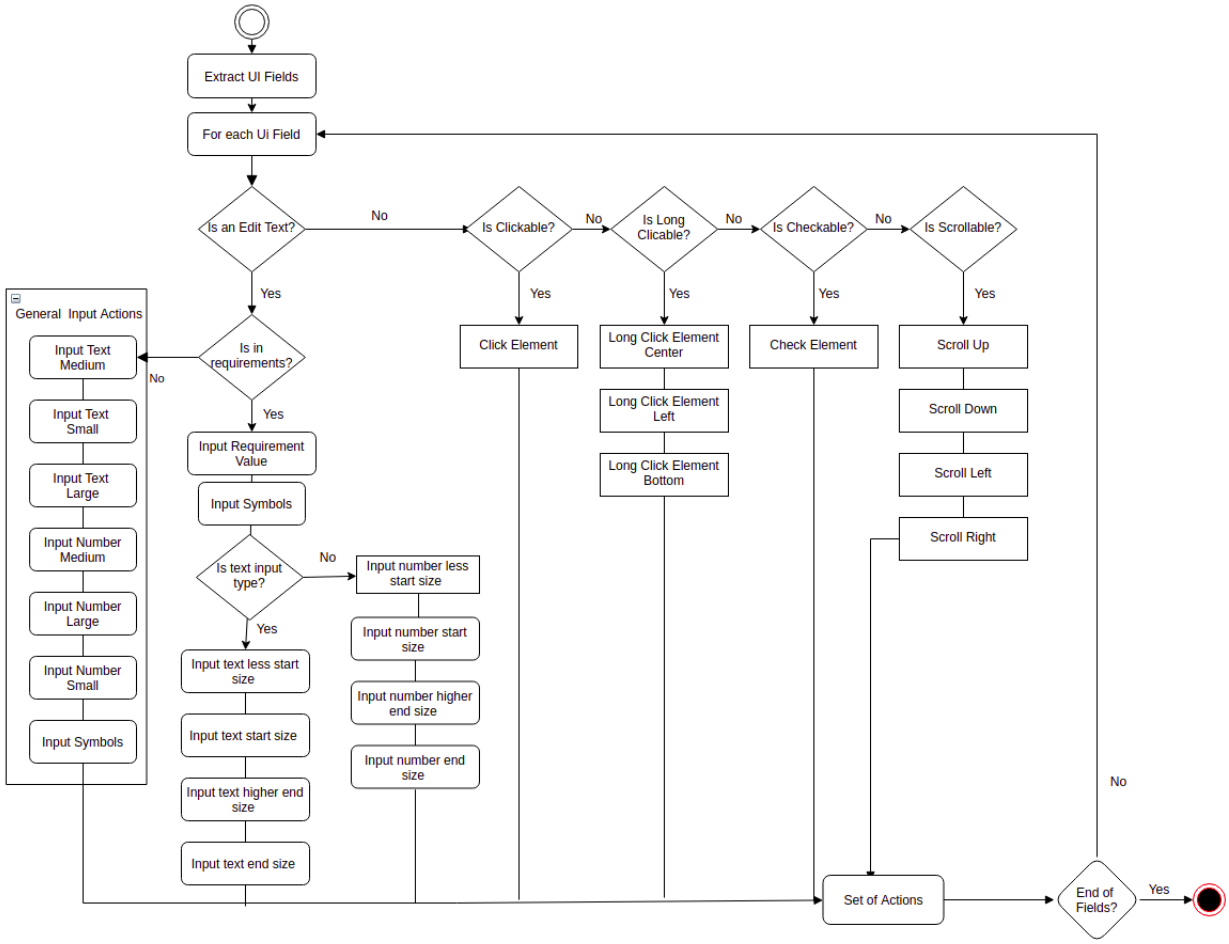
Figure 23 shows the requirements information that is needed for the tool: in the first column is the current activity information, in the second column the field type information

Figure 23 – Requirements

activity	field	id	action	type	size_start	size_end	value	Result_activity	Result_elem	Result_text
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	t.e_kanivets.moneytracker:id/etPrice	type	number	1	13	1500		e_kanivets.moneytracker:id/etPrice	1500
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	t.e_kanivets.moneytracker:id/etTitle	type	text	1	50	test		e_kanivets.moneytracker:id/etTitle	test
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	.moneytracker:id/etCategory	type	text	1	50	bill		e_kanivets.moneytracker:id/etCategory	bill
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	button	t.e_kanivets.moneytracker:id/fabDone	click	none				_kanivets.moneytracker/.activity.record.MainActivity	com.blogspot.e_kanivets.moneytracker:id/tvTitle	test

(edit text, button, image button, select, scroll, etc.), the third column is the id of the element (resourceid). In the fourth column, the action to be performed (type, click, long click, check). In the fifth column, the type of value input (text), number, or text, or symbols, and none if it has no input. The `size_start` and `size_end` columns are for specifying the minimum size limit for value input and the maximum size limit. This data is important for generating test entries. The value column is to specify an input value. Finally, the `Result_activity`, `Result_elem`, `Result_text` columns represent the values to be checked in the final result to assess whether the test passed or failed.

Figure 24 – Actions Generation Flow



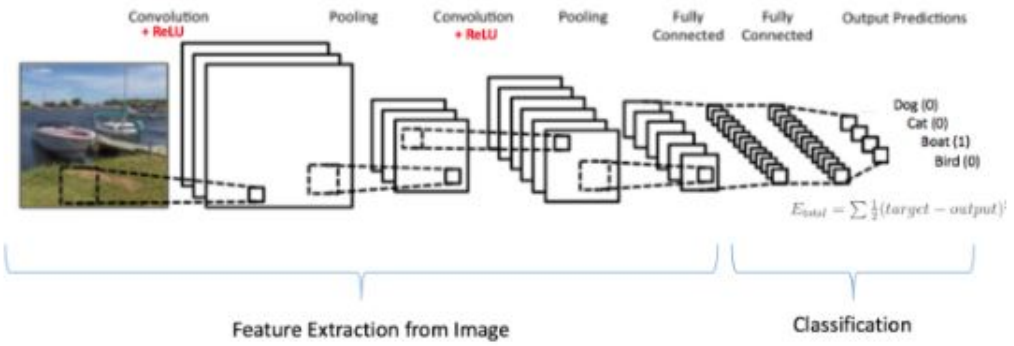
To better understanding, Figure 24 illustrates the flow for generating the actions vector. First, the method *get actions* extracts the UI fields from the actual activity and stores them in a list. For each field in the list, the element type is verified. If the field is clickable, the action "click" is added to a set of actions, same for "long clickable", "checkable," and "scrollable". For edit texts fields, if it has no requirements, the general actions for input texts are added to a set of actions (Table 9 shows GUI Actions). If It has requirements information, test inputs are derived according to the requirements, such as input value less than the limit (less start size), input value in the inferior limit (input value start size), input value higher than the superior limit (input value higher end size) and so on. The set of actions are returned in a vector to be sent to the DQN module.

The purpose of this feature is to enable the test generation that cover technical requirements for data entry in order to have automatic tests that check the basic conditions, not only navigation, ensuring that the application does not break when performing data entry different from the requirements. With this, the tester will be able to focus efforts on testing business rules that add value to the product.

4.2.2 Deep Q-Network Module

The module Deep Q-Network (DQN) has the goal to select actions that maximize cumulative future reward. In this sense, the convolutional neural network (CNN) is used to approximate the optimal action-value function. CNNs are a type of deep feedforward neural network specialized in image visualization and processing. A classical neural network has a structure divided into layers formed by neurons, with only the hidden ones working differently. The detection of characteristics of a CNN is done in the convolutional stage by convolutional filters, also called kernel, which are arrays much smaller than the input, but with the same number. Of dimensions, which in many cases is 3: height, width, and color channels, Figure 25 represents how CNN works (TAMADA,).

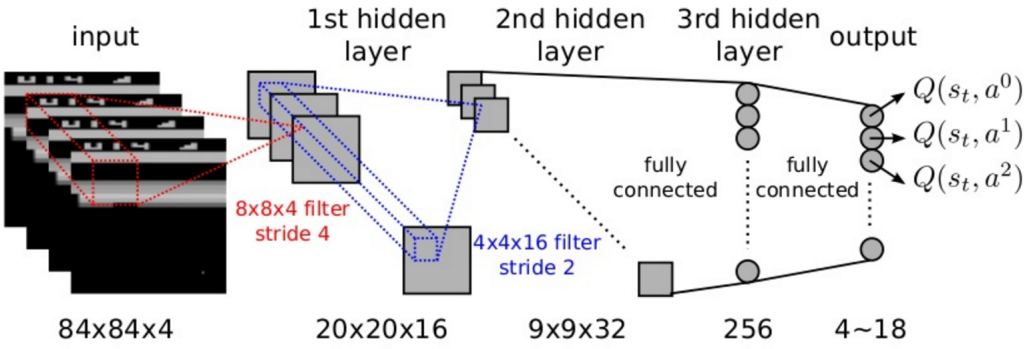
Figure 25 – CNN, (ALI, 2019)



In DQN, the images are the input, processing takes place in the hidden layers, and in the output layer, each neuron has a value that determines the action taken. However, as the past

images are not labeled, the evaluation of the network output is done in another way, through reinforcement learning. In other words, while the characteristics of the environment are defined by the network, the calculation of the error for the optimization of weights is done by the RL (TAMADA,).Figure 26 represents the DQN architecture.

Figure 26 – DQN architecture, (YOON, 2019)



When the transition probabilities are not known, so the Q function policy recommended is the Bellman's equation to obtain the value function from observations made about the environment (DANA; VAN, 1990):

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

This module has the optimization function that performs a single step of the improvement. It first samples a batch, concatenates all the tensors into a single one, computes $Q(s_t, a_t)$ and $V(s_{t+1}) = max_a Q(s_{t+1}, a)$, and combines them into our loss. By definition we set $V(s_t) = 0$ if s as a terminal state. To minimize the errors in the optimization function, Huber loss, which works like the mean squared error when the error is small, but as the mean absolute error when the error is large (PASZKE, 2017a).

$$\mathscr{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathscr{L}(\delta)$$

$$\text{where} \quad \mathscr{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{4.1}$$

This function is quadratic for small values of a, and linear for large values, with equal values and slopes of the different sections at the two points where $|a| = \delta |a| = \delta$. The variable refers to the difference between the observed and predicted values $a = y - f(x) a = y - f(x)$. Figure 23 shows the DQN module data flow. The Agent module sends action, state, and reward to be recorded in the replay memory and also runs optimization steps on every iteration. Optimization picks a random batch from the replay memory to do training of the new policy. The target net is also used in optimization to compute the expected Q values; it is updated occasionally to keep it current (PASZKE, 2017a).

The Reward is an important input to DQN. A reward is a function that provides a numerical score based on the state of the environment. The goal, in general, is to solve a given

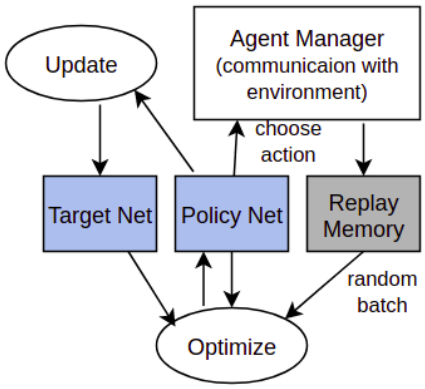


Figure 27 – DQN data frow

task with the maximum reward possible. It is through this that the agent will select the best action. Function 4.2 shows the reward rules.

$$r = \begin{cases} \Gamma & \text{if } s \notin S_{sv} \\ \Gamma & \text{if a} \in R_a \\ -\Gamma & \text{if a is repeated} \\ -\Gamma & \text{if } s \neq \text{AUT or Crash} \\ 0 & \text{other cases} \end{cases} \tag{4.2}$$

The reward is high (Γ) if the state(s) never observed before in the set of visited states (S_{sv}) and the action is in requirements. The reward is lower ($-\Gamma$) if the same action is repeated or the state is out of AUT (app under test), and when the app crashes, the reward is lower too. In all other cases, the reward value is 0.

Generally, actions that cause crashes close the application forcing to restart the application, and this step finishes the episode. The failures that cause crashes are important, but to prevent the agent bias from returning to the same crash action, the decrease in the reward manages to avoid the repetition of the same step several times in a new episode.

4.2.3 Agent Manager Module

Agent manager module generates tests through a selection of sequences of actions and states that maximizes cumulative reward. Algorithm 1 shows pseudocode for the proposed DQN based test generation algorithm. It takes as input the AUT, the end criteria (c) is the time to run the tool. Line 3 is the first loop while the episode is not completed, the AUT is started in line 4 (emulator or Device connected by USB), the android environment returns the actual state and actions for the application launched on line 5. In line 6, the vector activities are initialized; this vector stores the activities of the application accessed.

The strategy of this algorithm is to explore the entire application giving high reward to actions which discovery new activities and comply entry requirements (lines 15 to 18) to prioritize the selection of different actions to avoid repetition. However, actions performed outside the application, recently repeated actions and actions that cause crashes receive a decrease in reward (lines 10,11 and 13,14). Memory replay (DQN module) stores the state, action, next state, and reward in line 19. In line 22, the Optimization function (DQN module) described in Section 4.2 is called. If the terminal state criteria are satisfied in line 23, the episode is finished.

Algorithm 1 – Agent manager module

```
1  initialization episode duration
2  c = end criteria
3  while episode (i) not completed do
4      start AUT
5      state, actions = AUT environment
6      activities = []
7      steps =0
8      while True do
9          action = select action(state, actions)
10         if action was already performed in same state then
11             reward = small reward
           end
12         next state, actions, crash, activity = execute actions(action)
13         if activity is not AUT or App crashed then
14             reward = small reward
           end
15         if action comply requirements then
16             reward = large reward
           end
17         if activity is new then
18             reward = large reward
           end
19         Store in DQN memory replay (state, action, next state, reward)
20         state = next state
21         steps = steps+1
22         DQN Optimization model
23         if c or crash then
24             next state = 0
25             Epoch complete in steps
26             breaks
           end
       end
   end
```

The execute action function in line 12 will send the action to Android Environment. It will perform the action, returning the State resulted (next state), the set of possible actions

(actions vector generated), name of the activity and a boolean value for crash (True or False). In the Android Environment, the execution is monitored and the tests performed, errors in Logcat, and application screens are registered.

Every episode lasts until covering actons vector, or the action is out of the app or if the app crashes. Once an episode comes to an end, the app is restarted, and the tool uses the acquired knowledge to explore the app in the next episode. The algorithm's journey encourages the discovery of happy path scenarios, alternative paths, input variation and discovery of failures in the application.

4.2.4 Studies Comparison

Through the systematic mapping described in Chapter 3, we identified 13 studies referring to methods and tools similar to the solution proposed in this work, in the Table 10 we indicate these works in the column papers, the other columns refer to the characteristics that these tools have in comparison to the tool proposed in this work.

The column System Req. Entry means the tool has the feature of accepting requirements as input. The Test Case Output column indicates whether the tool generates test cases files as output. The column Test Input Variation refers to whether the tool performs different input types. The Test Report column indicates whether the tool generates a report or file of final results and failures found. The Tool column is for knowing if the job is a complete tool or complement another existing tool. The Available for download column reveals whether the tool is published and available for download. The Android 9+ column indicates whether the work can be performed on the latest Android 9+, which make up most of the current market. And lastly, column Results Verification means the tool verify the expected results. The answers N means No, Y is yes, and P is partial.

In comparison with other tools, DeepRLGUIMAT (DG) has functionalities that are useful for app test execution, since to generate tests that can be used it is necessary that they be registered to be reproduced and, in case of failure, debug. Another important point is the possibility of having as input requirements for data entry into the system, it allows for the tool to exercise the happy path and the alternative paths.The results verification is partially attended, since the tool just can verify what is specified in input requirements.

4.3 Implementation

The tool DeepRLGUIMAT was developed using Python language due to the wide variety of libraries for reading and writing images, CSV and XML files, and scientific libraries such as:

- Pytorch: It is a library for deep learning on irregular input data such as graphs, point clouds, and manifolds (PYTORCH, 2021).

Papers	System Req. Entry	Test Case Output	Input Variation	Test Report	Tool	Available for download	Android 9+	Results Verification
S1	N	N	N	N	Y	N	N	N
S2	N	N	N	N	Y	N	N	N
S3	N	N	N	N	Y	N	N	N
S4	N	Y	Y	Y	N	Y	Y	N
S5	N	N	N	N	Y	N	N	N
S7	N	N	N	N	Y	Y	N	N
S8	N	N	N	N	Y	N	N	N
S10	N	N	N	N	Y	N	N	N
S11	Y	N	N	N	Y	N	N	N
S13	N	N	N	N	Y	N	N	N
S29	N	Y	N	N	Y	Y	Y	P
S30	Y	Y	N	N	Y	N	N	Y
S32	N	Y	N	Y	N	Y	Y	N
DG	Y	Y	Y	Y	Y	Y	Y	P

- Tensorflow: It is an open source artificial intelligence library, using data flow graphs to build models (TENSORFLOW, 2021).
- Numpy: It is the fundamental package for scientific computing in Python (NUMPY, 2021).
- Pandas: It is an open source data analysis and manipulation tool (PANDAS, 2021).
- Android Studio: It is an integrated development environment for developing for the Android platform. It was used to instrument the Android apps to capture the code coverage.
- Ui Automator: It is a cross-program test library developed by Google for the Android platform (UIAUTOMATOR, 2021).

Anaconda is the python distribution platform used for implementation. It has a toolkit to perform Python/R data science and machine learning on a single machine, facilitates the configuration and installation of scientific libraries. (ANACONDA, 2021). The computer configuration used is notebook 64-bit with Ubuntu 18.04.5, processor Intel Core i7-7600U CPU 2.70GHz GPU Nvidia GeForce 940MX and memory RAM 15,2 GB.

The Android environment implementation used the UI Automator python library to dump the screen hierarchy of the GUI in the starting activity of the AUT. The screen hierarchy is analyzed by searching for edit texts, clickable, long-clickable, checkable and scrollable. These activities are stored in a dictionary containing several associated attributes (resource-id, content-desc, class name, package and bounds) and compose the action vector. At each step, the agent takes action according to the behaviour of the exploration algorithm.

The DQN implementation reference in this research followed (PASZKE, 2017a) adapted to the Android environment; in this case, the model receives the app's screenshot as state, next

state, action, and the reward. The action is selected according to the epsilon greedy policy. It sometimes means a convolutional network chooses the action, and sometimes one other is chosen randomly with probability ε as shows the python code below.

```
1: EPS_START = 0.9
2: EPS_END = 0.05
3: EPS_DECAY = 300
4:
5: def select_action(state):
6: global steps_done
7: sample = random.random()
8: eps_threshold = EPS_END + (EPS_START - EPS_END)
9: math.exp(-1. * steps_done / EPS_DECAY)
10: steps_done += 1
11: if sample > eps_threshold:
12:         with torch.no_grad():
13:         vals = model(Variable(state.type(dtype))).data[0]
14:         max_idx = vals[:len(actions)].max(0)[1]
15:         return LongTensor([[max__idx]])
16: else:
17:         return LongTenso([[random.randrange(n__actions)]])
```

EPS_START and will decay exponentially towards *EPS_END*. *EPS_DECAY* controls the rate of the decay.Line 16 returns exploit (go to CNN) and line 18 returns exploit (choose ramdon action). The CNN python implementation is showed below.

```
19: class DQN(nn.Module):
20:     def __init__(self):
21:         super(DQN, self).__init__()
22:         self.conv1 = nn.Conv2d(3, 16, kernel_size=5, stride=2)
23:         self.bn1 = nn.BatchNorm2d(16)
24:         self.conv2 = nn.Conv2d(16, 32, kernel_size=5, stride=2)
25:         self.bn2 = nn.BatchNorm2d(32)
26:         self.conv3 = nn.Conv2d(32, 32, kernel_size=5, stride=2)
27:         self.bn3 = nn.BatchNorm2d(32)
28:         self.head = nn.Linear(448, 30)
29:
30:     def forward(self, x):
31:         x = F.relu(self.bn1(self.conv1(x)))
32:         x = F.relu(self.bn2(self.conv2(x)))
33:         x = F.relu(self.bn3(self.conv3(x)))
```

```
34:         x = x.view(x.size(0), -1)
35:         return self.head(x)
36: Transition = namedtuple('Transition', ('state', 'action', '
    next_state', 'reward'))
37:
38: class ReplayMemory(object):
39:
40:     def __init__(self, capacity):
41:         self.capacity = capacity
42:         self.memory = []
43:         self.position = 0
44:
45:     def push(self, *args):
46:         """Saves a transition."""
47:         if len(self.memory) < self.capacity:
48:             self.memory.append(None)
49:         self.memory[self.position] = Transition(*args)
50:         self.position = (self.position + 1) % self.capacity
51:
52:     def sample(self, batch_size):
53:         return random.sample(self.memory, batch_size)
54:
55:     def __len__(self):
56:         return len(self.memory)
```

CNN takes in the difference between the current and previous screen patches. The network predicts the expected return of taking each action given the current input. The class Replay memory stores the agent's experiences at each step, using a large batch size (256). It allows us to break the correlation between subsequent steps in the environment.

The optimize model function in DQN (python code below) performs a single step of the optimization. It first samples a batch, concatenates all the tensors into one, and combines them into our loss. The target network has its weights kept frozen most of the time but is updated with the policy network's weights every so often.

```
1: last_sync = 0
2: def optimize_model():
3:     global last_sync
4:     if len(memory) < BATCH_SIZE:
5:         return
6:     transitions = memory.sample(BATCH_SIZE)
```

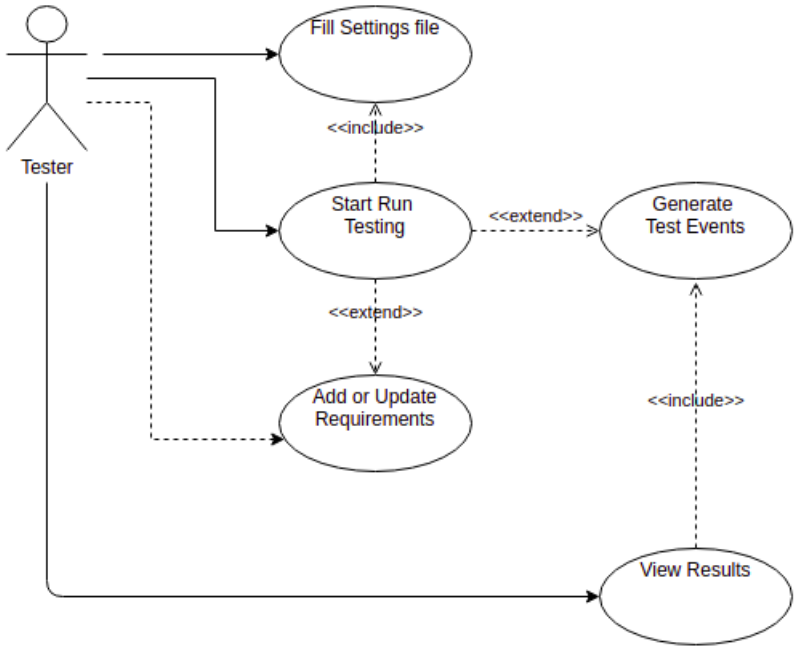
```
7:     batch = Transition(*zip(*transitions))
8:
9:     non_final_mask = BoolTensor(tuple(map(lambda s: s is not
       None, batch.next_state)))
10:    non_final_next_states_t = torch.cat(tuple(s for s in batch.
       next_state if s is not None)).type(dtype)
11:
12:
13:    with torch.no_grad():
14:
15:        non_final_next_states = Variable(
       non_final_next_states_t)
16:        state_batch = Variable(torch.cat(batch.state))
17:        action_batch = Variable(torch.cat(batch.action))
18:        reward_batch = Variable(torch.cat(batch.reward))
19:
20:        if USE_CUDA:
21:            state_batch = state_batch.cuda()
22:            action_batch = action_batch.cuda()
23:    # target network
24:    state_action_values = model(state_batch).gather(1,
       action_batch)
25:
26:    next_state_values = Variable(torch.zeros(BATCH_SIZE).type(
       Tensor))
27:    next_state_values[non_final_mask] = model(
       non_final_next_states).max(1)[0]
28:
29:    with torch.no_grad():
30:        next_state_values
31:
32:    # Bellman approximation
33:    expected_state_action_values = (next_state_values * GAMMA)
       + reward_batch
34:
35:    expected_state_len = len(expected_state_action_values)
36:    state_action_values = state_action_values.view(
       expected_state_len)
37:
```

```
38:     # Huber loss
39:     loss = F.smooth_l1_loss(state_action_values,
        expected_state_action_values)
40:
41:     # Optimize the model
42:     optimizer.zero_grad()
43:     loss.backward()
44:     for param in model.parameters():
45:         param.grad.data.clamp_(-1, 1)
46:     optimizer.step()
```

4.4 Modelling

In this section, the modeling of DeepRLGUIMAT is demonstrated. Figure 28 represents the use case diagram. Next, the defined use cases are detailed.

Figure 28 – Use Case Diagram

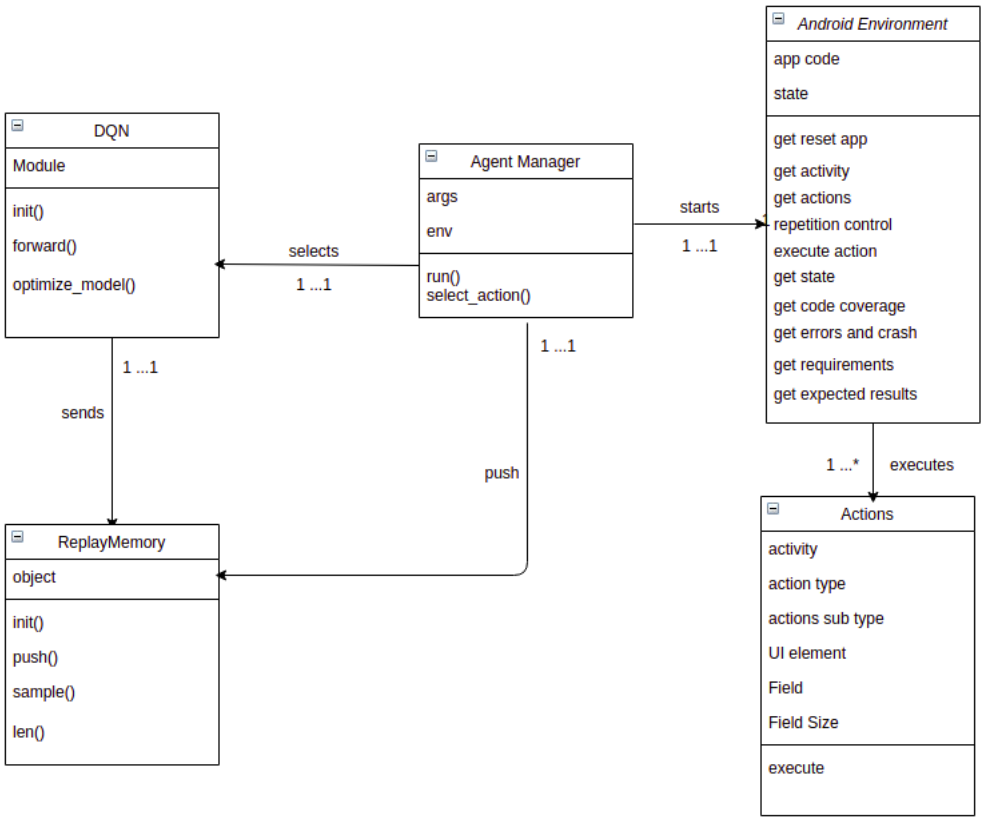


- **Fill Settings file**: The settings file has the information entry to the tool. The tester user must fill the file with the device's identifier connected to the computer or the emulator; insert the name of the apk to be tested and the source code address of the compiled application. The test time must also be filled in the file.

- **Start Run Test**: With the configuration information filled in, the tester must access the terminal and enter the commands to start running the tests.
- **Add or Update Requirements**: If the application has the requirements to perform the test exploration and creation, the user must fill in the fields of the requirements.csv file so that the application reads it and the requirements can guide the tests.
- **Generate Test Events**: As the tool runs, test events are generated and observed on the fly in the "test events" folder and the log file.
- **View Results**: At the end of execution, the tester user can view the results on files: test events, states, errors, and crashes.

Figure 29 maps the structure of the DeepRLGUIMAT tool showing the class diagram with the relationship between classes, attributes, and operations.

Figure 29 – Class Diagram



The agent module receives the configuration information args (device or emulator, app code address, apk, and execution time). The environment is initialized by agent to create the test events through execution and actions from this information. As discussed in previous sections, the environment captures the UI elements from the app to determine the possible actions to be performed.

When the actions are performed, this environment captures the state, code coverage, crashes, and error to be sent to the agent to calculate the reward. If there are no requirements, the DQN is trained randomly, and else the training will be carried out with the data-informed to guide the exploration. For each action executed, the tuple $(state, action, nextstate, reward)$ is stored in memory (ReplayMemory).

The DQN module receives the states and actions to enter in CNN to return the Q values $Q(s_t, a_t)$. The operation optimize model computes a mask of non-final states and concatenate the batch elements in ReplayMemory, the expected Q values, and Huber Loss for model optimization.

4.5 Operational Aspects

To install the tool, it is important, first, to obtain the following applications that are requirements:

- Python 3.8.X
- TensorFlow 2.2.0
- UI Automator 0.3.6
- Java
- Android SDK
- Add platform tools directory in Android SDK to PATH
- Linux: Ubuntu 18 or higher

The tool DeepRLGUIMAT is bundled into executable files to run in Linux. You don't need to install any other python dependency; just downloading all the files will suffice. The application under test may be in the DeepRLGUIMAT folder; the tool will install the app, and it can run in a physical phone connected to a computer or in an Android virtual machine. The download the DeepRLGUIMAT.zip available at:

$https://1drv.ms/u/s!Avu7qFzImegrh7FyBW7g0U4lddA9Qw?e=bOunEj$

Extracting the zip file in a folder, the structure of the folder appears as show Figure 30.

In this version of the DeepRLGUIMAT tool, there's no graphical user interface. The user should put the information of configuration in the settings.txt as shown in the Figure 31.

In APK NAME, the user should insert the path to the app. The PACKAGE is the name of the app package. In RESOLUTION is the information of device (or emulator) screen resolution. COVERAGE indicates that the app is instrumentalized, and the tool can collect the coverage file.

Figure 30 – Directory Structure

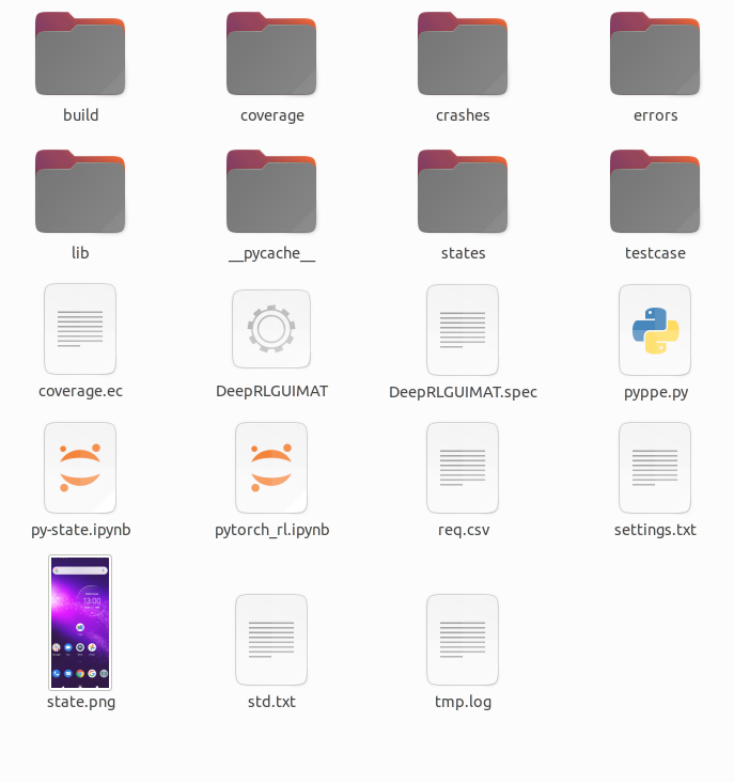


Figure 31 – File Settings

```
APK NAME:app.apk (or path until apk)
PACKAGE:app.package.com
RESOLUTION:1520x720
COVERAGE:yes
REQUIREMENT:yes
TIME:7200
```

REQUIREMENT specifies if the user filled the requirement CSV file. In Time, the user indicates how long the test will last in seconds.

If the user has the input entry requirements(REQUIREMENT:yes), the file requirements.csv should be filled, as shows the section 4.2.1.1 (Figure 23).

The user should open the terminal and type the commands ./DeepRLGUIMAT to start the tests. The tool will start the tests, the files log.txt, errors.txt, crash.txt, and the folders states and test events will be filled on the fly by the tool, like in Figure 30. The folder coverage files has the capture of code coverage during the test in case of apps instrumented. The folder states has the screenshots of the application of each action performed.The folder test case has the txt files with the test performed. The files crash, erros have the information of errors and crashes happened captured from Logcat, and finally the tmp.log file has the information of tool log execution.

4.6 Final Considerations

This section described the DeepRLGUIMAT approach to generate test cases through application exploration using DQN. In this approach, a deep Q-network agent is used to explore and model the application in a trial-and-error, reaching the best action to achieve a higher reward for new activities and test input variations using Systematic Functional Testing. The contribution of this work is offering a solution for the execution and monitoring of apps under test and generation of tests to help overcome the bottleneck of manual test design and repetitive test execution, assuring code coverage, finding failures, reproditibility and functional operations.

It is possible to highlight that this approach differs from other solutions in the literature by the concern to produce test inputs that really exercise the application's inputs in search of failures. It uses input requirements to guide exploration, which is a unique feature of this approach. In addition, there is also the concern of producing test events that are easy to trace and reproducible for the tester.

The next chapter will show the experimental study of the approach with the application of the tool in mobile applications and the analysis of the results obtained.

CHAPTER

5

EMPIRICAL STUDY

5.1 Initial Considerations

Empirical research is based on observed and measured phenomena and derives knowledge from experience rather than belief. Research may begin with a research question tested through verifiable experimentation. An experiment involves deliberately testing a hypothesis and concluding. The empirical study is useful in establishing the generalizability of results related to researchers' new subjects or data sets (WOHLIN; HÖST; HENNINGSSON, 2003).

This chapter presents the empirical experiment carried out with the DeepGUIMAT approach and the analyzed results. The tool was executed against 30 android applications. The experiment planning, research questions, hypotheses and environment are described in section 5.2. The results obtained are shown in section 5.3. They demonstrate the approach's feasibility as support for the automatic generation of tests in Android mobile apps, the higher code coverage and the functional coverage. The chapter also brings the case study of running the tool with input requirements in 5.4 and the Systematic Functional Testing (SFT) contribution analysis in 5.5. The threats of validity are analysed in section 5.6.

5.2 Empirical Experiment Planning

The evaluation of proposed tool DeepRLGUIMAT followed the Empirical Experiment, in this method an experiment is performed in order to either prove or disprove the research questions. The experiment compared the DeepRLGUIMAT to similar recognized approaches in the literature, they are: random test generation Monkey, the Model-based (with machine learning module) tool Droidbot, DroidBotX (droidbot with Reinforcement Learning-based) and Reinforcement Learning-based tool Q-testing in terms of code coverage (instruction, branch, line, and method), failures found, crashes and functionality coverage.

It was not possible to compare with the studies based on RL: S1, S2, S3, S5, S7, S8, S10, , S11, S13, and S30 because their tools are not available for download. The study S4 (Humanoid) presented issue that the deep learning is not running, the issue is in github but the author did not return to fix the problem. The study S7 (Aimdroid) also present problems to run, the author answered that the tool was discontinued. The tool Sapienz (MAO; HARMAN; JIA, 2016) could not be included because it was discontinued, it is closedsource and the available version is only compatible with Android 4.4 for Emulator (android version from 9 years ago not compatible with current apps), as parameter, the studies ((PAN *et al.*, 2020), and (YASIN; HAMID; YUSOF, 2021)) outperformed Sapienz.

This evaluation aims to remark the following research questions:

1. RQ1: Does DeepRLGUIMAT achieve higher code coverage in comparison to state-of-the-art testing tools?

2. RQ2: Is DeepRLGUIMAT able to find failures and crashes compared to state of art tools?

3. RQ3: Is DeepRLGUIMAT able to cover basic functional operations compared with state of the art tool?

These questions were defined to consolidate information that can serve as input to analyze whether DeepRLGUIMAT is a tool that can be used to answer the research question of this thesis, which asks: How can Deep Reinforcement Learning technique be incorporated into automated testing of mobile applications to contribute to the greater code coverage, detection of failures, and improve the test automation environment?

The metrics used in this evaluation are:

- Code coverage: the percentage of code which is covered by automated tests. A program with high test coverage has more of its source code executed during testing, which suggests it has a lower chance of containing undetected software failures compared to a program with low test coverage (GOPINATH; JENSEN; GROCE, 2014).

- Number of distinct failures: number of distinct failures (exceptions) returned by Android Logcat.

- Number of distinct crashes: number of distinct failures which cause unexpected exit caused by an unhandled exception or signal.

- Input types: number of distinct input types, such as text, number, symbols and so on.

- Operations coverage: the percentage of functional operations of the apps covered by tests.

5.2.1 Hypotheses

Hypotheses are the basis for conducting an experiment. A hypothesis is formally stated and the data collected during the course of the experiment is used to, if possible, reject the hypothesis at certain risks (BASILI; SELBY; HUTCHENS, 1986). There are two types of hypotheses, null hypotheses and alternative hypotheses. The null hypothesis states that there are no real trends or differences in the experiment. The alternative hypothesis is when the null hypothesis is false. The hypotheses for this experiment are

- RQ1: Does DeepRLGUIMAT achieve higher code coverage in comparison to state-of-the-art testing tools?
  - Null Hypothesis (H0): there was no difference between the code coverage values between DeepRLGUIMAT and the state of the art tools.
  - Alternative Hypothesis 1 (H1): code coverage values of the state of the art tools are higher than DeepRLGUIMAT.
  - Alternative Hypothesis 2 (H2): code coverage values of the DeepRLGUIMAT are higher than the state of the art tool.
- RQ2: Is DeepRLGUIMAT able to find failures and crashes compared to state of art tools?
  - Null Hypothesis (H0): there was no difference between the failures and crashes values between DeepRLGUIMAT and the state of the art tools.
  - Alternative Hypothesis 1 (H1): failures and crashes values of the state of the art tools are higher than DeepRLGUIMAT.
  - Alternative Hypothesis 2 (H2): failures and crashes values of the DeepRLGUIMAT are higher than the state of the art tool.
- RQ3: Is DeepRLGUIMAT able to cover basic functional operations compared with state of the art tool?
  - Null Hypothesis (H0): there was no difference between the functional operations coverage between DeepRLGUIMAT and the state of the art tools.
  - Alternative Hypothesis 1 (H1): functional operations coverage values of the state of the art tools are higher than DeepRLGUIMAT.
  - Alternative Hypothesis 2 (H2): functional operations values of the DeepRLGUIMAT are higher than the state of the art tool.

5.2.2 Empirical Experiment Design

The experiment design describes how the tests were organized and performed. The environment setup to run the experiments used the following configuration: Notebook 64-bit with

Ubuntu 18.04.5, processor Intel Core i7-7600U CPU 2.70GHz GPU Nvidia GeForce 940MX and memory RAM 15,2 GB. All experiments ran on smartphone dedicated to this experiment, model Motorola G8 Plus, Operational system Android 9 (API level 28), 4GB memory. The smartphone was connected by cable using USB port. In order to prevent data from previous sessions, all the data on the smartphone was removed and the device was restarted before starting a test cycle.

The choice of evaluation in a real device instead of an Android Virtual Machine aims to provide feedback regarding the performance of the tool since the objective of this tool is also to be able to run adequately to be used in real devices.

DeepRLGUIMAT and the state of the art tools were evaluated in 30 mobile applications from the F-droid open source repository (F-DROID, 2021), the apps are described in Table 11. All these apps were selected for having a diversity of GUI elements and interaction such as Edit Texts, Buttons, Image Buttons, links, checkboxes, radio buttons, Spinners, date pickers, options menu, popup menu, and list select, and they are open source. All the apps were instrumented by the bytecode using the tool JoCoCo (KHAMARU., 2017). For each test cycle the tool ran for 2 hours of test cycle to make sure DQN training to cover the applications. The state of the art tools also ran during 2 hours. We consider this time necessary to cover that app's functionalities according to the apps' size. We set the device actions commands (Home, Back, Menu, Rotate and Volume) probability to 5% provides a balance between short and long test cases (ADAMO *et al.*, 2018).The state of the art tools ran without input requirements, so DeepRLGUIMAT in this experiment ran in without input requirement mode for a fair comparison.

Monkey, Droidbot, DroidbotX and Q-testing were performed as default configurations indicated on its website in the experiment. Although, for a fair comparison between test tools, the apps' requirements were not considered input for DeepRLGUIMAT. In order to evaluate this aspect, an experiment considering a set of requirements of an application form was executed and will be described in the following sections.

The tools were executed 4 times in each application to avoid some possible effect of randomness during testing and confirm the trend of the results. The test cycle in each app generates many coverage files because when the action causes an application crash, the coverage file is copied from the device to a directory. The application is restarted, and the coverage file is written again; it is necessary to maintain the history of coverage values to calculate the values.

5.3 Results

In this section, the results obtained are discussed. Regarding RQ1, Table 11 shows the code coverage results for each application. Column DG is DeepRLGUIMAT, column QT is Q-Testing, column DB is Droidbot, column DBX is DroidbotX, and column M is Monkey. DeepRLGUIMAT presented higher values of code coverage are highlighted in grey.

Table 10 – 30 Apps

Id	Apps	Version	Instructions	Branches	Lines	Methods
App1	Add Loan	1.0.1	9.376	494	1.974	327
App2	Atime Track	1.0	8.091	531	1.657	187
App3	Better Count	1.6	1.335	106	358	48
App4	Biever	1.2.1	31.561	1.642	7.278	1.774
App5	Bmi	1.0	3.106	188	620	92
App6	Budget Watch	0.21.4	9.573	629	2.138	277
App7	Catima	1.7.1	10.123	692	2.212	338
App8	Cavevin	1.2.1	7.477	408	1.524	248
App9	Dailypill	1.0	2.006	150	655	137
App10	Exceer	0.2.3	4.957	360	1.024	201
App11	Farmer diary	1.72	8.159	868	2.036	260
App12	Food scale	1.2	717	34	143	27
App13	Grocy	1.10.1	88.689	8.529	20.831	3.457
App14	Just do	2.1	13.382	198	853	94
App15	Log28	0.6.2	6.104	357	1.367	113
App16	Medclin	0.2.8	2.357	122	622	86
App17	Meditation Assistant	1.6.3	37.581	3.427	13.121	1.506
App18	Money track	2.1.3	36.297	907	5.275	1.026
App19	Music	1.3.4	50.362	3.733	18.812	4.107
App20	Openfood	3.2.8	81.890	6.424	31.568	6.227
App21	Open workout	1.3.1	23.533	1.496	9.475	1.776
App22	Pass generator	1.4.2	4.227	280	909	204
App23	Periodical	1.64	28.715	667	3.253	362
App24	Silinote	1.0	2.335	60	799	187
App25	Simpledo	1.0	5.295	476	986	118
App26	Textpad	1.0	14.483	210	1.614	286
App27	Time Table	1.0	19.950	994	7.128	1.072
App28	Todo list	1.1	757	34	275	55
App29	Tricky	1.6.2	35.302	2.414	14.485	2.914
App30	Water droid	1.0	7.148	78	696	132

In apps, Add Loan (App1), Simpledo (App25), Bettercount (App3), Biever (App4), Ex-

Table 11 – Code Coverage in 30 Apps

	Instruction Coverage					Branch Coverage					Line Coverage					Method Coverage				
APPS	**DG**	**QT**	**DB**	**DBX**	**M**	**DG**	**QT**	**DB**	**DBX**	**M**	**DG**	**QT**	**DB**	**DBX**	**M**	**DG**	**QT**	**DB**	**DBX**	**M**
App1	0.57	0.21	0,14	0.21	0,04	0.37	0.15	0,05	0.09	0,01	0.59	0.20	0,14	0.22	0,04	0.71	0.28	0,2	0.31	0,07
App2	0.63	0.55	0,37	0.28	0,23	0.43	0.44	0,21	0.13	0,1	0,62	0.54	0,37	0.25	0,22	0.68	0.56	0,5	0.29	0,28
App3	0.55	0.24	0.41	0.55	0.24	0.35	0.12	0.25	0.37	0,13	0.57	0.22	0.41	0.56	0,22	0.75	0.41	0.60	0.68	0,41
App4	0.50	0.33	0.25	0.32	0.43	0.25	0.12	0.07	0.12	0,21	0.51	0.32	0.23	0.31	0,43	0.50	0.29	0.20	0.27	0,42
App5	0.74	0.74	0.51	0.1	0.59	0.43	0.43	0.23	0.1	0,32	0.73	0.73	0.50	0.1	0,56	0.84	0.84	0.64	0.1	0.63
App6	0.65	0.41	0,43	0.32	0,26	0,42	0.22	0,25	0.18	0,14	0.67	0.42	0,45	0.34	0,28	0.77	0.53	0,56	0.52	0,35
App7	0.67	0.27	0.11	0.66	0,61	0.44	0.15	0.05	0.44	0,45	0.67	0.28	0.12	0.66	0,6	0.72	0.34	0.17	0.72	0,65
App8	0.44	0.30	0.28	0.24	0.40	0.33	0.24	0.23	0.09	0.32	0.44	0.30	0.30	0.24	0.42	0.53	0.39	0.39	0.32	0.51
App9	0.78	0.45	0.75	0.76	0,78	0.54	0.20	0.42	0.45	0,54	0.81	0.41	0.78	0.79	0,81	0.83	0.51	0.82	0.83	0,83
App10	0.66	0.42	0.42	0.54	0,3	0.57	0.29	0.30	0.42	0,21	0.68	0.46	0.46	0.52	0,33	0.75	0.61	0.61	0.71	0,39
App11	0.53	0.53	0.34	0.03	0,54	0.37	0.39	0.23	0.01	0,37	0.53	0,54	0.35	0.03	0,54	0.69	0.64	0.45	0.06	0,69
App12	0.91	0.30	0.26	0.48	0.42	0.79	0.17	0.02	0.32	0.29	0.88	0.30	0.25	0.49	0.38	0.85	0.33	0.22	0.59	0.48
App13	0.28	0.22	0.23	0.26	0,19	0.18	0.13	0.14	0.17	0,12	0.27	0,21	0.22	0.25	0,19	0.31	0.23	0.25	0.28	0,23
App14	0.68	0.69	0.46	0.46	0.42	0.32	0.29	0.39	0.41	0.20	0.60	0.44	0.52	0.52	0.39	0.82	0.37	0.53	0.68	0.39
App15	0.83	0.52	0.56	0.30	0.86	0.57	0.29	0.29	0.08	0.61	0.85	0.52	0.60	0.32	0.88	0.87	0.53	0.66	0.31	0.91
App16	0,81	0.26	0.73	0.84	0.74	0,64	0.22	0.46	0.63	0.56	0,84	0.28	0.77	0.86	0,76	0.86	0.48	0.84	0.88	0,76
App17	0.35	0.34	0.20	0.42	0.33	0.24	0.24	0.11	0.28	0.24	0.35	0.34	0.21	0.42	0.34	0.41	0.39	0.28	0.50	0.41
App18	0.65	0.31	0.33	0.36	0,47	0.48	0.25	0.15	0.20	0,36	0.65	0.30	0.28	0.33	0,46	0.70	0.36	0.33	0.38	0,54
App19	0,35	0.18	0.20	0.19	0,3	0,24	0.10	0.12	0.11	0,23	0,39	0.20	0.22	0.21	0,34	0.46	0.25	0.28	0.27	0,39
App20	0.25	0,25	0.09	0.10	0.11	0.13	0,13	0.03	0.03	0.03	0.26	0,25	0.09	0.11	0.10	0.30	0,26	0.09	0.13	0,1
App21	0.26	0.18	0.25	0.14	0,1	0.15	0.09	0.12	0.06	0,04	0.28	0,19	0.27	0.14	0,11	0.31	0,2	0.33	0.16	0,13
App22	0.75	0.75	0.49	0.65	0,57	0.64	0.63	0.34	0.51	0,43	0.79	0.79	0.54	0.70	0,64	0.82	0.78	0.55	0.74	0,61
App23	0.64	0.61	0.56	0.53	0,59	0.47	0.62	0.49	0.40	0,57	0.62	0.62	0.59	0.51	0,6	0.66	0.69	0.53	0.44	0,56
App24	0.66	0.62	0.56	0.77	0,49	0.36	0.28	0.28	0.48	0,28	0.65	0.62	0.55	0.76	0,48	0.75	0.71	0.61	0.81	0.57
App25	0.48	0,22	0,34	0.30	0,21	0.29	0.06	0,14	0.14	0,07	0.52	0.24	0,36	0.36	0,23	0.61	0.29	0,48	0.48	0,35
App26	0.83	0.15	0.22	0.10	0,83	0.73	0.53	0.12	0.34	0,7	0,83	0.50	0.23	0.33	0,83	0,83	0.54	0.25	0.38	0,83
App27	0.35	0.33	0,22	0.24	0,19	0.22	0.22	0,12	0.15	0,12	0.36	0.32	0,2	0.24	0,19	0.43	0.22	0,22	0.28	0,14
App28	0.62	0.33	0.34	0.45	0,36	0.38	0.11	0.14	0.29	0,17	0.63	0.28	0.30	0.46	0,36	0.69	0.43	0.45	0.59	0,51
App29	0.53	0.15	0.17	0.24	0,19	0.35	0.07	0.08	0.12	0,09	0.53	0.17	0.19	0.25	0,21	0.59	0.20	0.23	0.30	0,25
App30	0.37	0.37	0.35	0.36	0,35	0.56	0.52	0.38	0.51	0,38	0.57	0.57	0.51	0.56	0,51	0.53	0.53	0.49	0.53	0,51

ceer(App10), Foodscale(App12), JustDo (App14), Meditation Assistant(App17) and Waterdroid (App30) have GUIs that mostly require simple interactions with text input fields and a few form validation rules. The Apps MoneyTrack (App 18) and BudgetWatch(App6) present form validation rules for the size and type of inputs. The input variation of DeepRLGUIMAT, even without requirements, could perform the forms that were not observed in other tools.

DeepRLGUIMAT has the function Control Repetition discourages repeating the same action in the same state context (e.g., insert small text in an edit text several times) provides access to most AUT's functionality. Q-testing can easily access the screen with many GUI elements (like a calendar) faster than DeepRLGUIMAT, which needed more time to analyze each UI element to choose the best choice. Figure 32 shows the variance of coverage values: Inst represents instruction coverage, Branch is the branch coverage, Line is the line coverage, and Method is the method coverage. In red the proposed tool DeepRLGUIMAT, in green Q-testing, in blue Monkey, in magenta Droidbot and black DroidbotX.

Table 12 – Coverage (p-value)

	DeepRLGUIMAT x Monkey (p-value)	**DeepRLGUIMAT x Q-Testing (p-value)**	**DeepRLGUIMAT x DroidBot (p-value)**	**DeepRLGUIMAT x DroidBotX (p-value)**
Instruction	0.0028	0.0001	0.00004	0.00058
Branch	0.0056	0.0005	0.00002	0.0021
Line	0.0028	0.0001	0.00006	0.00059
Method	0.0015	0.0001	0.00009	0.00113

Figure 32 – Coverage boxplot

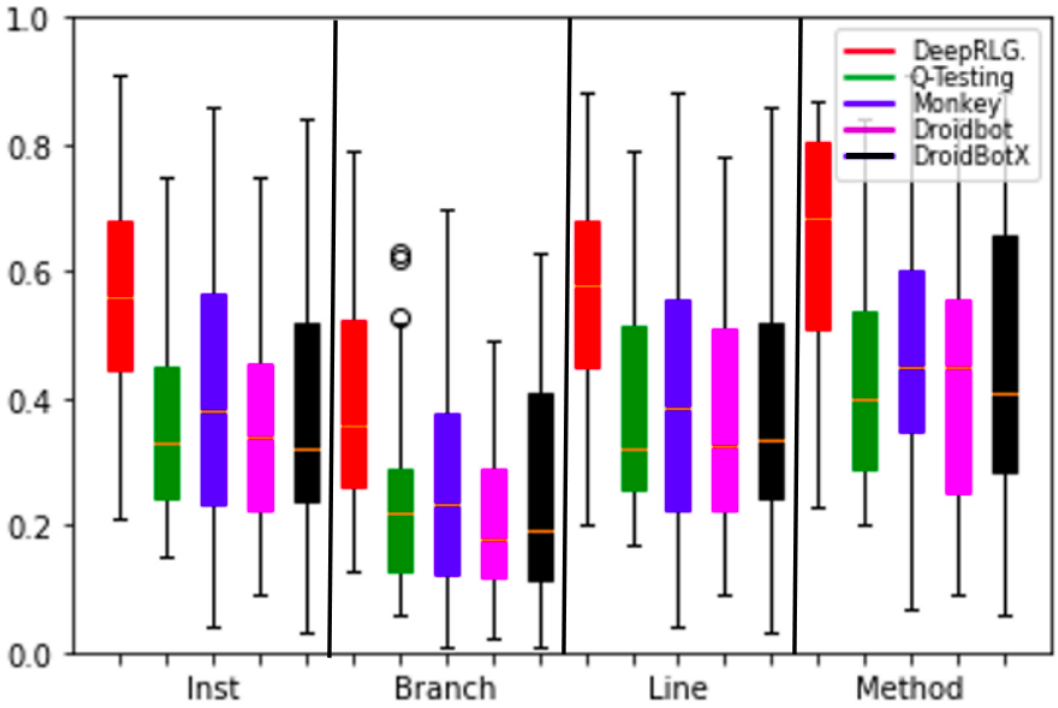
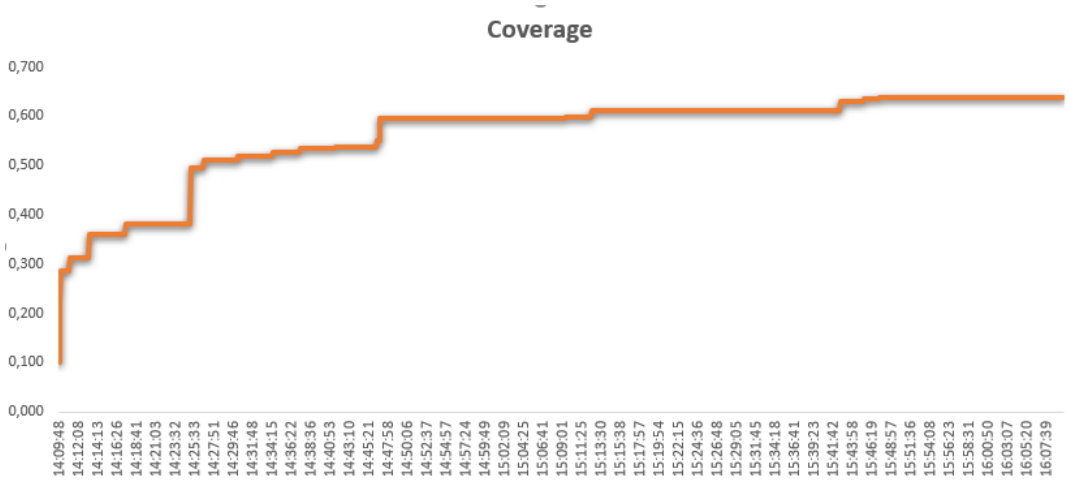


Table 12 shows the DeepRLGUIMAT performs better than other tools (Monkey and Q-testing) obtained higher code coverage at a statistically (The Mann-Whitney U-test, (MANN; WHITNEY, 1947)) significant level ($\rho < 0.05$) for instruction, branch, line and method coverage. The difference in coverage between the Monkey, Q-Testing, and DeepRLGUIMAT is significant. The null hypothesis was rejected and the alternative hypothesis 2 (H2) was confirmed.

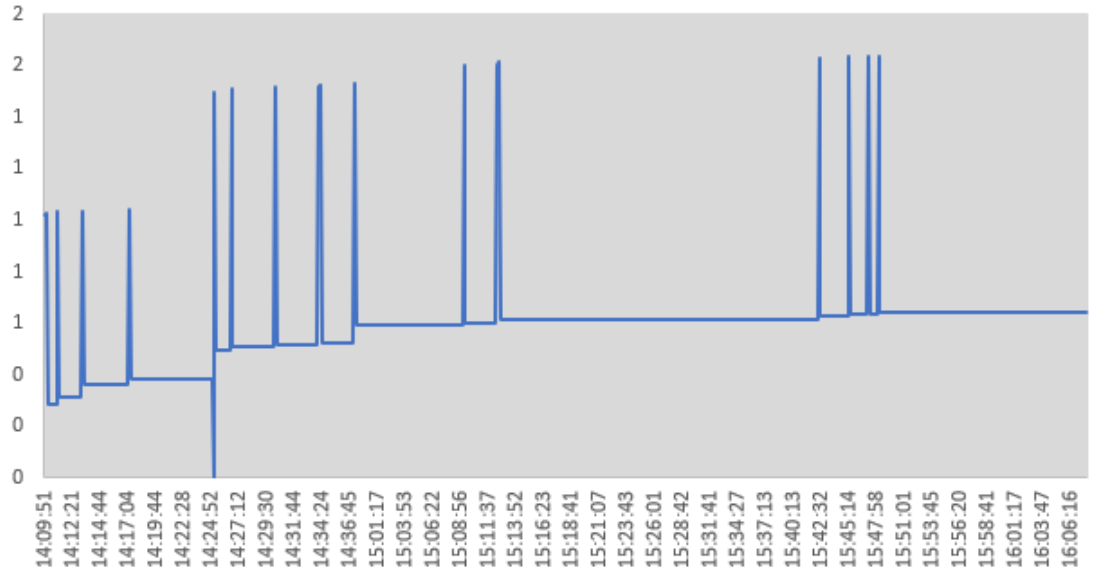
Figure 33 represents the trend of coverage values during the two hours of the test. After 30 minutes, the DQN is most frequently choosing the actions in training steps. This is the representation of the majority of apps behaviour, but the growth depends on the app characteristics such as size, rules, and UI elements.

Figure 33 – Coverage trends



The reward guides the agent DQN for the best selection of actions for app testing. Figure 34 shows the trend of average reward values over two hours of execution. We can observe that there is a discrete linear trend with higher peaks when the application exercises a new screen.

Figure 34 – Reward values during test execution



An important aspect of test tools is the ability to discover failures and crashes. The functional failures could not be confirmed because the apps have no requirements, so, in this evaluation we consider just the Logcat Android exceptions for RQ2. Table 14 shows the number of distinct failures and crashes found during execution in each application. This table lists the applications (Apps column), the number of failures found , and the number of crashes for each test tool. All failures and crashes found were observed in the android logcat exceptions and confirmed manually. The types of exceptions faults found are: InputDispatcher, java.lang.RuntimeException, AndroidRuntime Process, Unable to start activity, and Unable to stop activity. DeepRLGUIMAT found more failures in Time Table, Budget Watch, Dailypill, Foodscale, Grocy, Meditation Assistant, Openworkout, Passgenerator, Periodical, Waterdroid and Justdo apps. In Cavevin and Farmerdiary, Q-Testing and Monkey found one more distinct failure caused by request permission.

Table 13 – Faiures and Crashes (p-value)

	DeepRLGUIMAT x Monkey	**DeepRLGUIMAT x Q-Testing**	**DeepRLGUIMAT x DroidBot**	**DeepRLGUIMAT x DroidBotX**
Failures	0,094	0,11573	0,08068	0,00157
Crashes	0,42238	0,42238	0,47601	0,1431

The tools found the same distinct crashes except in the BMI, Foodscale, Openworkout app that DeepRLGUIMAT found a crash not discovered by other tools. In Cavevin, Music, Textpad, and Todo list, Q-Testing and Monkey found one more crash as indicated in Table 14. It

Table 14 – Distinct Failures and Crashes

	DG		QT		M		DB		DBX	
APPS	**F**	**C**	**F**	**C**	**F**	**C**	**F**	**C**	**F**	**C**
AddLoan	0	0	0	0	0	0	0	0	0	0
Atimetrack	2	0	2	0	2	0	0	0	1	0
Bettercount	2	0	2	0	2	0	0	0	0	0
Biever	5	0	5	0	5	0	1	0	2	0
Bmi	2	1	2	0	2	0	2	1	0	0
Budget Watch	2	0	0	0	0	0	0	0	2	0
Catima	3	1	3	1	3	1	2	1	2	0
Cavevin	2	0	3	1	3	1	2	0	2	0
Dailypill	3	1	2	1	2	1	1	1	1	0
Farmerdiary	1	0	2	0	2	0	2	0	1	0
Foodscale	1	1	0	0	0	0	1	1	0	0
Grocy	3	1	1	1	1	1	3	1	2	0
Justdo	1	0	0	0	0	0	1	0	0	0
Log28	0	0	0	0	0	0	1	0	1	0
Medclin	2	0	2	0	2	0	2	0	0	0
Meditation Assistant	2	0	1	0	1	0	2	0	2	0
Money track	0	0	0	0	0	0	1	0	3	0
Music	1	0	1	1	1	1	1	0	0	0
Openfood	1	0	1	0	1	0	1	0	0	0
Openworkout	1	2	0	0	0	0	1	1	0	1
Passgenerator	1	0	0	0	0	0	0	0	0	0
Periodical	1	0	0	0	0	0	1	0	0	0
Silinote	0	0	1	0	1	0	1	0	1	0
Simpledo	2	2	2	2	2	2	2	2	0	0
Textpad	1	0	1	1	1	1	0	0	0	0
TimeTable	2	0	1	0	1	0	1	0	0	0
Todo list	2	0	2	1	2	1	1	0	0	1
Tricky	1	0	1	0	1	0	2	0	1	0
Waterdroid	1	0	0	0	0	0	1	0	0	0
exceer	0	0	0	0	0	0	0	0	0	0

happened because they closed the apps and started without the file write permission, causing the crash because of the JaCoCo tool's instrumentation. Table 13 shows that difference in the number of failures is not statistically relevant for tools Monkey, Q-Testing and Droidbot. For DroidbotX the p-value indicates DeepRLGUIMAT can find more failures statistically. In the case of crashes, the difference in the values with all tools is not statistically relevant. So, we consider confirming the null hypothesis for RQ2.

The ability to create and perform tests to exercise functionalities is observed during the test cycle. Since the apps have no requirement documentation, to answer RQ3, we performed an analysis that listed the primary functional operations of the apps according to GUI. We also searched the F-droid and GitHub repositories for description information to help to list the

functions. We considered operation coverage not just going in and out of the app screen, but performing actions, data entry and performing, e.g. saving data, editing data, removing data and so on. The complete table of functional operations of 30 apps is available in annexe A. Table 15 shows the values of functional coverage obtained by this analysis.

Table 15 – Functional Operations Coverage

APPS	**DG**	**QT**	**DB**	**DBX**	**M**
App1	1	0.7	0.6	0.20	0.5
App2	0.86	0.73	0.33	0.21	0.3
App3	0.83	0.33	0.83	0.83	0.33
App4	1	0.76	0.3	0.38	0.4
App5	1	1	0.5	0.5	0.5
App6	0.62	0.55	0.37	0.25	0.20
App7	1	0.6	0.4	0.6	0.6
App8	0.45	0.27	0.54	0.1	0.2
App9	1	0.40	0.8	0.8	0.6
App10	1	0.75	1	1	1
App11	0.81	0.81	0.72	0.63	0.65
App12	1	0.66	0.33	0	0.33
App13	0.20	0.14	0.1	0.1	0.1
App14	0.85	0.62	0.28	0.42	0.18
App15	1	0.60	0.60	0.4	1
App16	0.83	0.66	0.5	0.33	0.7
App17	0.75	0.75	0.75	0.87	0.75
App18	0.5	0.31	0.11	0.1	0.11
App19	0.71	0.42	0.14	0.28	0.6
App20	0.50	0.50	0.25	0.25	0.15
App21	0.72	0.36	0.27	0.1	0.1
App22	1	1	0.4	0.5	0.4
App23	0.66	0.66	0.66	0.5	0.6
App24	1	0.83	0.33	0.5	0.5
App25	0.8	0.6	0.6	0.6	0.6
App26	1	0.6	0.6	0.6	0.5
App27	0.47	0.43	0,2	0,2	0.2
App28	0.37	0.25	0.12	0.25	0.25
App29	0.62	0.34	0.25	0.37	0.25
App30	1	1	0.66	0.66	0.66

A p-value statistical test was performed based on these values, as shown in table 16. The results indicated that the difference in operations coverage between DeepRLGUIMAT and the state of the art tools is statistically relevant. The null hypothesis was rejected, and the alternative hypothesis 2 (H2) was confirmed for RQ3.

The greatest variation in data entry values was a relevant factor in the coverage of features. DeepRLGUIMAT generates more input entries as shown in Chapter 4 (Table 9), for example, in apps with forms where the rule was to accept only numbers (as in Foodscale and MedicLog), or

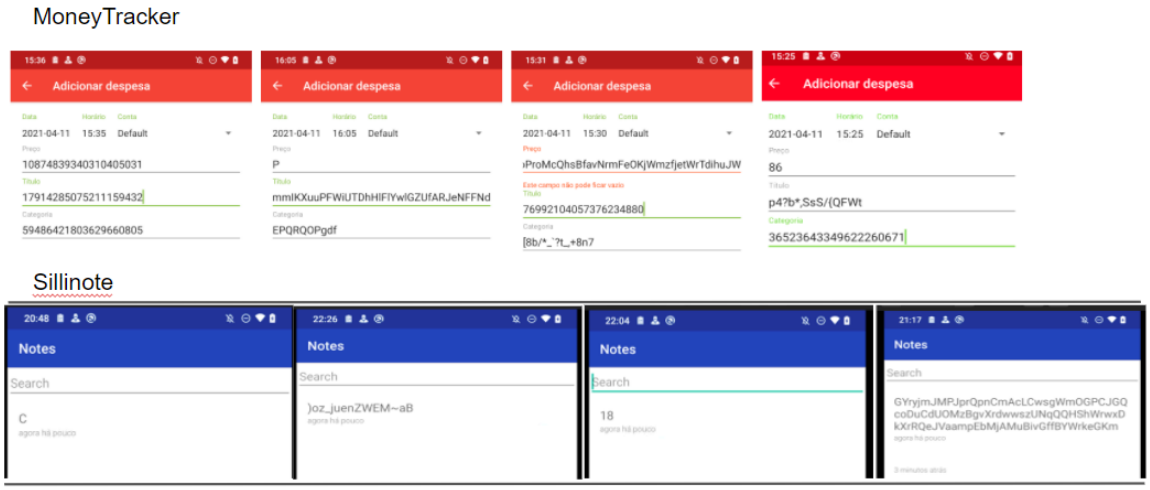
Table 16 – Functional Operations p-value

	DeepRLGUIMAT x Monkey	DeepRLGUIMAT x Q-Testing	DeepRLGUIMAT x DroidBot	DeepRLGUIMAT x DroidBotX
Functional Operations	0.00003	0.000692	0.00002	0.00002

as in the case where one of the fields only accepted numbers of a certain size (Budget Watch and Money Track) the tool DeepRLGUIMAT was able to operate, while the Q-testing tool in some apps could not. In the case of apps without field entry rules, Q-testing achieved similar coverage (BMI, Silinote, Periodical, Open food and Waterdroid).

Figure 35 demonstrates that the DeepRLGUIMAT tool tries to insert various types of different inputs in a text field: numbers, letters, and symbols in different sizes. It covers the basic operation and alternative flow, as in the example, the input error message in red.

Figure 35 – DeepRLGUIMAT inputs example

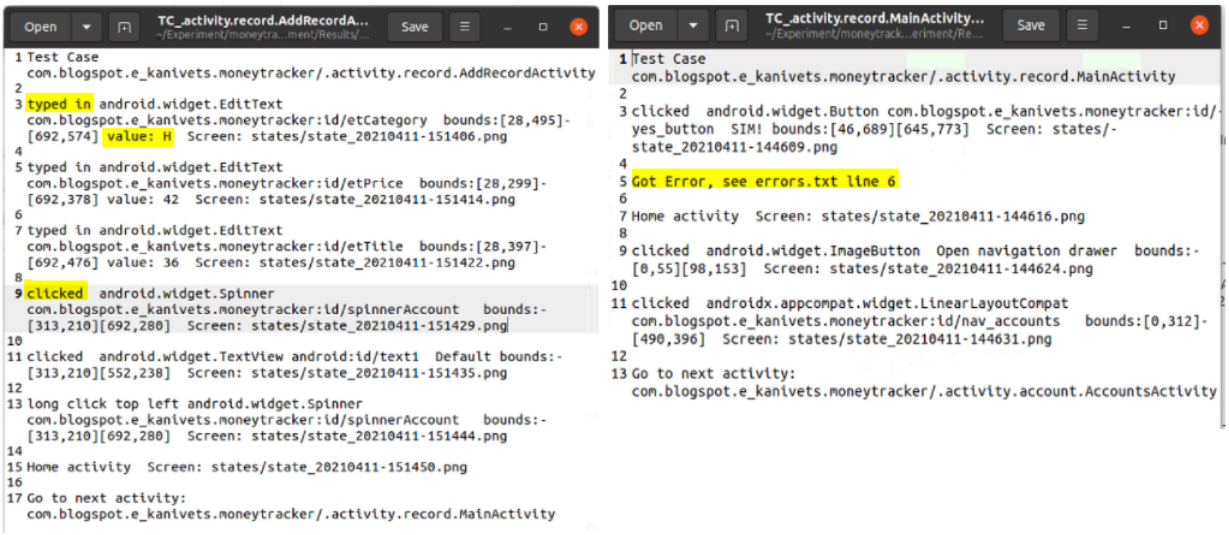


Another feature that helps to identify and track test events is the file generated by the tool (Figure 36) with all actions performed, application elements, activity, and errors, in case they occur.

5.4 DeepRLGUIMAT case study with input Requirements

In this section, the results of the tool analysis in a case study with the requirements input functionality are shown. Figure 36 shows the table of requirements as input for DeepRLGUIMAT describer in Chapter 4 (section 4.2.1.1). The tool will read the requirements and the test input generated will comply with the requirements of type field following the SFT guidelines. For each input, the possible actions generated are: input value, input greater than start size, input less than start size, input greater than end size, input less than end size, and input symbols.The app used in this study is Money Tracker, this app has a form with character size and type restriction.

Figure 36 – Test Events



```
Test Case
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity

typed in android.widget.EditText
com.blogspot.e_kanivets.moneytracker:id/etCategory  bounds:[28,495]-
[692,574] value: H  Screen: states/state_20210411-151406.png

typed in android.widget.EditText
com.blogspot.e_kanivets.moneytracker:id/etPrice  bounds:[28,299]-
[692,378] value: 42  Screen: states/state_20210411-151414.png

typed in android.widget.EditText
com.blogspot.e_kanivets.moneytracker:id/etTitle  bounds:[28,397]-
[692,476] value: 36  Screen: states/state_20210411-151422.png

clicked  android.widget.Spinner
com.blogspot.e_kanivets.moneytracker:id/spinnerAccount  bounds:-
[313,210][692,280]  Screen: states/state_20210411-151429.png

clicked  android.widget.TextView android:id/text1  Default bounds:-
[313,210][552,238]  Screen: states/state_20210411-151435.png

long click top left android.widget.Spinner
com.blogspot.e_kanivets.moneytracker:id/spinnerAccount  bounds:-
[313,210][692,280]  Screen: states/state_20210411-151444.png

Home activity  Screen: states/state_20210411-151450.png

Go to next activity:
com.blogspot.e_kanivets.moneytracker/.activity.record.MainActivity
```

```
Test Case
com.blogspot.e_kanivets.moneytracker/.activity.record.MainActivity

clicked  android.widget.Button com.blogspot.e_kanivets.moneytracker:id/-
yes_button  SIM! bounds:[46,689][645,773]  Screen: states/-
state_20210411-144609.png

Got Error, see errors.txt line 6

Home activity  Screen: states/state_20210411-144616.png

clicked  android.widget.ImageButton  Open navigation drawer  bounds:-
[0,55][98,153]  Screen: states/state_20210411-144624.png

clicked  androidx.appcompat.widget.LinearLayoutCompat
com.blogspot.e_kanivets.moneytracker:id/nav_accounts  bounds:[0,312]-
[490,396]  Screen: states/state_20210411-144631.png

Go to next activity:
com.blogspot.e_kanivets.moneytracker/.activity.account.AccountsActivity
```

The screen analyzed is Add Record, it has 3 fields type edit text, the field price has the input type number the size minimum 1 and maximum of 13 characters. The field Title is text type the minimum size is 1 and the maximum is 50. Same for the Category text field.

Figure 37 – DeepRLGUIMAT Money Tracker Requirements

activity	field	id	action	type	size_star	size_end	value	Result_activity	Result_elem	Result_text
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/etPrice	type	number	1	13	1500			
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/etTitle	type	text	1	50	test			
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/etCategory	type	text	1	50	bill			
com.blogspot.e_kanivets.moneytracker/.activity.record.AddRecordActivity	button	com.blogspot.e_kanivets.moneytracker:id/fabDone	click	none				com.blogspot.e_kanivets.moneytracker/.activity.record.MainActivity	com.blogspot.e_kanivets.moneytracker:id/tvTitle	test
com.blogspot.e_kanivets.moneytracker/.activity.account.AddAccountActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/etTitle	type	text	1	50	account			
com.blogspot.e_kanivets.moneytracker/.activity.account.AddAccountActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/et_init_sum	type	number	1	13	200			
com.blogspot.e_kanivets.moneytracker/.activity.account.AddAccountActivity	button	com.blogspot.e_kanivets.moneytracker:id/action_done	click	none				com.blogspot.e_kanivets.mo	com.blogspot.e_kanivets.moneytracker:id/tvTitle	account
com.blogspot.e_kanivets.moneytracker/.activity.exchange_rate.AddExchangeRateActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/et_buy	type	number	1	13	500			
com.blogspot.e_kanivets.moneytracker/.activity.exchange_rate.AddExchangeRateActivity	edittext	com.blogspot.e_kanivets.moneytracker:id/et_sell	type	number	1	13	2000			
com.blogspot.e_kanivets.moneytracker/.activity.exchange_rate.AddExchangeRateActivity	button	com.blogspot.e_kanivets.moneytracker:id/action_done	click	none				com.blogspot.e_kanivets.mo	com.blogspot.e_kanivets.moneytracker:id/tv_amount_buy	500

The tool ran around 2 hours 4 times. The average coverage results are displayed in table 17. The colum DG is the proposed tool without requirements input and the column DG-R is the tool with requirements. The coverage values have no significant variance.

Table 17 – Coverage Results

APP	**Instruction Coverage**		**Branch Coverage**		**Line Coverage**		**Method Coverage**	
	DG	**DG-R**	**DG**	**DG-R**	**DG**	**DG-R**	**DG**	**DG-R**
MoneyTracker	0,65	0,66	0,48	0,48	0,65	0,65	0.70	0,69

We observed that this approach DG-R generated more input types to forms than DG,

as shown the Table 18. The requirements allow generating inputs exploring the limit values of partition and exception cases such as insert symbols in field type number.

Table 18 – Inputs Generated

	Distinct Test Inputs	
Fields	**DG**	**DG-R**
Price	number size 2	Empty
	large number size 20	default value 1500
	small number size 1	small number size 2
	small letter size 1	small number size 1
	large text size 200	large number size 14
	symbols	large number size 13
		symbols
		medium letter size 10
Title	symbols	Empty
	large text size 200	Default text Test
	medium text size 10	small letter size 1
	large number size 10	large text size 51
	medium number size 2	large text size 50
	small number size 1	symbols
		number medium size 2
Category	large text size 200	Empty
	medium text size 10	default text Bill
	small number size 1	small letter size 1
	small letter size 1	large text size 51
	symbols	large text size 50
	large number size 20	symbols
	medium number size 2	number medium size 2

DG generated 28 files of test events for the activity Add Record while the tool with requirements generated 69 files of test events. Figure 38 shows the screenshot of the actions and Figure 39 the example of test events performed. In this file exist the lines 7 and 11 the Expected size of input informed when the tool inputs exception cases. It facilitates knowing when the tool is input, not valid values.

Regarding the failures found, the DG registered the exceptions returned by Logcat 1 android exception 11 times during tests, DG-R returned 3 distinct android exceptions 20 times in form Add Record. The failures regarding the interface details such as wrong error message text cannot be captured by the tool but we observed in screenshots.

We observed that DG-R focused the app exploration on-screen with requirements since the Agent DQN receives reward higher when select action of requirements, in this study, 42% of test events were generated for Add Record activity against 15,64% of DG. It provides the coverage of the requirements but in another side, the tool did not cover well other screens.

Figure 38 – Test Events Performed with Requirements

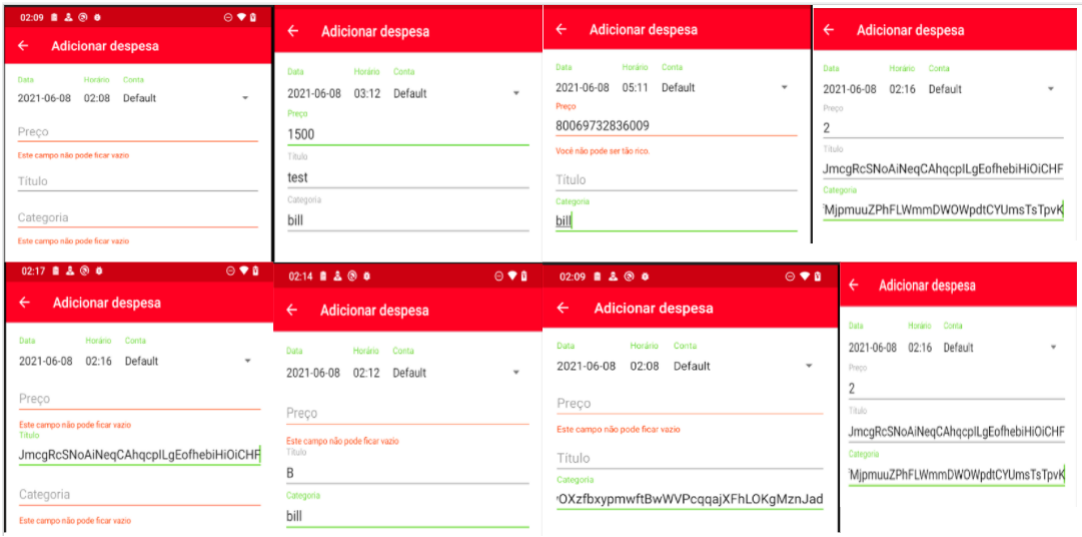
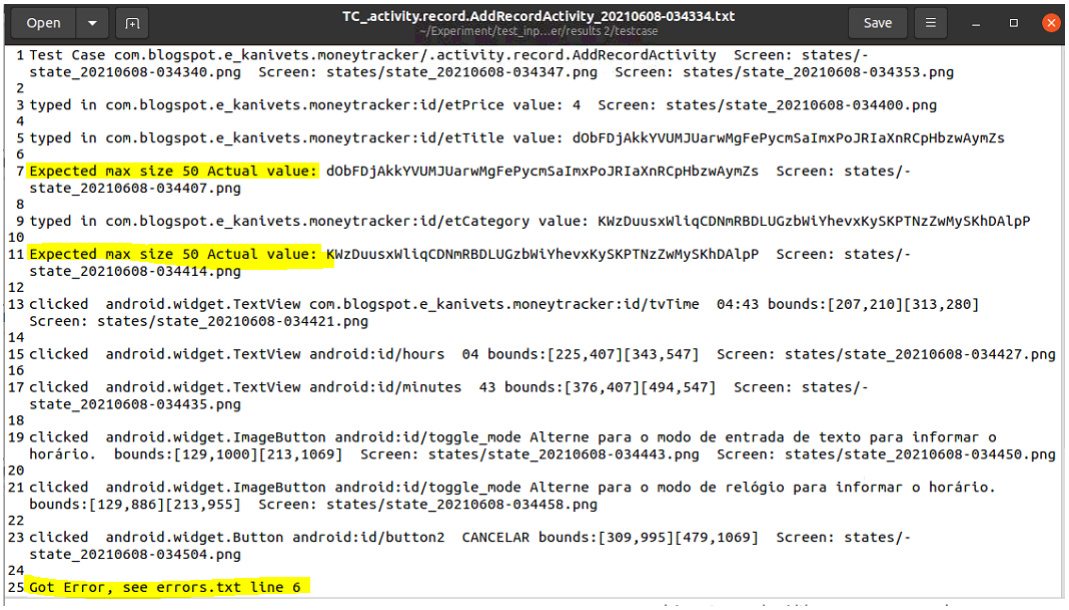


Figure 39 – Test Events Performed with Requirements



5.5 Test Input based on Systematic Functional Testing

In this section, we analysed if SFT contributed to this approach comparing the performance of the DeepRLGUIMAT in terms of code coverage using for comparison a previous version of the tool (DG1) without the use of the SFT criterion.

DG1 tool had only a basic set of actions such as Click, LongClick, Type Text, Type Number, Scroll, and Check. The text inputs were random generated without specified size. This version of the tool ran in five apps. The table 19 shows the code coverage values in five android apps. DG2 refers to actual tool with SFT.

With these results obtained, we identified the advantages such as higher code coverage, useful test cases with input types variation mainly when the requirements are informed. In the

Table 19 – Coverage comparison with Inputs based on SFT

	Instruction Coverage		Branch Coverage		Line Coverage		Method Coverage	
APPS	**DG1**	**DG2**	**DG1**	**DG2**	**DG1**	**DG2**	**DG1**	**DG2**
Atime Track	0,39	0,46	0,28	0,35	0,38	0,43	0,45	0,45
AddLoan	0,29	0,57	0,18	0,37	0,3	0,59	0,38	0,71
Simpledo	0,35	0,48	0,15	0,29	0,39	0,52	0,59	0,61
TimeTable	0,29	0,31	0,18	0,18	0,3	0,33	0,37	0,4
Budget Watch	0,41	0,46	0,22	0,25	0,42	0,47	0,53	0,59

5 applications observed the improvement of performance in terms of navigation with greater possibility of data entry.Issues related to execution time of actions in the application, slowness was not observed.

An important point is this approach is new in this context of test case generation for mobile applications; we noted that is important add variation of data entries efficiently generated to be useful to the testers.

5.6 Threats of Validity

The external threat of validity identified is the number of applications used for evaluation; to mitigate it, the applications chosen have different sizes and UI elements. Lack of functional requirements documentation of the apps is also an external threat. For this situation, we accessed the F-Droid store and checked the application description and, the basic functional operations in the application interface that made sense with the description were listed. Another external validity is the environment to run the experiments, a notebook with a GPU board, to avoid it, the tool was also tested in a notebook without a GPU board and the performance was not affected.

As internal threats, we identified the non-deterministic nature of this proposition. The coverage result can differ in each test cycle. To reduce it, the execution was repeated 3 times to confirm the resulting trend. This threat also applied to crashes and failures, we did the same methodology to measure crashes and failures for all testing tools, to make a fair comparison.

As the Construct Validity of our experiments, the confirmation of app failures and crashes, to certify it, we take the Log of each app executed during testing to verify the is indeed covered. DeepRLGUIMAT produces test event files. Hence we verify faults and crashes via replaying these tests manually.

5.7 Final Considerations

The empirical study carried out on 30 applications showed that the tool shows promising results for testing mobile applications compared to state-of-the-art tools Monkey, Droidbot, DroidbotX, and Q-testing. The tool revealed failures and crashes similar to the state of the art tools during the test execution. In terms of functional operations coverage, DeepRLGUIMAT achieved higher coverage than the state of the art tools. This approach produces test events with the SFT approach for input variation that exercises the application creating useful tests since the tool can insert a text or number too long, revealing flaws. The study case of DeepRLGUIMAT in MoneyTrack app with requirements showed that test cases that explore the input variation are generated according to SFT; the bias of the tool focused on the forms was also observed. More experiments with more apps with requirements are needed to evaluate the impact of the observed bias.

To evaluate the contribution of SFT to this approach, we compared the old version of the tool without SFT and the actual tool version with SFT in terms of code coverage; the results indicate the SFT promotes more exploration covering more paths of the apps. This result supports our strategy consolidating the importance of the input test data variation in test creation.

It is important to highlight that this tool inserts fewer commands per minute than the Monkey and Q-Testing tool; however, it was more effective due to the selection of actions for creating tests that better covered the apps and still with little repetition actions. The experiments carried out in the applications can also contribute to researchers.

In the next chapter, the conclusions and final considerations are discussed.

CHAPTER

6

CONCLUSION

6.1 Initial Considerations

Mobile applications are in our daily lives to execute various tasks such as communication, work, commerce and so on. The quality of these apps are important, and the public is less tolerant of failures. Testing mobile applications manually require effort and high cost due to all aspects of mobile context and models variation. The test automation for mobile app testing has been increasing due to the challenge of executing many possible interactions, input types, interface element changes, and fault reproduction. The ML technique Reinforcement Learning (RL) emerge as an opportunity to improve this area for test case generation through app exploration. However, the studies in this area showed that current solutions focus on navigation coverage and finding crashes.

This work offers a contribution in this area with the approach called DeepRLGUIMAT, which applies DQN to perform automated tests and generate test cases through exploration of the application under test covering a variety of input test data types using Systematic Functional Testing to satisfy the objective of covering functional operations, validate input data requirements and find failures. This chapter is organized as follows: Section 2 shows the main contributions of this research. Section 3 the limitations and future works, and section 4 the publication and the expectation of new publications.

6.2 Contributions

Considering the hypothesis that guided this work, the DeepRLGUIMAT was developed to use the Deep Reinforcement Learning technique to create and carry out black-box tests for mobile applications, using Systematic Functional Testing for input test data which will be available for use by the entire community.

This approach contemplates the objectives of providing a method that, through the DeepRLGUIMAT tool, performs the creation and execution of automatic tests efficiently that can insert input data variation, identify failures, promoting automatic tests, and cost savings.

As discussed in Chapter 3, Test generation initiatives using Reinforcement Learning and other machine learning methods do not address the use of test criteria or an oracle to guide the tests. Thus, the originality of this work is in the use of criteria of systematic functional testing to generate test input values and in the use of the requirements as previous knowledge in DQN to create the exploration of the application to cover more functionalities.

Thus, this work produced the following contributions:

- The development of a testing approach with RL and Systematic Functional Testing applied in mobile app testing and incorporated into the RL environment to vary test data inputs that allow the agent to generate more efficient tests. This approach helping to overcome the bottleneck of manual test design. A detail to be highlighted is that interface changes will not affect the tool's operation, as it explores the application without dependence on the id or GUI code. What typically happens in script-based test automation.

- The creation of a Reinforcement Learning Android environment with a variety of input actions following test criteria to provide functionality coverage. The RL environment which allows the agent to explore the android application space is important for the research of this method, in the software testing context, in addition to the environment enabling touch actions, it is necessary to offer different types of inputs so that the data entry requirements can be validated.

- The tool developed for the Android platform for proof of concept of the created approach is available to the research community. ($https://github.com/licollins/deepguit$)

- The empirical evaluation using the actual environment of android version 9 in a real device with performance compared to state-of-the-art tools random test generation Monkey, DroidBot RL tools DroidBotX and Q-Testing. (the experiments are also available in $https://github.com/licollins/deepguit$)

It is noteworthy that this tool can be instantiated to be used on other platforms as tests in hybrid applications, and the method can be applied to other mobile operating systems.

6.3 Limitations and Future Works

Due to the time constraints of this work, it was not possible to carry out a large-scale experiment with the use of the tool in software companies in the market to guarantee evidence of performance and efficiency. In chapter 5, we report the experiments performed in 30 applications

compared to the state of the art tools,which uses a similar Reinforcement Learning methodology. DeepRLGUIMAT outperformed the tools on code coverage metrics and outperformed on feature coverage. Certainly, more studies will be needed in groups of companies in the market that will allow the evolution of this tool.

The use of the DeepRLGUIMAT tool in academia should also be considered to assess the quality of the tests generated and transfer knowledge from tests performed to feedback the tool.

The DeepRLGUIMAT tool has some limitations that must be worked on to improve the usability and flexibility of your application. The following limitations are listed below:

- DeepRLGUIMAT does not yet have a graphical user interface for entering configuration data. The settings.txt file can lead to typos, compromising the user's experience with the tool.
- The DeepRLGUIMAT tool needs to generate a complete visualization of the results arranged in tables and graphs to indicate the features found in the application.
- Requirements data entry can also be enhanced through a graphical interface for the user to enter requirements in full. Using NLP (Natural Language Processing) concepts it may be possible to extract the requirements through text, making it easier for the user.
- It is necessary to improve the identification of exceptions or failure conditions of the apps interface which are not displayed in Android logcat.

As a future work, studies involving the improvement of DQN performance should be implemented for the feasibility of the tool for standalone execution. Another study to be conducted is the use of NLP to identify the context of application functionality to generate tests for business rules.

6.4 Publications

The partial results of this evaluation generate a paper published in Brazilian Symposium on Software Engineering 2021. The reference is listed below.

- Eliane Collins, Arilo Neto, Auri Vincenzi, and José Maldonado. 2021. Deep Reinforcement Learning based Android Application GUI Testing. Brazilian Symposium on Software Engineering. Association for Computing Machinery, New York, NY, USA, 186–194. DOI:https://doi.org/10.1145/3474624.3474634

This research was also described in articles for submission in the scientific journals, follow the papers submitted:

- Machine Learning based Mobile Application Testing: Systematic Mapping Study - Jornal Of Software Engineering Research and Development - submitted in 8 of January of 2022.
- Deep Reinforcement Learning Approach for Mobile Application Testing Guided by Input Entry Requirements - 2nd International Workshop on Artificial Intelligence in Software Testing - submitted in 27 of january of 2022.
- A Deep Reinforcement Learning-based GUI Mobile Application Testing Approach - Deep-RLGUIMAT - Journal Article Template for Software Testing, Verification and Reliability - to be submitted in 10 of February of 2022
- The Impact of Systematic Functional Testing input for Test Case Generation using Deep Q-Network Approach - Journal Article Template for Software Testing, Verification and Reliability - to be submitted in 25 of February of 2022

BIBLIOGRAPHY

829-2008 - IEEE Standard for Software and System Test Documentation - Redline | IEEE Standard | IEEE Xplore. <https://ieeexplore.ieee.org/document/5983353?denied=>. (Accessed on 06/20/2021). Citation on page 26.

ADAMO, D.; KHAN, M. K.; KOPPULA, S.; BRYCE, R. Reinforcement learning for android gui testing. In: **Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation**. New York, NY, USA: Association for Computing Machinery, 2018. (A-TEST 2018), p. 2–8. ISBN 9781450360531. Available: <https://doi.org/10.1145/3278186.3278187>. Citations on pages 20 and 92.

ALI, A. **Convolutional Neural Network(CNN) with Practical Implementation | by Amir Ali | Wavy AI Research Foundation | Medium**. 2019. <https://medium.com/machine-learning-researcher/convlutional-neural-network-cnn-2fc4faa7bb63>. (Accessed on 06/25/2021). Citations on pages 11 and 74.

ALPAYDıN, E. Machine learning. **WIREs Computational Statistics**, v. 3, n. 3, p. 195–203, 2011. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.166>. Citation on page 41.

ANACONDA. **Anaconda | Individual Edition**. 2021. <https://www.anaconda.com/products/individual>. (Accessed on 07/04/2021). Citation on page 79.

ANAND, S.; BURKE, E. K.; CHEN, T. Y.; CLARK, J.; COHEN, M. B.; GRIESKAMP, W.; HARMAN, M.; HARROLD, M. J.; MCMINN, P.; BERTOLINO, A.; Jenny Li, J.; ZHU, H. An orchestrated survey of methodologies for automated software test case generation. **Journal of Systems and Software**, v. 86, n. 8, p. 1978–2001, 2013. ISSN 0164-1212. Available: <https://www.sciencedirect.com/science/article/pii/S0164121213000563>. Citation on page 59.

ANDROID. **Android | The platform pushing what's possible**. 2021. <https://www.android.com/>. (Accessed on 06/20/2021). Citations on pages 11 and 33.

ARORA, I.; TETARWAL, V.; SAHA, A. Open issues in software defect prediction. **Procedia Computer Science**, v. 46, p. 906–912, 2015. ISSN 1877-0509. Proceedings of the International Conference on Information and Communication Technologies, ICICT 2014, 3-5 December 2014 at Bolgatty Palace Island Resort, Kochi, India. Available: <https://www.sciencedirect.com/science/article/pii/S1877050915002252>. Citation on page 60.

BADAMPUDI, D.; WOHLIN, C.; PETERSEN, K. Experiences from using snowballing and database searches in systematic literature studies. In: **Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2015. (EASE '15). ISBN 9781450333504. Available: <https://doi.org/10.1145/2745802.2745818>. Citation on page 48.

BASILI, V. R.; SELBY, R. W.; HUTCHENS, D. H. Experimentation in software engineering. **IEEE Transactions on Software Engineering**, SE-12, n. 7, p. 733–743, 1986. Citation on page 91.

CARBON, R.; CIOLKOWSKI, M. Evaluating open source software through prototyping. **Handbook of Research on Open Source Software**, p. 269–281, 2007. Citation on page 23.

CERVANTES, A. Exploring the use of a test automation framework. In: **2009 IEEE Aerospace conference**. [S.l.: s.n.], 2009. p. 1–9. Citation on page 31.

CHAPELLE, O.; SCHOLKOPF, B.; ZIEN, A. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. **IEEE Transactions on Neural Networks**, IEEE, v. 20, n. 3, p. 542–542, 2009. Citation on page 40.

DANA, R.; VAN, C. L. On the bellman equation of the overtaking criterion. **Journal of optimization theory and applications**, Springer, v. 67, n. 3, p. 587–600, 1990. Citation on page 75.

DELAMARO MARIO JINO, J. M. M. **ntrodução ao Teste de Software**. 2. ed. [S.l.]: Campus, 2016. ISBN 9788535283525. Citations on pages 20, 27, 28, and 32.

DEVELOPERS, A. **Desenvolvedores Android**. 2021. <https://developer.android.com/>. (Accessed on 06/20/2021). Citations on pages 32, 34, and 35.

ELBAUM, S.; ROTHERMEL, G.; KANDURI, S.; MALISHEVSKY, A. G. Selecting a cost-effective test case prioritization technique. **Software Quality Journal**, Springer, v. 12, n. 3, p. 185–210, 2004. Citation on page 61.

F-DROID. **F-Droid - Free and Open Source Android App Repository**. 2021. <https://f-droid.org/pt_BR/packages/>. (Accessed on 06/25/2021). Citation on page 92.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. *et al.* Inteligência artificial: Uma abordagem de aprendizado de máquina. **Rio de Janeiro: LTC**, v. 2, p. 192, 2011. Citation on page 38.

FAN, J.; WANG, Z.; XIE, Y.; YANG, Z. A theoretical analysis of deep q-learning. In: PMLR. **Learning for Dynamics and Control**. [S.l.], 2020. p. 486–489. Citation on page 42.

GAO, J.; BAI, X.; TSAI, W.; UEHARA, T. Mobile application testing: A tutorial. **ACM SIGPLAN/SIGSOFT Workshop on Program Analysis for Software Tools and Engineering**, IEEE Computer Society, v. 47, n. 2, p. 46–55, 2014. ISSN 0018-9162. Citations on pages 19, 20, 32, 34, and 35.

GONZÁLEZ, M.; BERGMEIR, C.; TRIGUERO, I.; RODRÍGUEZ, Y.; BENÍTEZ, J. M. Self-labeling techniques for semi-supervised time series classification: an empirical study. **Knowledge and Information Systems**, Springer, v. 55, n. 2, p. 493–528, 2018. Citation on page 40.

GOPINATH, R.; JENSEN, C.; GROCE, A. Code coverage for suite evaluation by developers. In: **Proceedings of the 36th International Conference on Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2014. (ICSE 2014), p. 72–82. ISBN 9781450327565. Available: <https://doi.org/10.1145/2568225.2568278>. Citation on page 90.

GYM. 2020. Available: <https://gym.openai.com/>. Citation on page 69.

HALLER, K. Mobile testing. **SIGSOFT Softw. Eng. Notes**, Association for Computing Machinery, New York, NY, USA, v. 38, n. 6, p. 1–8, Nov. 2013. ISSN 0163-5948. Available: <https://doi.org/10.1145/2532780.2532813>. Citation on page 20.

ISTQB. **CTFL Syllabus**. 2018. <https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html>. (Accessed on 06/20/2021). Citations on pages 29, 57, and 68.

JIMENEZ, W.; MAMMAR, A.; CAVALLI, A. Software vulnerabilities, prevention and detection methods: A review1. **Security in model-driven architecture**, Citeseer, v. 215995, p. 215995, 2009. Citation on page 60.

JOORABCHI, M. E.; MESBAH, A.; KRUCHTEN, P. Real challenges in mobile app development. In: IEEE. **2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement**. [S.l.], 2013. p. 15–24. Citation on page 37.

JUNIT.ORG. **JUnit 5**. 2021. <https://junit.org/junit5/>. (Accessed on 06/20/2021). Citation on page 36.

KHAMARU., S. **Code Coverage for Android using Jacoco - QA Learning Guide**. 2017. <http://www.qalearningguide.com/2017/10/code-coverage-for-android-using-jacoco.html>. (Accessed on 06/21/2021). Citation on page 92.

KIRUBAKARAN, B.; KARTHIKEYANI, V. Mobile application testing — challenges and solution approach through automation. **2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering**, p. 79–84, 2013. Citations on pages 21 and 35.

KONG, P.; LI, L.; GAO, J.; LIU, K.; BISSYANDE, T.; KLEIN, J. Automated testing of android apps: a systematic literature review. **IEEE Transactions on Reliability**, IEEE, Institute of Electrical and Electronics Engineers, v. 68, n. 1, p. 45–66, Mar. 2019. ISSN 0018-9529. Citations on pages 20 and 67.

Kong, P.; Li, L.; Gao, J.; Liu, K.; Bissyandé, T. F.; Klein, J. Automated testing of android apps: A systematic literature review. **IEEE Transactions on Reliability**, v. 68, n. 1, p. 45–66, 2019. Citations on pages 21 and 67.

KOROGLU, Y.; SEN, A. Functional test generation from ui test scenarios using reinforcement learning for android applications. **Software Testing, Verification and Reliability**, n/a, n. n/a, p. e1752. E1752 stvr.1752. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1752>. Citation on page 20.

KROPP, M.; MORALES, P. Automated gui testing on the android platform. **Testing Software and Systems**, p. 67, 2010. Citation on page 21.

KUMAR, D.; MISHRA, K. The impacts of test automation on software's cost, quality and time to market. **Procedia Computer Science**, v. 79, p. 8–15, 2016. ISSN 1877-0509. Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016. Available: <https://www.sciencedirect.com/science/article/pii/S1877050916001277>. Citations on pages 30 and 31.

LINKMAN, S.; VINCENZI, A. M. R.; MALDONADO, J. C. An evaluation of systematic functional testing using mutation testing. In: **7th International Conference on Empirical Assessment in Software Engineering –EASE**. [S.l.: s.n.], 2003. Citations on pages 22, 29, and 68.

MANN, H. B.; WHITNEY, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 18, n. 1, p. 50 – 60, 1947. Available: <https://doi.org/10.1214/aoms/1177730491>. Citation on page 95.

MAO, K.; HARMAN, M.; JIA, Y. Sapienz: Multi-objective automated testing for android applications. In: **Proceedings of the 25th International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2016. (ISSTA 2016), p. 94–105. ISBN 9781450343909. Available: <https://doi.org/10.1145/2931037.2931054>. Citation on page 90.

MARTIN, W.; SARRO, F.; JIA, Y.; ZHANG, Y.; HARMAN, M. A survey of app store analysis for software engineering. **IEEE transactions on software engineering**, IEEE, v. 43, n. 9, p. 817–847, 2016. Citation on page 32.

MASI, E.; CANTONE, G.; MASTROFINI, M.; CALAVARO, G.; SUBIACO, P. Mobile apps development: A framework for technology decision making. In: SPRINGER. **International Conference on Mobile Computing, Applications, and Services**. [S.l.], 2012. p. 64–79. Citation on page 32.

MCCABE IQ - Software Metrics Glossary. <http://www.mccabe.com/iq_research_metrics.htm>. (Accessed on 07/19/2021). Citation on page 58.

MEIRELES, S. R. A. *et al.* Evolução da ferramenta web guitar para geração automática de casos de teste de interface para aplicações web. Universidade Federal do Amazonas, 2015. Citation on page 30.

MELO, F. S. Convergence of q-learning: A simple proof. **Institute Of Systems and Robotics, Tech. Rep**, p. 1–4, 2001. Citation on page 41.

MITCHELL, T. M. **Machine Learning**. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Citations on pages 37, 38, and 40.

MYERS, G. The art of software testing. ny john wiley & sons. **Inc.–2004.–254 p**, 2004. Citation on page 25.

NAGAPPAN, M.; SHIHAB, E. Future trends in software engineering research for mobile apps. In: . [S.l.: s.n.], 2016. p. 21–32. Citation on page 35.

NUMPY. **What is NumPy? — NumPy v1.21 Manual**. 2021. <https://numpy.org/doc/stable/user/whatisnumpy.html>. (Accessed on 07/04/2021). Citation on page 79.

PAN, M.; HUANG, A.; WANG, G.; ZHANG, T.; LI, X. Reinforcement learning based curiosity-driven testing of android applications. In: **Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2020. (ISSTA 2020), p. 153–164. ISBN 9781450380089. Available: <https://doi.org/10.1145/3395363.3397354>. Citations on pages 20 and 90.

PANDAS. **pandas - Python Data Analysis Library**. 2021. <https://pandas.pydata.org/>. (Accessed on 07/04/2021). Citation on page 79.

PARSIFAL. **Parcifal**. [S.l.], 2020 (accessed June 3, 2020). <https://parsif.al/>. Citation on page 50.

PASZKE, A. **Reinforcement Learning (DQN) tutorial — PyTorch Tutorials 0.2.0_4 documentation**. 2017. Available: <http://seba1511.net/tutorials/intermediate/reinforcement_q_learning.html>. Citations on pages 21, 67, 75, and 79.

____. **Reinforcement Learning (DQN) tutorial — PyTorch Tutorials 0.2.0_4 documentation**. 2017. <http://seba1511.net/tutorials/intermediate/reinforcement_q_learning.html>. (Accessed on 07/01/2021). Citation on page 42.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: **Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering**. Swindon, GBR: BCS Learning amp; Development Ltd., 2008. (EASE'08), p. 68–77. Citations on pages 23, 45, and 46.

PUTERMAN, M. L. Chapter 8 markov decision processes. In: **Stochastic Models**. Elsevier, 1990, (Handbooks in Operations Research and Management Science, v. 2). p. 331 – 434. Available: <http://www.sciencedirect.com/science/article/pii/S0927050705801720>. Citation on page 41.

PYTORCH. **PyTorch**. 2021. <https://pytorch.org/>. (Accessed on 07/04/2021). Citation on page 78.

ROBOLECTRIC. 2021. <http://robolectric.org/>. (Accessed on 06/20/2021). Citation on page 36.

SAID, K. S.; NIE, L.; AJIBODE, A. A.; ZHOU, X. Gui testing for mobile applications: Objectives, approaches and challenges. In: **12th Asia-Pacific Symposium on Internetware**. New York, NY, USA: Association for Computing Machinery, 2020. (Internetware'20), p. 51–60. ISBN 9781450388191. Available: <https://doi.org/10.1145/3457913.3457931>. Citation on page 21.

SILVA, D. G. e; JINO, M.; ABREU, B. T. de. Machine learning methods and asymmetric cost function to estimate execution effort of software testing. In: IEEE. **2010 Third International Conference on Software Testing, Verification and Validation**. [S.l.], 2010. p. 275–284. Citation on page 32.

SMOLA A. J.; VISHWANATHAN, S. **Introduction to machine learning**. [S.l.], 2020 (accessed June 3, 2020). <http://alex.smola.org/drafts/thebook.pdf,/bib/smola/smola2008ml/thebook.pdf>. Citations on pages 11, 38, 39, and 40.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. [S.l.]: MIT press, 2018. Citations on pages 20, 21, and 41.

TAMADA, V. K. T. Estudo de caso de deep q-learning. Citations on pages 74 and 75.

TENSORFLOW. **TensorFlow**. 2021. <https://www.tensorflow.org/?hl=pt-br>. (Accessed on 07/04/2021). Citation on page 79.

UIAUTOMATOR. **GitHub - xiaocong/uiautomator: Python wrapper of Android uiautomator test tool.** 2021. <https://github.com/xiaocong/uiautomator>. (Accessed on 07/04/2021). Citation on page 79.

VEENENDAAL, E. V.; GRAHAM, D.; BLACK, R. "foundations of software testing: Istqb certification. **Cengage Learning EMEA**, p. 30, 2008. Citations on pages 11, 26, 27, 28, and 58.

VIDAL, A. R. **Biblioteca Digital de Teses e Dissertações: Teste funcional sistemático estendido: uma contribuição na aplicação de critérios de teste caixa-preta**. 2011. <https://repositorio.bc.ufg.br/tede/handle/tde/2887>. (Accessed on 01/04/2022). Citation on page 29.

VILKOMIR, S. A.; KAPOOR, K.; BOWEN, J. P. Tolerance of control-flow testing criteria. In: IEEE. **Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003**. [S.l.], 2003. p. 182–187. Citation on page 27.

WANG, W.; LI, D.; YANG, W.; CAO, Y.; ZHANG, Z.; DENG, Y.; XIE, T. An empirical study of android test generation tools in industrial cases. In: IEEE. **2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.], 2018. p. 738–748. Citation on page 21.

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 279–292, 1992. Citation on page 41.

WOHLIN, C.; HÖST, M.; HENNINGSSON, K. Empirical research methods in software engineering. In: **Empirical methods and studies in software engineering**. [S.l.]: Springer, 2003. p. 7–23. Citation on page 89.

WYSOPAL, C.; NELSON, L.; DUSTIN, E.; ZOVI, D. D. **The art of software security testing: identifying software security flaws**. [S.l.]: Pearson Education, 2006. Citation on page 57.

YASIN, H. N.; HAMID, S. H. A.; YUSOF, R. J. R. Droidbotx: Test case generation tool for android applications using q-learning. **Symmetry**, v. 13, n. 2, 2021. ISSN 2073-8994. Available: <https://www.mdpi.com/2073-8994/13/2/310>. Citation on page 90.

YOON, C. **Vanilla Deep Q Networks. Deep Q Learning Explained | by Chris Yoon | Towards Data Science**. 2019. <https://towardsdatascience.com/dqn-part-1-vanilla-deep-q-networks-6eb4a00febfb>. (Accessed on 06/25/2021). Citations on pages 11 and 75.

ANNEX

A

FUNCTIONAL OPERATION COVERAGE

Table 20 below represents the basic functional operations of 30 apps explained on Chapter 5 to evaluate the Functional coverage.

Table 20 – Functional Operations

APPS	Main Operations
AddLoan	Add Loan
	Add loan ->Add a person from contacts
	Add loan ->Add a person by name
	Add loan ->person added ->Add item
	Loan History ->change name
	Loan History ->delete from loaned
	Loan History ->link to contact
	Loan History ->history
	Loaned debug ->show by person, show by item, stats, settings
	Settings - date format, notifications
Atimetrack	Add task
	Enter in Report
	More - change date range
	More - export view
	More - Backup up to sd card
	More - Restore from backup
	More - Settings
	More - Help
	Edit Task
	Delete Task
	Show Task Times
	Task time - add new time range
	Task time - edit
	Task time - delete
	Task time - move time
Bettercount	Add counter ok
	counter added ->+1
	counter added ->undoing
	conter added - calendar
	counter added ->graph
	Add counter cancel

APPS	Main Functional Operations
Biever	Add Biever
	Edit Biever
	Import Biever
	Add Tastings
	Import tastings
	Add beer
	import beer
	export beer
	add beer style
	edit beer style
	delete beer style
	add breweries
	export/import
Bmi	calculate
	load
	save
	clear
Budget Watch	Add budget
	Edit budget
	Add Expense
	Edit Expense
	Add Revenue
	intro
	settings
	import/export
Catima	import/export
	settings
	add card
	edit card
	Add groups
Cavevin	new entry
	export database
	import database
	export to csv
	delete
	stats
	edit entry
	new cellar
	edit cellar
	choose cellar
	rename cellar
Dailypill	settings - time remember
	settings- theme
	about
	forgot
	remembered

APPS	Main Operations
Exceer	Training now
	Press Start
	Abort
	Press Pause/Continue
Farmerdiary	Write a note
	Choose date in calendar
	Search
	Add time to note
	Add events to note
	Add midia to note
	Edit style
	Edit script
	Backup
	Sharing
	Settings
Foodscale	Add
	Remove Last
	Clean
Grocy	Add shopping list
	Add product
	Add product not found
	Shop all items
	Edit notes
	Edit shop list
	Delete shop list
	Consume
	Edit product
	Delete product
	Edit places
	Delete places
	Edit stores
	Delete stores
	Edit unity
	Delete unity
	Edit group products
	Remove group products
	Settings
	Feedback
Justdo	Write a note
	Edit notes
	Delete a note
	Export note
	Import note
	Settings
	Help

APPS	Main Operations
Log28	Add info
	Set date on Calendar
	Access History
	Welcome
	Settings
Medclin	Privacy
	Save log
	Send log
	Delete log
	Set Preferences
	Access history
Meditation Assistant	Medit
	Pause medit
	Continue medit
	Verify Progress
	Set settings
	Access About
	Mindfullness bell
	See Tutorial
Money Track	Add recipe
	Edit recipe
	delete recipe
	add expense
	edit expense
	delete expense
	Add account
	edit account
	delete account
	transfer
	add taxes
	edit taxes
	delete taxes
	summary results
	results - graph
	Backup
	Import/Export
	Settings
music	library
	scan media
	equalizer
	settings
	search
	music
	new list

APPS	Main Operations
Openfood	Scan bar code
	Compare
	Start
	Access history
	Add a list
	Remove a list
	Write a bar code
	Category
	Complete products
	Set settings
	Edit Offline
	Search
Openworkout	Start training
	Add trainings
	Import trainings
	Edit training
	Delete training
	Set settings
	Add new section
	Edit section
	Delete section
	Access about
	Access help
Passgenerator	Generate pass
	Copy pass
	Reset fields
	Set app theme
	Access tips
	Set pass strengh
Periodical	Edit details
	Mark day
	Detail list
	Set backup
	Restore backup
	Set preferences
Silinote	Add new note
	Edit note
	Delete note
	Copy note
	Set settings
	Search
Simpledo	Create item
	Create item date time
	Delete item
	Edit item
	Quick reschedule

APPS	Main Operations
Textpad	Add new text
	Open a text
	Save text / Save as
	Search text
	Set settings
TimeTable	Add subject
	Edit subject
	Delete subject
	Access Summary
	Add exam
	Edit exam
	Delete exam
	Add teacher
	Delete teacher
	Add homework
	Edit homework
	Delete homework
	Add a note
	Edit note
	Delete note
	Set setings
	Backup
	Add profile
	Edit profile
	Delete profile
	Restore backup
Todo list	Add new list
	Add task
	Mark task
	Unmark task
	Mark all
	Unmark all
	Rename list
	Clear list
Tricky	Add traveler to trip
	Edit traveler trip
	Transfer money
	Create payment
	Deactivate (for costs)
	Show report
	Export tricky
	Set settings
Waterdroid	Add a message
	Edit message
	Set notifications


ICMC
USP
SÃO CARLOS