STM JOURNALS

Recent Trends in
Programming Languages

ISSN: 2455-1821
Volume 12, Issue 3, 2025
September–December
DOI (Journal): 10.37591/RTPL

https://journals.stmjournals.com/rtpl

Research RTPL

Leveraging Generative AI for Test Case Creation in Complex Systems

Sachin Shankarrao Ajmire[1,*], Karuna Vivekanand Agarkar[2]

Abstract

Modern software systems exhibit increasing complexity, demanding sophisticated testing methodologies to ensure reliability and functionality. Traditional manual testing approaches often struggle to keep pace with this complexity, leading to inadequate test coverage and increased risk of unforeseen issues. This study explores the potential of Generative AI (GAI) in revolutionizing test case creation for complex systems. We delve into the practical application of GAI techniques, such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), to generate diverse and realistic test inputs, simulate user behavior, and identify critical edge cases. We discuss the benefits of GAI-driven testing, including enhanced test coverage, improved fault detection, and accelerated testing cycles. Furthermore, we address the inherent challenges, such as data quality requirements, model interpretability, and the need for robust evaluation metrics. Finally, we present a roadmap for successful GAI implementation in real-world testing scenarios, emphasizing the importance of human-in-the-loop approaches and continuous model refinement.

Keywords: Generative AI, test data, complex systems, software testing, generative adversarial networks (GAN), variable autoencoders (VAE), transformers, model training, data collection, program extension, efficiency, adaptability, verification, integration, edge cases, error detection, user reporting, system logging, workflow evaluation, hybrid model, automatic verification

INTRODUCTION

Today's software systems are becoming more complex and require more testing and optimization methods. Traditional testing methods often fail to address the diversity and nature of these systems [1]. This study aims to explore the application of AI in generating test cases for complex systems, thereby improving the overall quality and reliability of software products.

MOTIVATION

As software systems become more complex, the need for extensive testing increases, manually creating cases is time-consuming and prone to human error, and traditional automation systems cannot cover all possible cases.

***Author for Correspondence**
Sachin Shankarrao Ajmire
E-mail: sachinajmire7@gmail.com

[1]Research Scholar, Department of MCA (Master of Computer Application), G H Raisoni University, Amravati, Maharashtra, India
[2]Research Scholar, Department of Bachelor of Commerce, G H Raisoni University, Amravati, Maharashtra, India

Received Date: May 12, 2025
Accepted Date: June 05, 2025
Published Date: October 17, 2025

Citation: Sachin Shankarrao Ajmire, Karuna Vivekanand Agarkar. Leveraging Generative AI for Test Case Creation in Complex Systems. Recent Trends in Programming Languages. 2025; 12(3): 16–22p.

OBJECTIVE

Discover how generative AI can help create testable solutions for complex processes, increasing coverage and efficiency.

STRUCTURE

This study is divided into sections containing background information, methodology, results, challenges, case studies, and conclusions.

BACKGROUND

Generative AI

Generative AI consists of a class of algorithms that generate new data similar to the data given [2]. Key technologies include:

Variational Autoencoders (VAEs): VAEs uses a graphical design model to create new data by sampling from the latent space. This allows for the creation of changes from existing data by capturing a variety of possible states.

Transformers: Transformers use a control system to control sequence-to-sequence operations, making them suitable for creating sequences such as letters or numbers. They are especially useful for creating complex workflows and multi-step processes.

Software Testing

Software testing is an important phase in the software development lifecycle that aims to identify and fix defects before deployment [3–7]. Test cases are an essential part of testing and are designed to check the specific performance and behavior of the system under test.

- *Types of Testing:* With unit testing, integration testing, system testing, and validation, each test focuses on a different level of the system.
- *Importance of Test Cases:* A test case evaluates the conditions under which the system is tested by specifying the input, expected results, and execution steps. They are important to ensure that the system works as intended [8–13].

Current Methods and Limitations

Methods for creating test problems include building blocks, randomized tests, and standardized tests. However, these methods often face the following problems:

- *Scalability:* Manual and stochastic methods are labor intensive and do not scale well with complex methods. The number of tests needs to increase exponentially as the system grows.
- *Coverage:* It is difficult to keep track of all events and cases. Traditional methods may miss important cases and cause no defects.
- *Efficiency:* Developing and managing cases can be time-consuming and prone to human error. This slows down the testing process and results in lower test scores.

METHODOLOGY

Data Collection

The first step in creating a test problem using generative AI involves collecting different data and agents [14–20]. This data can include existing test cases, user reports and system files, storage of various applications and events.

- *Existing Test Cases:* Use existing test data as the basis for modeling to ensure that the test data generated is relevant and meets the requirements of the system.
- *User Reports:* Collecting information from user profiles can help capture real-world situations and situations that are not covered by current scenarios.
- *System Logs:* Analyzing system data can provide insight into the behavior of the system over time and highlight potential areas for new trends.

Here is a small comparative table for the different data sources used in generative AI-based test problem creation (Table 1).

Table 1. Examples of data sources for test scenario generation.

Data Source	Description	Example
Existing Test Cases	Predefined test cases used as a reference for generating new test scenarios.	Functional test cases for a banking application.
User Reports	Data from users reporting issues or usage patterns, helping to cover real-world scenarios.	Customer complaints about mobile app crashes.
System Logs	Historical system data capturing behavior over time, useful for detecting anomalies.	Server logs indicating failed API requests during peak hours.

Model Training

Generative models are trained on the collected data to learn the patterns and behaviors of the system. The selection of the model depends on the nature of the material and the specific requirements of the test data design study.

- *GANs:* It is used to create accurate and versatile test results by learning the distribution of training data. GANs can create new problems that follow patterns found in objects, thus providing a variety of states.
- *VAEs:* It is used to create variations of the experiment, exploring different permutations of the input conditions. VAE can generate new data that is similar to the training data but exhibits dynamic changes covering a wide range of possibilities.
- *Transformers:* It is used to create a series of test steps, especially useful for testing processes and procedures. Transformer is an expert at handling complex processes and can create cases that contain multiple steps and interactions.

Test Case Generation

When models are trained, they can generate new data and new problems. These tests can be further validated and improved using domain knowledge and other considerations.

- *Novelty and Relevance:* Ensure that test cases are developed to reflect new conditions while checking the impact on performance.
- *Validation:* Use domain knowledge and heuristics to analyze test designs and ensure they are usable and useful for testing.
- *Refinement:* Revise test cases based on recommendations and evidence to improve their quality and effectiveness.

BENEFITS

Increased Coverage

The design is capable of generating a variety of experiments, including edge applications that traditional methods may miss [20–23]. This leads to more testing and more error detection.

- *Edge Cases:* Generative AI can enable systems to be tested in a variety of situations by creating test cases that cover rare or unexpected scenarios.
- *Diverse Scenarios:* Generative AI helps generate more help and reduce the risk of conflict by creating a wide range of variables.

Efficiency

The automated test case generation process reduces the time and effort required to create a case. This allows the diagnostic team to focus on the complexity and importance of the system.

- *Time Savings:* Generative AI can quickly generate more scenarios, reducing the time required to create tests.
- *Resource Optimization:* Through automated processes, the testing team can allocate resources more efficiently and focus on important tasks such as test execution and analysis.

Adaptability

Generative models can be iterated on with new data, allowing them to continue to adapt and evolve with changes in the system. This ensures that test data remains relevant and timely.

- *Continuous Improvement:* As the process evolves, new components can be introduced into the training process, allowing the model to generate cases that reflect new and updated trends.
- *Adaptation to Change:* Generative AI quickly adapts to changes in the system, ensuring test data remains relevant and up-to-date.

CHALLENGES

Model Training

Training a suitable model requires a lot of good data and computational resources, and ensuring that these resources are available can be difficult.

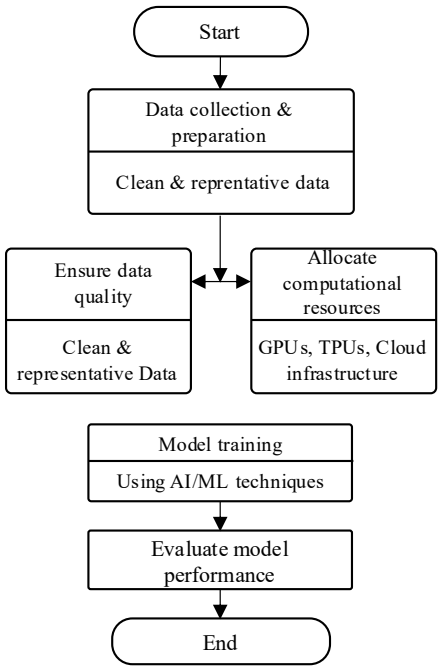

Figure 1. Flow chart.

Data Quality: Good training materials are essential for good model training. Ensuring cleanliness and data representation can be difficult.

Computational Resources: Training such models, especially large models like GANs and Transformers, requires a lot of computing power. Access to sufficient resources is critical for training a successful model (Figure 1).

Validation

The test materials created must be proven to be effective and relevant. This may include additional steps such as book review or fact checking based on a previous process.

- *Relevance:* Ensure that the test data generated is relevant to the system's performance and requirements.
- *Manual Review:* In some cases, manual review is required to ensure that test data was generated with full effort.
- *Automated Validation:* Establishing a validation process to assess the quality and impact of experimental design problems can help improve the validation process.

Integration

Integrating generative AI into existing testing models and processes requires careful planning and coordination. Building a strong relationship and seamless integration is key to getting the most out of this approach.

- *Compatibility:* Ensure that AI models can be integrated with existing measurement tools and models without disrupting existing operations.
- *Coordination:* Coordinate the integration process with testing teams and other stakeholders to ensure effective adoption and effective use of generative AI.

CASE STUDIES

Case Study 1: Financial Systems

A financial method was chosen to demonstrate the use of AI in creating business challenges. The model was trained from historical business data and test data was generated to identify vulnerabilities

in the system's business operations. The results show that detection of flaws is greatly improved and the time required for testing is short.

- *Data Collection:* Write job history documents that detail various types of jobs, case studies, and user interactions.
- *Model Training:* The GAN is trained based on the collected data and gains knowledge about the distribution of work.
- *Test Case Generation:* The GAN is trained to generate new scenarios that include similar events and business benefits.
- *Results:* Effective testing improves error detection and identifies previously undiagnosed issues in business logic. It also reduces the time required to develop cases.

Case Study 2: E-commerce Platforms

An e-commerce project was chosen to demonstrate the use of generative AI in testing user interactions and feedback processes. The model is trained on user interaction data and system data to create test data that includes a variety of user behaviors and states. This allows for better coverage of the checkout edge and results in a more robust test.

- *Data Collection:* Collect customer interaction data and system data to capture various user behaviors, interactions, and returns processes.
- *Model Training:* The transformer model is trained based on the collected data to learn the sequence of user interactions and responses.
- *Test Case Generation:* Training transformers created new tests that included different users of different situations and rescue methods.
- *Results:* Test results better cover cases and improve the performance of the testing system. This e-commerce business benefits from greater defect detection and better testing procedures.

CONCLUSION

Generative AI provides an effective way to test data generated for complex systems. By using state-of-the-art design, the testing team can achieve greater service, efficiency, and flexibility in the testing process. While challenges remain, the potential benefits make generative AI an important tool in the software testing arsenal.

Summary

Generative AI technologies like GANs, VAEs, and Transformers can improve the experimental design process for complex systems. Generative AI improves coverage, efficiency, and flexibility by creating test problems, enabling more efficient and comprehensive testing.

Future Work

Future research could focus on integrating AI with other measurement tools, exploring hybrid models that combine different techniques, and creating standard layers to control design test problems. Additionally, additional case studies and practical applications could provide a deeper understanding of the benefits and challenges of developing cases using generative AI.

Call to Action

Additional support and testing with AI products is essential in software testing. Organizations should explore the potential of generative AI to improve testing processes and increase the efficiency and reliability of software systems.

REFERENCES

1. Ale NK. Implementing Machine Learning Algorithms to Improve Test Case Generation and Execution. Journal of Scientific and Engineering Research. 2022; 9(7): 151–7.
2. Guo X, Okamura H, Dohi T. Automated software test data generation with generative adversarial networks. IEEE Access. 2022 Feb 22; 10: 20690–700.

3. Talakola S. The optimization of software testing efficiency and effectiveness using AI techniques. International Journal of Artificial Intelligence, Data Science, and Machine Learning. 2024 Oct 31; 5(3): 23–34.
4. Bajaj Y, Samal MK. Accelerating software quality: Unleashing the power of generative ai for automated test-case generation and bug identification. Int J Res Appl Sci Eng Technol. 2023 Jul; 11(7): 345–50.
5. Korol V. Integrating AI Tools Into the Continuous Testing Process. The American Journal of Engineering and Technology (TAJET). 2025 Jun 30; 7(06): 222–9.
6. Natarajan DR. AI-Generated Test Automation for Autonomous Software Verification: Enhancing Quality Assurance Through AI-Driven Testing. International Journal of HRM and Organizational Behavior. 2020 Nov 18; 8(4): 89–103.
7. Othman R, Zein S. Test Case Auto-Generation For Web Applications: A Model-Based Approach. In 2022 IEEE International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT). 2022 Oct 20; 18–25.
8. Mohapatra PS. Artificial Intelligence-Driven Test Case Generation in Software Development. In: Intelligent Assurance: Artificial Intelligence-Powered Software Testing in the Modern Development Lifecycle. Deep Science Publishing; Turkey. 2025 Jul 27; 4: 1-163.
9. Bajaj Y, Samal MK. Accelerating software quality: Unleashing the power of generative ai for automated test-case generation and bug identification. Int J Res Appl Sci Eng Technol. 2023 Jul; 11(7): 345–50.
10. Plein L, Ouédraogo WC, Klein J, Bissyandé TF. Automatic generation of test cases based on bug reports: a feasibility study with large language models. In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. 2024 Apr 14; 360–361.
11. Nabeel M, Nimara DD, Zanouda T. Test code generation for telecom software systems using two-stage generative model. In 2024 IEEE International Conference on Communications Workshops (ICC Workshops). 2024 Jun 9; 1231–1236.
12. Baqar M, Khanda R. The Future of Software Testing: AI–Powered Test Case Generation and Validation. In Intelligent Computing-Proceedings of the Computing Conference. Cham: Springer Nature Switzerland; 2025 Jun 19; 276–300.
13. Porres I, Rexha H, Lafond S. Online GANs for automatic performance testing. In 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). 2021 Apr 12; 95–100.
14. Damyanov I, Tsankov N, Nedyalkov I. Applications of Generative Artificial Intelligence in the Software Industry. TEM J. 2024 Nov 1; 13(4): 2724–2733.
15. Bandi A, Nukala HS, Tatavarthi B, Boggavarapu A. Automated Test Case Generation for Software Testing Using Generative AI. In International Conference on Computers and Their Applications. Cham: Springer Nature Switzerland; 2025 Mar 17; 78–87.
16. Sun L, Tao S, Hu J, Dow SP. Metawriter: Exploring the potential and perils of ai writing support in scientific peer review. Proc ACM Hum-Comput Interact. 2024 Apr 23; 8(CSCW1): 1–32.
17. Muhammad A. AI-Driven Testing Automation: Harnessing Machine Learning for Intelligent Test Case Creation and Predictive Defect Analysis. Int J Artif Intell Appl. 2024; 9(3): 22–35.
18. Hernández R, Manuel B, Ayala S, Martın J, Melo M, Andrés J. Generative ai for software architecture. Tech rep. Fundación Universitaria de Ciencias de la Salud; 2024 Jun 3.
19. Mehralian M, Karasfi B. RDCGAN: Unsupervised representation learning with regularized deep convolutional generative adversarial networks. In 2018 IEEE 9th conference on artificial intelligence and robotics and 2nd Asia-pacific international symposium. 2018 Dec 10; 31–38.
20. Rybkin O, Daniilidis K, Levine S. Simple and effective VAE training with calibrated decoders. In International conference on machine learning, PMLR. 2021 Jul 1; 9179–9189.
21. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S. Language models are few-shot learners. Adv Neural Inf Process Syst. 2020; 33: 1877–901.

22. Reed S, Akata Z, Yan X, Logeswaran L, Schiele B, Lee H. Generative adversarial text to image synthesis. In International conference on machine learning, PMLR. 2016 Jun 11; 1060–1069.
23. Brock A, Donahue J, Simonyan K. Large scale GAN training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096. 2018 Sep 28.

