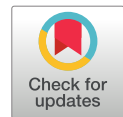




Journal of Data Analytics and Artificial Intelligence Applications

Review Article

 Open Access

Machine Learning Implementation in Automated Software Testing: A Review



Normi Sham Awang Abu Bakar¹  

¹ International Islamic University Malaysia, Department of Computer Science, Kulliyah of ICT, Kuala Lumpur, Malaysia

Abstract

The integration of Machine Learning (ML) in automated software testing represents a transformative approach aimed at enhancing the efficiency, accuracy, and scope of testing processes. This paper explores the theoretical and practical aspects of employing ML techniques within the realm of software testing, focusing on key areas such as test case generation, defect prediction, and test suite optimisation. Through a comprehensive literature review and case studies, this study illustrates the potential benefits associated with ML-driven testing methodologies. The findings indicate that ML can significantly reduce manual intervention and improve defect detection rates, thereby facilitating more reliable software delivery. This paper also addresses the benefits of ML implementation in automated testing and future research directions to bridge existing gaps and further leverage ML in software testing.

Keywords

Artificial intelligence · automated testing · software testing activities · machine learning algorithm



Citation: Normi Sham Awang Abu Bakar. 2025. Machine Learning Implementation in Automated Software Testing: A Review. *Journal of Data Analytics and Artificial Intelligence Applications* 1, 1 (January 2025), 110-122. <https://doi.org/10.26650/d3ai.001>

 This work is licensed under Creative Commons Attribution-NonCommercial 4.0 International License. 

© 2025. Abu Bakar, N. S.

 Corresponding author: Normi Sham Awang Abu Bakar nsham@iium.edu.my

1. INTRODUCTION

A fundamental component of the software development lifecycle has always been software testing, or ST. However, software has grown in size and complexity as it has become more widely used [1], posing new difficulties for software testing procedures [2]. Consequently, there is interest in examining how artificial intelligence (AI) has been applied to enhance testing procedures, since AI can improve knowledge work. The interaction between AI and ST has been the subject of numerous studies [3]. However, because each of these fields is so vast and complex, excellent review studies typically concentrate their attention on orthogonal choices within each of these fields.

The main goal of this paper is to explore the machine learning implementation in the automated software testing context. In this study, the main focus is on the use of the machine learning algorithms to make the automated testing more efficient, which will assist the software testers to focus on test executions, rather than on test planning and design.

To achieve this goal, 34 papers were reviewed for their relevancy in both the ST and AI areas, which discuss the AI-driven methodologies and tools to improve the efficiencies of the automated software testing activities. In particular, the machine learning techniques are also explored to add more depth to the understanding of the most frequently used techniques to support automated software testing.

As such, two research questions are developed for this study:

RQ1: What are the machine learning techniques frequently used to support the automated software testing activities?

RQ2: How are the machine learning techniques being implemented in the automated software testing activities?

The remainder of the article is organised as follows. Section 2 introduces the background and the prior related works in this study, Section 3 describes the implementation of machine learning in automated testing, Section 4 highlights the advantages of using AI in ST, and Section 5 concludes the findings of the paper and discusses the future work.

2. BACKGROUND

Current research directions in Software Engineering automation could be perfectly complemented by recent developments in generative AI. Specifically, generative AI naturally pairs well with automated test data generation. Despite the generative AI approach's potential to produce highly human-readable, domain- and context-aware solutions, its propensity for hallucinations makes it somewhat unreliable when used alone. Nevertheless, automated test data generation can eliminate these delusional features of AI-based solutions while also adding the essential assurances.

There are important implications regarding the recent findings that generative AI models can exhibit robust emergent behaviours [4], [5]. Their behaviour is therefore both powerful and inherently difficult to understand. Because the emergent behaviour of the models cannot be cross-checked against a ground truth, it may be problematic in applications lacking a ground truth, such as general inquiries about arbitrary facts about reality. However, for software engineering tasks like code enhancement and testing, we have an extremely reliable ground truth: the execution of the improved code or the recommended test.

2.1. Artificial Intelligence

Despite the fact that there are numerous definitions of AI, the definition given in [6] is used for the purposes of this investigation: “AI is a generic term that refers to any machine or algorithm that is capable of observing its environment, learning, and based on the knowledge and experience gained, taking intelligent action or proposing decisions. There are many technologies that fall under this broad AI definition. At the moment, ML techniques are the most widely used.”

The AI domain dealing with the ability of systems to automatically learn, decide, predict, adapt and react to changes and improve from experience, without being explicitly programmed, is the learning domain [7]. According to the AI Watch report, there are five core scientific domains:

2.1.1. Reasoning

The field of artificial intelligence studies methods for turning data into knowledge and drawing conclusions from it. Knowledge representation, automated reasoning, and common sense reasoning are the three sub-domains that make up this domain.

2.1.2. Planning

The area of artificial intelligence that focuses on creating and implementing strategies for performing tasks, usually carried out by unmanned vehicles, intelligent agents, and autonomous robots. In this field, strategies are distinguished by intricate solutions that need to be found and refined in a multidimensional environment. This domain consists of three closely related sub-domains: searching, optimisation and planning and scheduling. The optimisation of the search for solutions to scheduling and planning issues is the focus of these sub-domains.

2.1.3. Learning

The branch of artificial intelligence that deals with a system’s natural capacity to learn, make decisions, forecast outcomes, adjust to changes, and grow through experience—all without the need for explicit programming. Machine learning (ML)-related concepts are primarily used in the construction of the corresponding branch of the resulting taxonomy.

2.1.4. Communication

The field of artificial intelligence deals with the recognition, processing, comprehension, and creation of data from spoken and written human communication. The field of natural language processing (NLP) primarily deals with this domain [5].

2.1.5. Perception

This field indicates a system’s capacity to perceive its surroundings through its hearing and vision, such as computer vision.

2.2. Machine Learning Techniques

Machine learning (ML) is the science of getting computers to learn and act like humans do. It uses algorithms and mathematical models to progressively improve their performance on a specific task [8]. In essence, machine learning (ML) is the process of identifying patterns in data and using that knowledge to solve problems with regression or classification. The representation of the data that the machine learning algorithms are given has a major impact on how well they perform. In fact, machine learning algorithms “learn”



how to accomplish certain tasks through a training phase using training datasets, which are representative sample data [1].

Machine learning can handle unsupervised learning problems (like clustering or dimensionality reduction) where no ground truth is provided, as well as supervised learning problems (like classification and regression) where training sets are annotated (or labelled) with the ground truth values. Reinforcement learning (RL) algorithms are based on a feedback-directed mechanism that allows them to continuously adapt to their operating environment. To maximise an expected cumulative reward function, the algorithm makes a decision, considers the effects of that decision and then modifies its approach.

Among the main ML methods that are related to ST are [7]:

- (i) artificial neural networks (ANN), a group of supervised algorithms that are modelled after biological neural networks discovered in animal brains [8]. It is necessary to observe the input and expected output data and establish the probability-weighted associations between the two to train a neural network. The network's data structure, which is composed of layers of connected perceptions, then stores these associations [9].
- (ii) boosting is a group meta-algorithm for minimising the components of bias and variance error [10],
- (iii) classification, a supervised task that includes the process of training a model on a population of instances labelled with a discrete set of labels yields a set of predicted labels for a given collection of unobserved instances [11].
- (iv) clustering, given a similarity function for an unsupervised task, objects are grouped into clusters based on how much more similar they are to one another than they are to objects in other clusters [12].
- (v) convolutional neural networks (CNN), a particular neural network where at least one layer substitutes convolution for general matrix multiplication [13].
- (vi) decision trees, a family of classification and regression algorithms that learn the hierarchical structures of fundamental decision rules from the data. The resulting models can be visualised as trees, where nodes represent decision rules and leaf nodes represent outcomes [14], [15].
- (vii) probabilistic models, a family of classifiers that can forecast a probability distribution across a range of classes given an observation of an input [16], [17].
- (viii) reinforcement learning, the algorithms address the “problem faced by an agent that must learn behaviour through trial-and-error interactions with a dynamic environment” and one of the core paradigms of machine learning [18].
- (ix) regression, with a set of mathematical techniques, data scientists can forecast a continuous outcome based on the value of one or more predictor variables [19].
- (x) supervised learning, a paradigm for machine learning when the available data is limited to labelled examples [20].
- (xi) support vector machines (SVM), supervised learning algorithms that, after the input features are non-linearly mapped to a very high-dimension feature space, build a linear decision surface to generate models for classification and regression analysis [21].
- (xii) unsupervised learning, a basic machine learning paradigm in which computers attempt to identify patterns in unlabelled data [20].



The machine learning techniques discussed previously are summarised in Figure 1.



Figure 1. Machine Learning Techniques

2.3. Software Testing

Software Testing is defined by the 29119-1-2013 ISO/IEC/IEEE International Standard as: “A process made by a set of interrelated or interacting activities aimed at providing two types of confirmations: verification and validation” [22]. Validation proves that the work item can be used by users for their particular tasks, while verification verifies that a given software product (work item or test item) satisfies the specified requirements.

In this study, the ST domain that will be investigated is the Testing Activities. This ST domain describes the tasks that testing teams and testers can complete into precise, controlled processes. To guarantee that the test objectives are satisfied in an economical manner, these activities range from test planning to test output evaluation.

Among the testing activities identified in this study are:

- (i) Test Case Generation whose goal is to create executable test cases according to the specific testing methods and the amount of testing that needs to be done.
- (ii) Test Planning is a fundamental activity of the ST process; it encompasses staff coordination, test equipment and facility availability, test-related documentation creation and upkeep, and scheduling of additional testing activities.
- (iii) Test Results Evaluation is performed to determine if the testing was carried out successfully. “Successful” usually refers to the software operating as anticipated and producing no significant unexpected results. Unexpected results aren’t always bad; occasionally, they turn out to be noise. An analysis and debugging effort is required to isolate, identify, and describe a fault before it can be fixed.

- (iv) Test Execution symbolises both running test cases and documenting the outcomes of those runs. A fundamental tenet of scientific experimentation should be applied when conducting tests: all procedures should be carried out and recorded in a way that makes it possible for another individual to repeat the findings.
- (v) Test Oracle Definition is the process carried out to assist in the creation of test oracles or to generate them automatically.
- (vi) Test Case Design and Specification is carried out to define or design the testing cases. The requirement analysis of the system being tested is typically the first step in this process.
- (vii) Test Case Optimization/ Prioritisation/Selection is carried out to select, prioritise, and reduce test cases for execution in an optimal manner [23].
- (viii) Test Data Definition (test data generation) is the process that generates the test case data [13].

2.4. Automated Testing

Writing a programme in any programming or scripting language that uses an external automation helper tool to replicate the manual test case steps is known as software testing automation. It entails developing toolkits for testing the implemented source code. Its objective is to increase the automation of the testing procedures. The tasks associated with development are programme development and test script writing; the former relates to the application itself, while the latter is concerned with the scripts that will be utilised to test the application [23].

Software test automation is defined by Dustin et al. as “management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool”. In theory, test automation should be seen as a more all-encompassing concept that includes other tasks in addition to automated test scripting and execution during the software testing process.

Because testing is a repetitive process and it is advised to test every scenario, automation is crucial. Test automation will boost productivity and expand test coverage. Automated testing allows for the testing of different input values, conditions, and repeated execution of the tests. There will be a reduction in the testing time and resources. There are many tools available to automate acceptance, system, and functional tests. Watir, JMeter, and Selenium are a few of them.

3. MACHINE LEARNING TECHNIQUES IN AUTOMATED SOFTWARE TESTING

The fundamentals of AI testing are based on the idea of “automatic abstraction of application and test logic” [24]. Intelligent learning agents, which are capable of autonomously perceiving and responding to their surroundings, can help achieve this. By investigating the functionality and understanding the operation of the application, they can plan and develop the test cases on the target system. Ultimately, they can run the tests and analyse the test findings. The agents can operate at various levels of hierarchy and are arranged with other agents.

To answer both RQ1 and RQ2, 34 papers have been reviewed and the main keywords for the paper search are ("AI" OR "artificial intelligence" OR "ML" OR "machine learning") AND ((test* AND (automated OR automation)) AND (“software engineering” OR “software”).



Test frameworks for artificial intelligence (AI) can be broadly applied to test both cross-domain and multiple applications within a domain. A requirement for the test "library of the common user flows" is that the AI test framework be designed to function with both cross-domain applications and multiple applications within the domain as a general framework. The actions and the elements are interconnected in this library. The AI agent can query the database to find the test cases for a particular type of element when it sees one in an application that it can interact with.

This section focuses on the results of the paper review on the topic under study, which is the implementation of the machine learning techniques in automated software testing. From the articles found on the topic, the machine learning techniques are divided into several categories, as shown in Table 1. In addition, they were mapped into the relevant software testing activities, as reported in the articles.

Table 1 depicts the mapping between the machine learning techniques in the software testing activities, where the corresponding articles highlighted in the table reported the implementation of various techniques of machine learning in software testing. Specifically, artificial neural networks have been applied to various testing tasks, including oracle definition, test-case generation, test-case refinement, and test-case evaluation; studies [25] and [26] covered these tasks and found that machine learning algorithms resulted in predicted output oracles, metamorphic relations, and test verdicts. Nearly all research uses a supervised or semi-supervised methodology, training models (e.g., neural networks, support vector machines, adaptive boosting, and decision trees) on labelled system executions or code metadata.

Table 1. Machine learning techniques in software testing

	Artificial neural network	Boosting	Classification	Clustering	Convolutional neural network	Decision trees	Probabilistic model	Reinforcement learning	Regression	Supervised learning	Support vector machine	Unsupervised learning
Test Case Generation	[24]					[24]	[24]				[24]	
Test Planning				[5]			[25]					
Test Oracle Definition	[5] [24] [26]	[26]	[25]		[24]	[25]		[25]	[26]	[25] [26]	[24]	
Test Case Design and Specification				[25]			[25]	[25]		[25]		
Test Case Optimisation/ Prioritization/ Selection	[28]	[28]	[25][27] [28]	[25] [27] [28]		[25] [28]	[24][29]	[25][27] [28]	[27] [28]	[25] [28]	[28]	[25][28]
Test Results Evaluation	[5]			[25]			[25]		[25]			[25]
Test Data Definition	[26]				[24]			[24]	[26]	[26]	[26]	
Test Execution			[24]	[24]				[24]				

Furthermore, Garousi et al. [5] found that, when compared to test oracles created using current conventional methods, those created using artificial neural networks for the Test Oracle Definition activity are more effective, efficient, and reusable. Furthermore, the primary benefits of utilising machine learning and artificial neural networks were noted by Durelli et al. [25] as being their scalability and low requirement for human intervention. According to Durelli et al. [25] and Fontes & Gay [26], the primary challenge encountered by researchers attempting to use artificial neural networks and machine learning to address software testing

issues is the requirement for a significant quantity of high-quality training data, which is essential for machine learning algorithms to perform as intended.

Khatibsyarbini et al. [27] also claimed that based on the publication trend of ML technique applied to Test Case Prioritisation, the most popular ML technique category was classification, followed by clustering and reinforcement learning as the least preferred ML technique category. Additionally, they stated that the most popular machine learning technique is classification because it uses historical data and yields high average percentages of faults detected and effective code coverage. They also emphasised that reinforcement learning needs to be improved and given more structure before it can be taught in undergraduate programmes.

According to Pan et al. [28], Reinforcement learning, clustering, and classification AI approaches have been widely used for test case optimisation, prioritisation and selection. According to their report, reinforcement learning, unsupervised learning (clustering), and supervised learning (ranking models) are the three main machine learning techniques used for test case prioritisation and selection. Any machine learning method that depends on ranking or classification models is called supervised learning.

In addition, the methods that use reinforcement learning to rank test cases based on their length, past performance, and failure history have also been reported by Durelli et al. [25]. Furthermore, Pan et al. highlighted that although supervised learning, unsupervised learning, reinforcement learning, and natural learning processing are the four main machine learning (ML) techniques used for test case selection and prioritisation, various combinations of these techniques have also been reported in the literature. To improve the test case prioritisation performance, supervised or unsupervised learning was integrated with NLP-based techniques, which are frequently used for feature preprocessing. They also emphasised how difficult it is to draw trustworthy conclusions about the effectiveness of ML-based test case selection and prioritisation due to the absence of appropriate publicly available datasets and standard evaluation processes that are derived from the execution of real-world case studies.

4. THE IMPACT OF AI IN ST

The field of artificial intelligence for software testing, or AIST, is a young one that aims to create AI tools for software testing, test methods for AI systems, and create software that can self-test and/or self-heal. The process of manually encoding a predetermined set of programme input actions and output verification steps into a script that can be run by a machine is commonly referred to as “test automation” in software testing [10]. A log of the results is created, saved, and linked to the run after it is executed. The test execution and logging are the only parts of this process that are automated. To properly test software, human testers must set testing objectives, gain the knowledge required, create and specify comprehensive test scenarios, write test automation scripts, perform scenarios that cannot be automated, and evaluate test results to identify potential project risks.

Researchers and practitioners have begun looking into how AI and ML can be used to create the next generation of testing tools, since the majority of testing is currently focused on manual testing and the manual writing of test scripts [3]. The idea is to use big data, cloud computing, and AI/ML advancements to bridge the gap between human-present and machine-driven testing.

AI-driven testing has several benefits, including being robust, scalable, adaptable, reusable, and all-purpose. Machine learning techniques can be used to solve several issues. For instance, practically any mathematical

function can be approximated using a straightforward feed-forward neural network with a single hidden layer [20]. Consequently, various testing types, applications, and domains can benefit from the use of AI-driven testing.

Because AI-driven tests are typically not tied to any particular application, they can be applied to different applications within the same domain (like add item to cart) or to different domains (like login). By creating new tests every time, the pesticide paradox—a narrow scope of fault detection brought on by repeatedly running the same tests—can be avoided. Finally, accelerated test coverage is a key advantage of AI-driven testing, which is achieved by fusing large-scale test execution in the cloud with AI-based test generation [3].

Despite the promising results, the implementation of ML in automated software testing is not without challenges. The quality and quantity of data, the interpretability of ML models, and the integration of ML tools with existing testing frameworks are critical factors that need to be addressed. Moreover, the continuous evolution of software systems necessitates ongoing adaptation and learning, which poses additional challenges for ML-based testing solutions.

4.1. Case Study of AI implementation in ST

4.1.1. One of the AI technique implementations in ST is the usage of Reinforcement Learning for Test Case Optimisation. The details are

Objective: Optimise test case selection and prioritisation based on failure history and test execution performance. Furthermore, to prioritise and optimise test cases in regression testing by learning from historical test execution data, focusing on factors such as failure history, execution cost, and risk.

4.1.2. Implementation:

- Reinforcement learning algorithms were used to rank the test cases.
- Factors such as past failures, execution cost, and risk level were considered for prioritisation.

4.1.3. Outcomes:

- Higher fault detection rates.
- Reduced testing efforts and costs by focusing on critical test cases first.
- Example Application: Used in regression testing scenarios where frequent updates require selective testing.

4.1.4. Implementation Steps

4.1.4.1. Problem Formulation

The task is modelled as a reinforcement learning problem:

- State (S): Represents the attributes of the test case, such as the historical success rate, execution cost, and risk factor.
- Action (A): Decide whether to execute or skip a test case.
- Reward (R): A numeric value based on the detection of critical defects and cost savings (e.g., 1 for a defect found, -1 for skipping a necessary test).

4.1.4.2. Dataset

The input data includes historical test case executions:

- Features: Test case ID, previous pass/fail outcomes, execution time, code coverage metrics, and defect severity.
- Labels: Whether to execute (1) or skip (0) the test case.

4.1.4.3. Results

- Output: The list of test cases prioritised for execution based on their predicted effectiveness.
- Benefits:
 - Ensures that critical test cases with higher defect detection probability are executed first.
 - Reduces the unnecessary execution of low-priority test cases, saving time and resources.

5. CONCLUSION AND FUTURE WORK

The integration of machine learning (ML) into automated software testing has shown significant potential in enhancing the efficiency and effectiveness of the software development lifecycle. This paper explored various ML techniques and their applications in different phases of software testing, including test case generation, test suite optimisation, defect prediction, and automated test script maintenance. By leveraging ML algorithms, software testing processes can be more adaptive and intelligent, leading to improved detection of defects, reduced testing time, and optimised resource allocation.

The reviews presented in this paper demonstrate the feasibility and advantages of using ML in automated software testing. Specifically, the use of supervised learning for defect prediction and clustering algorithms for test case prioritisation has proven to be effective in identifying high-risk areas of the software and optimising testing efforts. In addition, reinforcement learning techniques have shown promise in automating the generation and maintenance of test scripts, reducing the manual effort required and enhancing test coverage.

To further advance the field of ML in automated software testing, several areas warrant further research and development:

1. **Data Quality and Availability:** Ensuring high-quality and diverse datasets is crucial for training robust ML models. Future research should focus on developing methods for generating synthetic test data, handling imbalanced datasets, and improving data preprocessing techniques.
2. **Model Interpretability and Explainability:** As ML models become more complex, their interpretability becomes a significant concern. Future work should aim at developing techniques that provide insights into the decision-making process of ML models, enabling testers to understand and trust the predictions and recommendations made by these models.
3. **Integration with DevOps Practises:** Integrating ML-based testing solutions with modern DevOps practises can enhance continuous integration and continuous deployment (CI/CD) pipelines. Research should explore ways to seamlessly incorporate ML algorithms into these pipelines, ensuring that testing processes remain agile and responsive to changes in the software.
4. **Scalability and Performance Optimisation:** As software systems grow in complexity, the scalability of ML-based testing solutions becomes critical. Future research should investigate ways to optimise the performance of ML algorithms, ensuring that they can handle large-scale software projects efficiently.



5. **Cross-Project Learning and Transfer Learning:** Leveraging knowledge from previous projects can enhance the performance of ML models in new projects. Future work should explore transfer learning techniques and cross-project learning approaches to make ML models more generalisable and applicable across different software domains.
6. **Human-AI Collaboration:** The collaboration between human testers and ML models can lead to more effective testing strategies. Research should focus on developing interactive tools that facilitate this collaboration, allowing testers to leverage the strengths of both human expertise and ML capabilities.

The suggestions for overcoming data quality challenges and enhancing model interpretability are given below:

5.1. Overcoming Data Quality Challenges

5.1.1. Ensuring High-Quality Data

5.1.1.1. Data Preprocessing

- Remove noise and irrelevant features through normalisation, scaling, and feature selection techniques.
- Detect and handle outliers using methods like Isolation Forest or Z-score analysis.

5.1.1.2. Imbalanced Data Handling

- Use techniques like Synthetic Minority Oversampling Technique (SMOTE) to balance datasets when defect-prone areas are underrepresented.
- Employ cost-sensitive learning to penalise misclassifications of critical data.

5.1.1.3. Data Augmentation

- Generate synthetic data to compensate for the limited datasets.
- Use domain-specific methods like mutation testing to create diverse test cases.

5.1.1.4. Data Cleaning

- Automate error detection in datasets (e.g., duplicate entries, missing labels).
- Verify correctness through manual reviews of critical entries.

5.2. Enhancing the Model Interpretability

5.2.1. Explainable AI (XAI) Techniques

5.2.1.1. Local Interpretability

- Use tools such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) to explain individual predictions.
- Visualise feature importance to highlight critical attributes influencing test case selection or defect predictions.

5.2.1.2. Global Interpretability


- Employ decision tree models or surrogate models to approximate complex models like neural networks for easier understanding.
- Provide feature summary plots to show the overall trends in the model's decision-making process.



Peer Review	Externally peer-reviewed.
Conflict of Interest	The author has no conflict of interest to declare.
Grant Support	The author declared that this study has received no financial support.

Author Details **Normi Sham Awang Abu Bakar**

¹ International Islamic University Malaysia, Department of Computer Science, Kulliyah of ICT, Kuala Lumpur, Malaysia

 0000-0002-8069-3323

References

- [1] Gerhard Lakemeyer and Bernhard Nebel. 1994. Foundations of Knowledge Representation and Reasoning. Springer, Berlin, 1–12. https://doi.org/10.1007/3-540-58107-3_1
- [2] Santiago Matalonga, Domenico Amalfitano, Andrea Doreste, Anna Rita Fasolino, and Guilherme Horta Travassos. 2022. Alternatives for testing of context-aware software systems in non-academic settings: Results from a rapid review. *Info. Softw. Technol.* 149 (2022), 106937. <https://doi.org/10.1016/j.infsof.2022.106937>
- [3] Tariq M. King, Jason Arbon, Dionny Santiago, David Adamo, Wendy Chin, and Ram Shanmugam. 2019. AI for testing today and tomorrow: Industry perspectives. In Proceedings of the IEEE International Conference On Artificial Intelligence Testing (AITest'19). IEEE, 81–88. <https://doi.org/10.1109/AITest.2019.000-3>
- [4] P. Paygude and S. D. Joshi. 2020. Use of evolutionary algorithm in regression test case prioritization: A review. In Proceeding of the International Conference on Computer Networks, Big Data and IoT (ICCB'18). Lecture Notes on Data Engineering and Communications Technologies, A. Pandian, T. Senjyu, S. Islam, and H.Wang (Eds.). Vol. 31, Springer, Cham, 56–66. https://doi.org/10.1007/978-3-030-24643-3_6
- [5] Vahid Garousi, Sara Bauer, and Michael Felderer. 2020. NLP-assisted software testing: A systematic mapping of the literature. *Info. Softw. Technol.* 126 (2020), 106321. <https://doi.org/10.1016/j.infsof.2020.106321>
- [6] M. Craglia, A. Annoni, P. Benczur, P. Bertoldi, B. Delipetrev, G. De Prato, C. Feijoo, E. Fernandez Macias, E. Gomez Gutierrez, M. Iglesias Portela, H. Junklewitz, M. Lopez Cobo, B. Martens, S. Figueiredo Do Nascimento, S. Nativi, A. Polvora, J. I. Sanchez Martin, S. Tolan, I. Tuomi, and L. Vesnic Alujevic. 2018. Artificial Intelligence: A European Perspective. Technical Report KJ-NA-29425-EN-N. Luxembourg. <https://doi.org/10.2760/11251>
- [7] Domenico Amalfitano, Stefano Faralli, Jean Carlo Rossa Hauck, Santiago Matalonga, and Damiano Distanto. 2023. Artificial Intelligence Applied to Software Testing: A Tertiary Study. *ACM Comput. Surv.* 56, 3, Article 5 (October 2023), 38 pages. <https://doi.org/10.1145/3616372>
- [8] J. J. Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A.* 79, 8 (Apr. 1982), 2554–2558. <https://doi.org/10.1073/pnas.79.8.2554>
- [9] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. 1994. The basic ideas in neural networks. *Commun. ACM* 37, 3 (Mar. 1994), 87–92. <https://doi.org/10.1145/175247.175256>
- [10] Leo Breiman. 2000. Bias, Variance, and Arcing Classifiers. Technical Report 460, Statistics Department, University of California.
- [11] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas. 2006. Machine learning: A review of classification and combining techniques. *Artific. Intell. Rev.* 26, 3 (Nov. 2006), 159–190. <https://doi.org/10.1007/s10462-007-9052-3>
- [12] Dongkuan Xu and Yingjie Tian. 2015. A comprehensive survey of clustering algorithms. *Ann. Data Sci.* 2, 2 (2015), 165–193. <https://doi.org/10.1007/s40745-015-0040-1>
- [13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press. Retrieved from <http://www.deeplearningbook.org>
- [14] Bernard M. E. Moret. 1982. Decision trees and diagrams. *ACM Comput. Surv.* 14, 4 (Dec. 1982), 593–623. <https://doi.org/10.1145/356893.356898>



- [15] D. Opitz and R. Maclin. 1999. Popular ensemble methods: An empirical study. *J. Artific. Intell. Res.* 11 (Aug. 1999), 169–198. <https://doi.org/10.1613/jair.614>
- [16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. Springer, New York. <https://doi.org/10.1007/978-0-387-84858-7>
- [17] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* 5, 2 (1994), 157–166. <https://doi.org/10.1109/72.279181>
- [18] L. P. Kaelbling, M. L. Littman, and A.W. Moore. 1996. Reinforcement learning: A survey. *J. Artific. Intell. Res.* 4 (1996), 237–285. <https://doi.org/10.1613/jair.301>
- [19] G. Udny Yule. 1897. On the theory of correlation. *J. Roy. Stat. Soc.* 60, 4 (1897), 812–854. <https://doi.org/10.1111/j.2397-2335.1897.tb02784.x>
- [20] Stuart Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach*. Pearson. Retrieved from <https://books.google.it/books?id=XS9CjwEACAAJ>
- [21] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.* 20, 3 (Sep. 1995), 273–297. <https://doi.org/10.1007/BF00994018>
- [22] International Organization for Standardization. 2013. ISO/IEC/IEEE international standard—software and systems engineering—software testing—Part 1: Concepts and definitions. ISO/IEC/IEEE 29119-1:2013(E) (2013), 64. <https://doi.org/10.1109/IEEESTD.2013.658853>
- [23] George Candea, Stefan Bucur, and Cristian Zamfir. 2010. Automated software testing as a service. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC '10)*. Association for Computing Machinery, New York, NY, USA, 155–160. <https://doi.org/10.1145/1807128.1807153>
- [24] Anna Trudova, Michal Dolezel, and Alena Buchalceva. 2020. Artificial intelligence in software test automation: A systematic literature review. In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'20)*. INSTICC, SciTePress, 181–192. <https://doi.org/10.5220/0009417801810192>
- [25] Vinicius H. S. Durelli, Rafael S. Durelli, Simone S. Borges, Andre T. Endo, Marcelo M. Eler, Diego R. C. Dias, and Marcelo P. Guimar.es. 2019. Machine learning applied to software testing: A systematic mapping study. *IEEE Trans. Reliabil.* 68, 3 (2019), 1189–1212. <https://doi.org/10.1109/TR.2019.2892517>
- [26] Afonso Fontes and Gregory Gay. 2021. Using machine learning to generate test oracles: A systematic literature review. In *Proceedings of the 1st International Workshop on Test Oracles (TORACLE'21)*. ACM, New York, NY, 1–10. <https://doi.org/10.1145/3472675.3473974>
- [27] Muhammad Khatibsyarbini, Mohd Adham Isa, Dayang N. A. Jawawi, Muhammad Luqman Mohd Shafie, Wan Mohd Nasir Wan-Kadir, Haza Nuzly Abdul Hamed, and Muhammad Dhiauddin Mohamed Suffian. 2021. Trend application of machine learning in test case prioritization: A review on techniques. *IEEE Access* 9 (2021), 166262–166282. <https://doi.org/10.1109/ACCESS.2021.3135508>
- [28] Rongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, and Lionel Briand. 2021. Test case selection and prioritization using machine learning: A systematic literature review. *Empir. Softw. Eng.* 27, 2 (Dec. 2021), 29. <https://doi.org/10.1007/s10664-021-10066-6>
- [29] Gerson Barbosa, Erica Ferreira de Souza, Luciana Brasil Rebelo dos Santos, Marlon da Silva, Juliana Marino Balera, and Nandamudi Lankalapalli Vijaykumar. 2022. A systematic literature review on prioritizing software test cases using Markov chains. *Info. Softw. Technol.* 147 (2022), 106902. <https://doi.org/10.1016/j.infsof.2022.106902>

