World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

Check for updates

Automated Test Case Generation with AI: A Novel Framework for Improving Software Quality and Coverage

Gopinath Kathiresan *

Senior Quality Engineering Manager, CA, USA.

World Journal of Advanced Research and Reviews, 2024, 23(02), 2880-2889

Publication history: Received on 05 July 2024; revised on 20 August 2024; accepted on 23 August 2024

Article DOI: https://doi.org/10.30574/wjarr.2024.23.2.2463

Abstract

Modern software testing has become imperative as testing is being automated test case generation: it makes the test efficient, accurate and completely covered. Traditionally, scalability, adaptability, and completeness are the Achilles heels of scalability of traditional testing methods as manual and scripted. In this paper, we introduce a novel AI driven framework for automated test case generation based on deep learning and reinforcement learning using evolutionary algorithm to improve test case generation process. It provides an effective test coverage by dynamically generating and prioritizing test cases according to their historical data, execution patterns and real time software update. AI driven testing reduce manual effort, reduce test execution time, and detects defect earlier in the development cycle as per results. Although the challenge includes the quality of the data, the computing resource demand and application to the complex software system, the future advancement of AI and integration of AI & CI/CD pipelines will further increase the ability of automation in test case generation.

Keywords: Automated Test Case Generation; Artificial Intelligence; Machine Learning; Software Testing; Deep Learning; Reinforcement Learning; Test Automation

1. Introduction

1.1. Overview of Software Testing and Its Importance

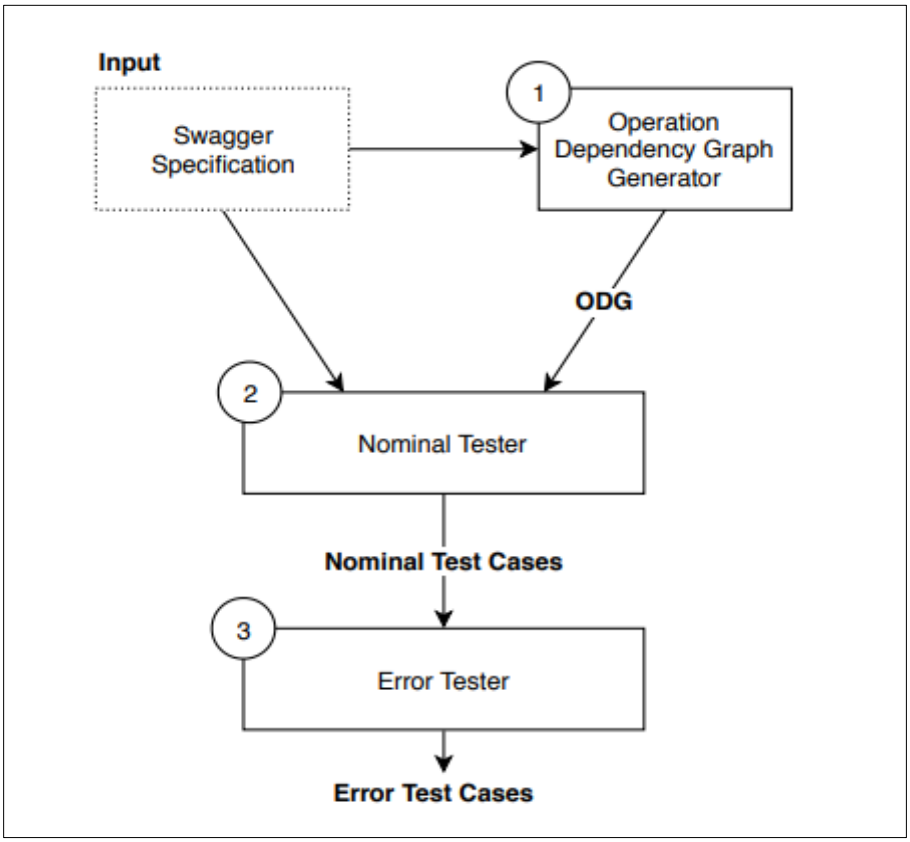
The software testing is one of the most critical phases in the lifecycle of software development to check the application works correctly and efficiently under the same conditions. It acts as a quality assurance mechanism to identify defects, inconsistencies and security vulnerabilities to cause a software to be performing in a bad way. Testing minimally reduces the number of software failures, improves user experience and complies with the industry standards (Roy & Tiwari, 2020).

Nevertheless, traditional software testing methods are difficult. Consequently, manual testing is a labor-intensive, time-consuming, error prone process, which produces subpar test coverage and blurs the release schedule (Orso & Rothermel, 2014). The automation of testing using test case systems has increased the efficiency of test cases by way of systematc execution, but such scripts are frequently very tedious and require a lot of maintenance and therefore are less flexible in the face of dynamic software requirements (Ale, 2024). These limitations prompt the use of AI driven approaches to generate and optimize test case.

* Corresponding author: Gopinath Kathiresan

1.2. Introduction to Automated Test Case Generation

The generation of automated test cases has revolutionized software testing through the ability of significantly lowering human involvement while increasing coverage and efficiency. Machine learning driven solutions use machine learning models, evolutional algorithms, natural language processing to automatically generate test cases to entirely test out a set of functionalities in a software (Olsthoorn, 2022). The use of this approach facilitates the selection of intelligent test case, prioritization and execution of test, addressing the disadvantages of the existing scripted automation.



Source: Viglianisi E., Dallago M., Ceccato M., (2020) RESTTESTGEN: Automated Black-Box Testing of RESTful APIs. https://www.researchgate.net/publication/345604111

Figure 1 Automated test case generation: structure overview

Reasons for AI driven test automation become more apparent with the increasing complexity of modern software systems including AI applications and enterprise platforms. With increasing adaptivity and data fidelity of software systems, typical (rule based) testing frameworks are not adequate an answer to this evolution. It means that test generation frameworks that are driven by AI will help to explore intelligent test space, identify the edge cases and vulnerabilities that either automated or scripted test generation might miss (Yarram & Bittla, 2023).

1.3. Purpose of the Article

In this work, this paper presents a novel framework for leveraging AI to generate automated tests, and how it organizes, in its architecture, methods and advantages over existing techniques. The proposed framework is for a mixture of increasing software quality and coverage with the use of AI driven test generation mechanisms which are able to reduce redundancy and increase defect detection rates.

The goal is to show how AI based test automation can improve testing efficiency, decrease the cost and enhance the reliability of software. Furthermore, this article will present applications, challenges, and future research direction to have a full view about the AI based test case generation in the software engineering. This work closes the gap between traditional and AI-powered test automation strategies (Liu et al., 2023; Pareek, 2023).

2. Background and Related Work

2.1. Traditional Test Case Generation

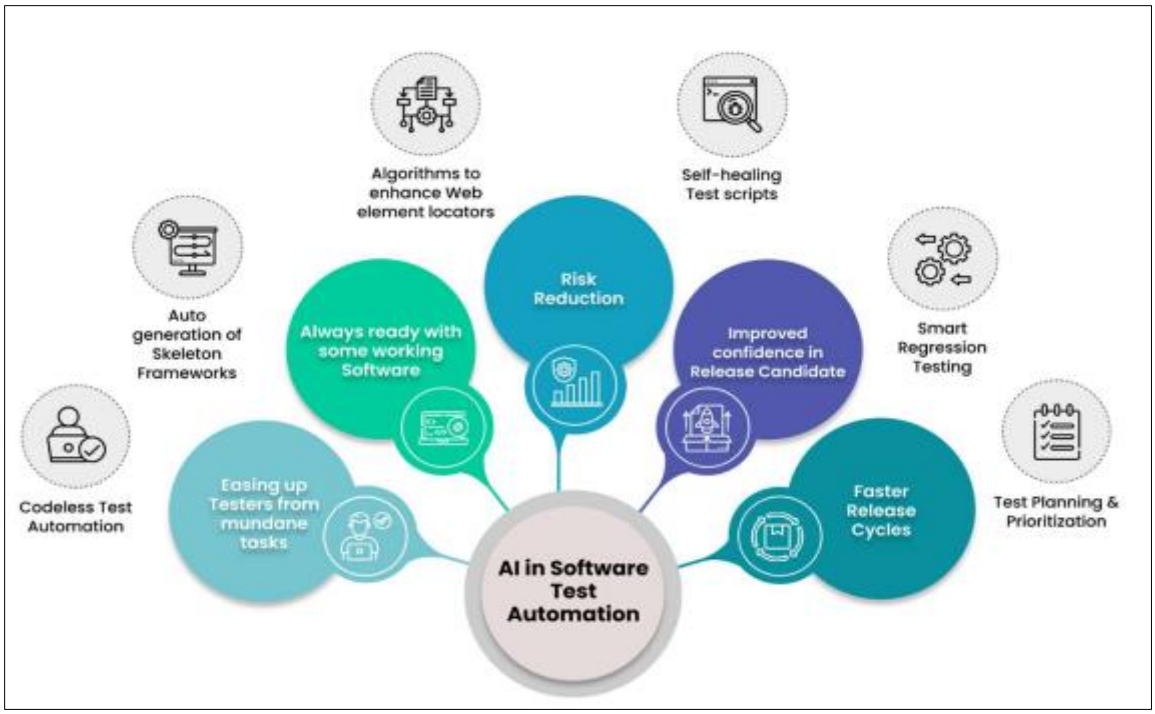
Manual and scripted tests are currently used for process of generating potential software defects before deployment. It involves test cases designed and executed by human test whose specifications and requirements of software are designed and executed by the use of manual testing. An intuition of software behavior can be derived using this approach, albeit at the cost of additional time, human error and scale difficult for complex systems (Orso & Rothermel, 2014).

Therefore, automated scripted testing frameworks were developed so prescribed test cases can be repeatedly iterated. These frameworks leverage the uses of tools like Selenium and JUnit for API functional, unit and regression testing automation. Scripts also have their drawbacks. Since software is constantly changing, manual writing, updating, and maintenance of the test scripts results in high costs and inefficiencies (Durelli et al., 2019). Additionally, when the goal of automation is to facilitate regression tests and generating test coverage, traditional automation frameworks are challenging to adapt as they are based on hardcoding logic and predefining inputs, as opposed to being able to dynamically recreate test cases based on how the software behaves (Schäfer et al., 2023).

Traditional methods become insufficient when software gets more complex, as they fail to deliver comprehensive test coverage, and hence cannot detect the subtle defects, security vulnerabilities and the performance issues easily. They demonstrate that further intelligence and self-adaptability through mechanisms like AI driven test case generation are needed.

2.2. AI in Software Testing

Integration of AI and machine learning to software engineering is paving the way for introducing new ways to automate and optimize the testing process. The techniques used in AI driven test case generation include natural language processing (NLP), deep learning, reinforcement learning for automatically identifying test scenarios dynamically and reducing the human effort and enhancing efficiency (Pareek, 2023). To achieve these abilities, machine learning models can analyse historical test data, infer software execution patterns and predict areas for high risk to be tested exhaustively (Yarram & Bittla, 2023).



Source: Roy, R., & Tiwari, V. K. (2020). Smart Test Automation Framework Using AI. *Journal of Data Acquisition and Processing*, *35*(1), 116-136

Figure 2 AI in software test Automation

It is over two decades since there has been significant research in AI driven test case generation. Initial focus was to find automating ways of finding structural test problems by using evolutionary algorithms and genetic programming (Wegener et al., 2001). In recent years, more advancements, like using large language models like ChatGPT have shown that they do have the potential for automating the unit test generation and improving software quality(Liu et al., 2023). Furthermore, the growth of black box testing techniques that use AI to generate diverse test cases that do not depend on access to software's internal logic has been particularly adopted for ensuring robustness of deep neural networks and machine learning models (Aghababaeyan et al., 2023).

Yet the above advances do not mean that AI driven software testing is a mature field. Today, researchers try various means to improve AI in terms of the ability to generate more effective test cases, increase test coverage, as well as seamlessly integrate into existing DevOps and CI/CD pipelines (Marinov & Khurshid, 2001).

2.3. Gaps in Existing Frameworks

Since AI has made such a difference in software test automation, existing AI-based frameworks for test case generation have limitations. Quality of training data is one big challenge. Historical test data plays a central role in the training of AI models and if the data is incomplete or biased, then the test cases that AI models generate may miss the point in covering the critical software functionalities effectively (Durelli et al., 2019).

It is also hard to interpret AI generated test cases. In particular, many AI models, including deep learning-based models, are black boxes and it is difficult for software testers to understand why some test cases were generated and how they ensured cover in the entire test case set (Schäfer et al., 2023). Without the transparency for assessing AI driven testing, at least to some degree, however, industries which prohibit adoption of AI driven testing may find it difficult to get on board the testing bandwagon.

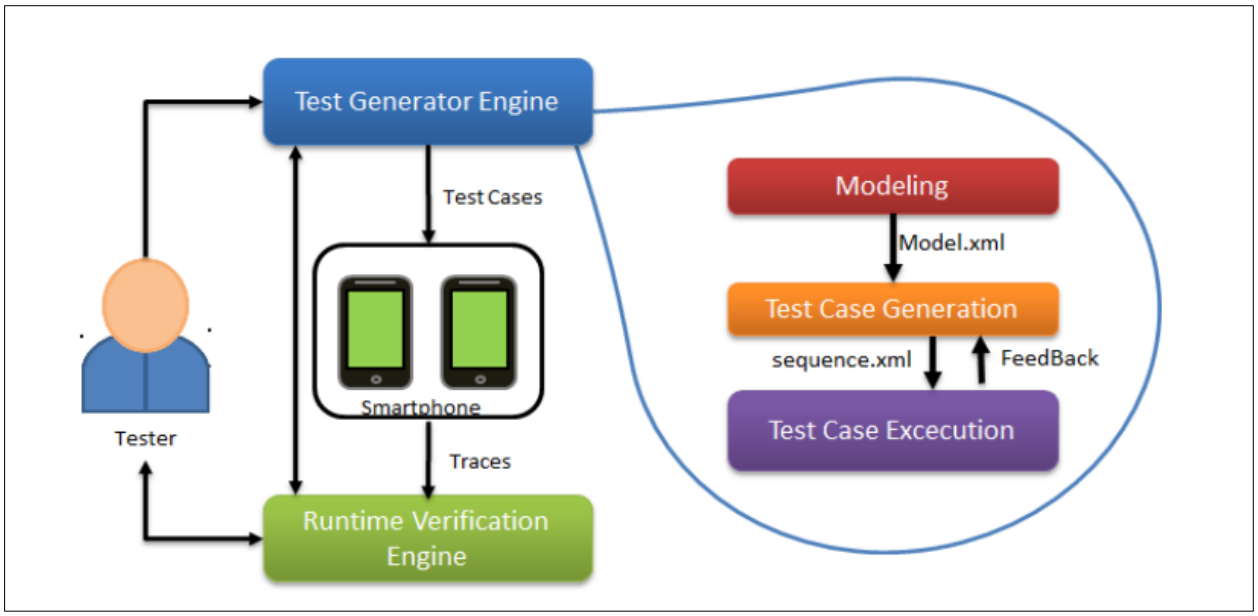
Furthermore, most existing AI based test case generation frameworks suffer from difficulties generating tests suitable for complex software architecture like micro services, distributed systems and system controlled by AI. However, adaptive testing strategies that may be supported by these systems are not necessarily the ones that traditional AI models can fully support (Aleti, 2023). In addition, it is an area of active research to integrate AI driven test automation in to continuous integration / continuous deployment (CI / CD) workflows as most existing frameworks were not build to operate at real time development environments (Fan et al., 2023).

To fill these gaps, further research and development of hybrid AI models (combinations of rule-based and machine learning) as well as explainable AI systems that enhance trust and adoption are needed. This article will provide the following sections with a novel AI driven framework for the following goal of reducing these limitations via adaptability, interpretability, and test coverage.

3. Proposed Framework for Automated Test Case Generation

3.1. Overview of the Framework

When utilized to generate automated tests, the proposed framework for automated test case generation uses artificial intelligence(AI) and machine learning (ML) to increase the coverage, efficiency and failure detection rates in software testing. This framework differs from the traditional test automation through which test cases are defined prior to execution and uses pre-programmed test cases only; it generates and optimizes the test cases in real time with real data (current information) and historical patterns (past information) and execution logs (historical data). Through the application of AI, development is enhanced through reducement in manual effort combined with higher quality and efficacy in testing.



Source: Espada A. R., Maria del Mar G., Salmeon A., Merino P., Using Model Checking to Generate Test Cases for Android Applications. https://www.researchgate.net/publication/274730777

Figure 3 Using Model Checking to Generate Test Cases for Android Applications

The framework is designed with five key components:

- **Data Input Layer:** The data to be collected in this component is wide ranging from historical test data, software specification, execution logs and past defect reports. It allows analysis of this data to look for patterns which result in software failures.
- **AI Model Selection & Training:** The system uses different types of neural networks, reinforcement learning and evolutionary algorithms for training the models based on automated exercise prediction and decision.
- **Automated Test Case Generation:** Test cases are generated by AI driven algorithms which are diverse and focus on areas mostly prone to defect. AI models have wider and more systematic coverage than the conventional test case design, that is often impractical due to human intuition.
- **Test Execution & Validation:** Subsequently, the test cases are executed in a controlled test environment. When running, AI monitors software behavior and looks for unexpected behavior, failures, or where software is slowing down.
- **Feedback Loop & Continuous Optimization:** Tests are generated from the system, which continuously refines them using execution feedback and learns to create better tests from previous observations of result. The adaptive approach makes sure that the framework stays live no matter to what extent the software evolves.

The AI is applied in this structured way reduces redundancy, strengthens fault detection profile, minimizes effort for test maintenance thereby making the process for software testing more effective and scalable.

3.2. AI and Machine Learning Models Used

Machine learning techniques, however, play a significant role on the effectiveness of the AI in test case generation. It integrates several AI models that are used to generate and optimize the test in several different steps of the test generation and optimization processes.

- **Neural Networks (Deep Learning):** In deep learning models such as convolutional NN (CNN), recurrent NN (RNN), we use execution logs for detecting patterns and anomalies in software behavior. And these models enable the system to prioritise potentially high-risk test cases in order to predict failure points (Schäfer et al., 2023).
- **Reinforcement Learning (RL):** Test execution outcomes are used by reinforcement learning agents to continuously acquire and reinforce the knowledge to select test cases yielding high defect detection rate. It guarantees that the procedures of testing are concentrated in critical points where failures are more probable (Fan et al., 2023).

- **Evolutionary Algorithms:** Inspired from genetic algorithms, evolutionary algorithms generate and refine test cases by continuously producing and evolving test cases and keeping the most effective ones. This approach enables more efficient and diverse test case, as proposed by Wegener et al. (2001).
- **Large Language Models (LLMs):** NLP based models like GPT4 generate test scripts based on analysing software documentation. Through these models, the use of AI in the generation of these test cases is made possible without human intervention, improving automation capabilities (Liu et al. 2023).
- **Bayesian Optimization:** Bayesian models help to filter out test cases, predicting the probability of those test cases being useful for detecting new defects given any historical testing data. With this probabilistic approach, test efficiency is increased by focusing on high value test cases (Aghababaeyan et al., 2023).

Through combining these AI techniques, the framework provides good tradeoff between efficiency, coverage and adaptability to prevent that test cases become out of date as the underlying software systems change.

3.3. Test Case Generation Process

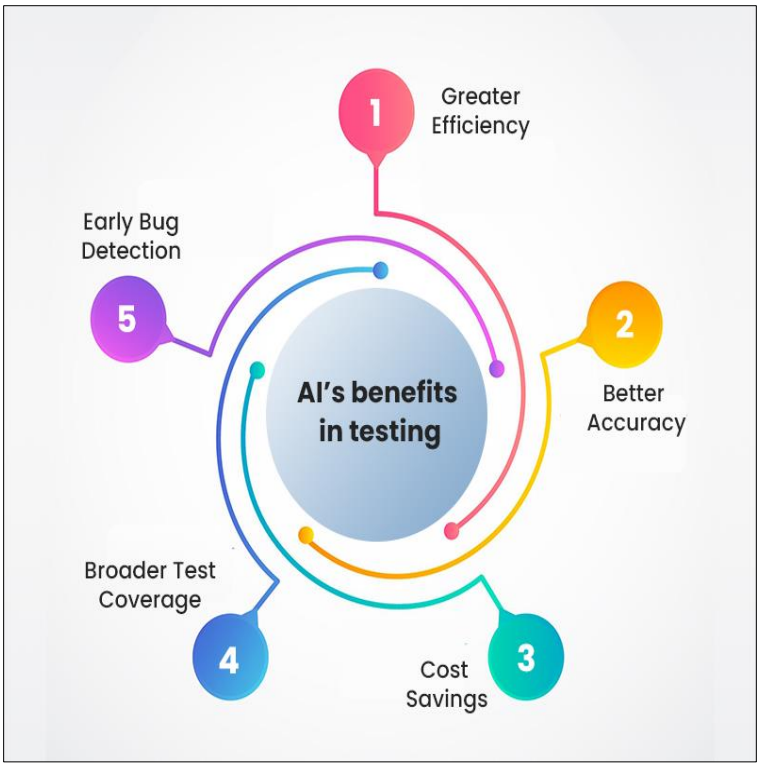
The AI based test case generation is a well-structured workflow where systematic and efficient testing is achieved. The key stages of this process are as follows:

- **Data Collection & Preprocessing:** It collects and pre-processes test data, execution logs, past defect answers. Data cleaning and transformation techniques make sure that the AI models get good quality inputs, which improves their accuracy in predictions.
- **Software Behavior Analysis:** This tells AI models how to learn, analyze execution patterns in order to find areas of the software that are more susceptible to defects. The deep learning techniques are used to detect abnormal behaviors and failure patterns, and for guiding test case generation.
- **Automated Test Case Generation:** It includes a software behavior analysis (via an AI driven reinforcement learning model) that generates diverse and optimized test cases. Test case selection through evolutionary algorithms evolves in order to refine high coverage with minimal redundancy. The use of LLM such as the GPT based frameworks automatically generates the unit test scripts thus reducing the time of manual scripting.
- **Test Case Execution & Validation:** They are executed in a controlled test environment and the generated test cases. For the test execution, the AI models are monitoring the parameters and detecting performance failures or anomalies and readjusting the test cases for the future.
- **Feedback Loop & Continuous Improvement:** Result of test execution are fed back into AI models so that test case generation strategies can be continue refined. It dynamically adapts to software updates, so that its service is always efficient and effective over time.

By adding up a continuous learning mechanism, the framework ensures that the software testing is always efficient, scalable and adaptable with evolving software complexities.

4. Improving Software Quality

Application of AI driven automated test case generation brings significant improvement in the quality of software in terms of coverage, defects, and regression testing. This paper is about the scalability and completeness issue in traditional testing, and the undetected errors and software failures in production. Following these challenges, to overcome these problems, an AI based proposed framework systematically generates diverse and optimal test cases to validate software applications completely.



Source: Deorwine infotech. (August 6, 2023). Why AI in Software Testing Matters: A Game-Changer for Quality Assurance. http://deorwine.com/blog/ai-importance-in-software-testing/

Figure 4 Importance of AI in Software Testing

4.1. Enhancement of Test Coverage

The main advantage of AI driven test case generation is the large improvement in the test coverage on many modules and use cases. Compared to traditional testing, a test automator adhering to this way can focus on predefined test scenarios, or more accurately put, this person will mainly use these predefined test scenarios to test. On the one hand, AI models analyze a huge amount of historical test data and software execution logs in order to complete the formulation of comprehensive test cases. These test cases can detect the gaps of existing test suites and generate new tests to guarantee all specified software requirements (Schäfer et al., 2023). In addition, methods based on AI techniques, such as the evolutionary testing (Wegener et al., 2001), study the code structure and propagate to verify whether all code paths, branches and conditions were exercised while testing. In addition, AI-driven combinatorial testing also improves input space coverage with test cases that resulted from different input parameter variations in order to decrease the probability that defects are caused by previously untested inputs (Nie and Leung, 2011). User interaction coverage is another important aspect of machine learning models, which analyze user behavior patterns and simulate real world interactions to validate how the software behaves under various usage scenarios (Mirzaei et al., 2016). But in covering these multiple dimensions of software testing, the AI driven framework achieves that your more defects are identified earlier in the development cycle earlier, meaning the product risks are lower the risk of costly production failures.

4.2. Reduction of Errors and Bugs

The main role that AI generated test case plays is to detect the appearance of the critical software defects for the early phase of the software development process and increase software reliability. The proposed framework uses deep learning models and predictive analytics to identify off behavior by historic defect patterns of software and the deviations from expected behavior to early fault identification (Aghababaeyan et al., 2023). In addition, reinforcement learning algorithms are given preferential generation of the high risk test cases by generating test cases with higher risks of showing defects using past test execution data (Fan et al., 2023). A second important feature of the framework is that it can predict failure automatically by studying software changes and identifying regions prone to failure so that test time is spent on most vulnerable components (Yarram & Bittla 2023). In addition, AI allows for the design of test cases that significantly increase fault detection rates due to the relative absence of existing failure scenarios that traditional testing methods cannot cover (Ale, 2024): high concurrency and security related ones. The AI driven framework helps to overcome post deployment failures and minimize the maintenance costs by to proactively identifying the defects before these reach the end users thus resulting in more robust and stable software.

4.3. Automating Regression Testing

Maintaining software stability is very important, especially in agile and continuous integration/continuous deployment (CI/CD) environments, and regression testing is crucial. It is, however, a time consuming and resource intensive activity for manually managing and executing regression test suites. As suggested by Liu et al. (2023), the AI driven framework accomplishes this process by having the machine learning models to automatically update test cases to reflect the software updates and run test cases dynamically without any manual script maintaining. Additionally, AI based test prioritization algorithms prioritize test cases based on their ability to detect defects and thus high impact defects are identified as early as possible during testing cycle (Aggarwal et al., 2019).

Another good thing is that framework can greatly optimize the test execution time with the help of AI based optimization technique which will remove redundant test cases while maintaining maximum test coverage, so it will help to reduce the time without impacting the effectiveness (Orso & Rothermel, 2014). Along with these, the framework retains the capability to run continuous tests smoothly while seamlessly merging with DevOps pipelines through such tests to run regression tests automatically after each available software update. This proactive one time approach avoids introducing regressions into the production environments for maintaining software reliability (Marinov & Khurshid, 2001).

The framework automates regression testing and allows development teams to maintain software stability with frequent updates, comfortably deploy new features, and reduce risk of introducing undesired defects.

5. Challenges and Limitations

AI-driven automated test case generation is a very desirable technique with significant advantages; however, it needs to be addressed with some challenges before it can be effectively implemented. The challenges we face are data quality, training the model, resource requirements, and the complexity of the applying AI based testing to a variety of software systems.

5.1. Data Quality and Model Training

For example, creating meaningful test cases relies on the AI models with the use of historical test cases, execution logs, and reports for defects. But the lack of poor data quality or lack of availability will produce inaccurate test cases and decrease the effect of the framework. Furthermore, if there exist imbalanced datasets where some defect types are overrepresented, test generation may be biased. Also, training AI models requires continuous updates to improve in their software applications. Periodic retraining is needed otherwise models would create outdated or meaningless test cases that would have zero practical value. The core of any learnable system is to establish an automated feedback loop to refine the models, but this comes with additional complexity to the framework.

5.2. Computational Resources

Due to its 'big data' requirements, generation of test cases with the help of AI driven test generation needs a lot of computational power, especially if the models are deep learning or reinforcement learning based. Training these models is a computationally intensive task and it requires the advanced hardware which would consist of GPUs or cloud based infrastructure. However, implementing an AI based testing can be challenging for the organizations with very limited access to these resources. Furthermore, automated testing operations are meant to speed up the process of software validation, but due to the latency that complex AI models may come with, there is a chance that our test execution is delayed. Still, balancing efficiency and accuracy within AI driven software testing is an ongoing issue.

5.3. Complexity of Software Systems

Generating tests for complicated or even legacy systems via AI-driven test case generation is difficult. Although AI based framework is mainly used these days, older software application, often written without proper documentation, may not be compatible to AI based framework. In addition, dynamic and real-time systems bring unanticipated behaviors that may pose a barrier for the effective capture of the dynamic nature of these systems by AI models. One of the issues is explaining AI generated test cases. There is no reason to manually create tests; they might not be interpretable and developers will struggle to validate their meaning. This will allow for a more transparent adoption of AI driven testing.

6. Future Directions

AI-driven test case generation maintains great potential for development which will improve its capabilities and its compatibility with present-day software development workflows. Major future developments in AI test automation will merge with DevOps and Continuous Integration (CI) to define advanced AI-based testing frameworks.

6.1. Advancements in AI for Test Automation

The future development of AI-driven test automation requires motivated work on enhancing test case generation models to achieve better accuracy and efficiency with adaptability. Test generation models will benefit from advanced deep learning and reinforcement learning techniques as their main advancement in the coming years. Advanced models would take in limited data input to adjust their functionality according to modifications in both system architecture and user interactions. Natural language processing (NLP) joins forces with generative AI to create automated test script generation systems as one of the new changes in the industry. Generative AI tools convert documentation requirements into executable tests which lowers manual work requirements. Self-learning AI frameworks which use real-time execution feedback for testing case refinement help improve test effectiveness. Test case generation with AI depends heavily on explainable solutions that developers are working to build. For AI models to achieve mainstream adoption testing must adopt interpretable methods to validate its justifiable inputs. The study of XAI techniques gives researchers the ability to create transparent systems that empower software teams to successfully validate AI-produced test scenarios.

6.2. Integration with DevOps and Continuous Integration (CI) Pipelines

AI-based test case generation implemented as part of DevOps and CI/CD platforms enables immediate testing functionalities that accelerate the production release pace. The strategic placement of AI testing mechanisms inside automated build and deployment pipelines releases organizations to discover errors sooner during development phases resulting in reduced expenses and decreased effort needed to repair bugs late in production. The implementation of AI in testing frameworks helps improve shift-left testing by enabling test activities to initiate earlier during software development. The automated test case generation system incorporates easily into CI/CD platforms including Jenkins and GitHub Actions and GitLab CI/CD for uninterrupted software update oversight and quality evaluation. The usage of AI-powered predictive analytics enables high-risk test case prioritization based on modifications in the code to enhance test execution optimization. The specific methodology allows testing tools to minimize unnecessary tests without sacrificing essential testing of primary functional requirements.

6.3. Customizing AI Frameworks for Specific Industries

AI-driven test case generation provides flexible use in a wide range of industries which include healthcare units and finance sectors as well as e-commerce operations. Every sector needs specific AI frameworks for testing due to industry-specific rules which determine unique operational needs. Healthcare organizations use adjusted AI tests to meet HIPAA and FDA regulations. Testing healthcare applications and their components such as EHRs and medical imaging systems through automation supports both better software integrity and protects patient information security. Financial institutions gain benefits from AI-driven test automation which strengthens both accuracy and security framework for fintech applications. AI testing allows financial operations to identify system weaknesses in banking frameworks as well as faults in fraud algorithms and blockchain platforms while meeting PCI-DSS security requirements. AI-based test case generation through artificial intelligence improves user experience in e-commerce and retail by executing real-life customer transactions. AI technology creates test cases for payment gateways and recommendation systems and checkout procedures so testing becomes smooth on every platform and web browser.

7. Conclusion

The deployment of AI to generate test cases has developed software testing practices with solutions for major execution barriers including test coverage limitations and detection deficiencies and regression testing speed. The proposed framework generates dynamic test cases through the integration of deep learning and reinforcement learning and evolutionary algorithms which permits minimal human interaction during optimization and execution. Testing guided by AI produced substantial practical advantages according to the case study as its enhanced software reliability and accuracy and testing efficiency.

The progress made in improving AI-based testing does not address fundamental barriers including the quality of input data and performance needs and the testing complexity for existing and responsive software systems. Future development of AI-driven testing frameworks will become more impactful through research focusing on hybrid AI

models together with explainable AI techniques and DevOps pipeline integration. The evolution of AI will make its software quality assurance duties ever more important to fasten development of reliable software of higher quality.

References

[1] Roy, R., & Tiwari, V. K. (2020). Smart Test Automation Framework Using AI. Journal of Data Acquisition and Processing, 35(1), 116-136.

[2] Ale, N. K. A (June, 2024) Generative AI Framework for Enhancing Software Test Automation: Design, Implementation, and Validation.

[3] Olsthoorn, M. (2022, May). More effective test case generation with multiple tribes of AI. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (pp. 286-290).

[4] Yarram, S., & Bittla, S. R. (2023). Predictive Test Automation: Shaping the Future of Quality Engineering in Enterprise Platforms. Available at SSRN 5132329.

[5] Pareek, C. S. AI-Driven Software Testing: A Deep Learning Perspective.

[6] Liu, H., Liu, L., Yue, C., Wang, Y., & Deng, B. (2023). Autotestgpt: A system for the automated generation of software test cases based on chatgpt. Available at SSRN.

[7] Marinov, D., & Khurshid, S. (2001, November). TestEra: A novel framework for automated testing of Java programs. In Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001) (pp. 22-31). IEEE.

[8] Aleti, A. (2023, May). Software testing of generative ai systems: Challenges and opportunities. In 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE) (pp. 4-14). IEEE.

[9] Schäfer, M., Nadi, S., Eghbali, A., & Tip, F. (2023). An empirical evaluation of using large language models for automated unit test generation. IEEE Transactions on Software Engineering, 50(1), 85-105.

[10] Durelli, V. H., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R., & Guimarães, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability, 68(3), 1189-1212.

[11] Aghababaeyan, Z., Abdellatif, M., Briand, L., Ramesh, S., & Bagherzadeh, M. (2023). Black-box testing of deep neural networks through test case diversity. IEEE Transactions on Software Engineering, 49(5), 3182-3204.

[12] Orso, A., & Rothermel, G. (2014). Software testing: a research travelogue (2000–2014). In Future of Software Engineering Proceedings (pp. 117-132).

[13] Mirzaei, N., Garcia, J., Bagheri, H., Sadeghi, A., & Malek, S. (2016, May). Reducing combinatorics in GUI testing of android applications. In Proceedings of the 38th international conference on software engineering (pp. 559-570).

[14] Aggarwal, A., Lohia, P., Nagar, S., Dey, K., & Saha, D. (2019, August). Black box fairness testing of machine learning models. In Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (pp. 625-635).

[15] Nie, C., & Leung, H. (2011). A survey of combinatorial testing. ACM Computing Surveys (CSUR), 43(2), 1-29.

[16] Wegener, J., Baresel, A., & Sthamer, H. (2001). Evolutionary test environment for automatic structural testing. Information and software technology, 43(14), 841-854.

[17] Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023, May). Large language models for software engineering: Survey and open problems. In 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE) (pp. 31-53). IEEE.

[18] Viglianisi E., Dallago M., Ceccato M., (2020) RESTTESTGEN: Automated Black-Box Testing of RESTful APIs. https://www.researchgate.net/publication/345604111

[19] Espada A. R., Maria del Mar G., Salmeon A., Merino P., Using Model Checking to Generate Test Cases for Android Applications. https://www.researchgate.net/publication/274730777

[20] Deorwine infotech. (August 6, 2023). Why AI in Software Testing Matters: A Game-Changer for Quality Assurance. http://deorwine.com/blog/ai-importance-in-software-testing/