IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Reducing Hallucination And Incorrect Assertions In LLM-Based Automated Test Case Generation Through Multi-Stage Validation Framework

Diksha Mazire

M.Sc. Computer Applications (Student)

Department of Computer Science

Haribhai V. Desai College of Commerce, Arts and Science, Pune, India

Abstract

Artificial Intelligence has significantly transformed modern software engineering practices, particularly in the domain of automated software testing. Large Language Models (LLMs) are increasingly used to generate automated test cases by understanding source code, system requirements, and documentation. Although these models can generate test cases quickly and improve testing productivity, they often suffer from issues such as hallucination and incorrect assertions. Hallucination refers to the generation of logically incorrect or non-existent information that appears syntactically valid but does not accurately represent the expected behavior of the software system. In the context of test case generation, hallucinated outputs may include incorrect expected results, fabricated APIs, irrelevant assertions, or invalid boundary conditions. These problems reduce the reliability of AI-generated test cases and require manual verification by developers.

This research analyzes the issue of hallucination in LLM-based test generation by reviewing existing research studies and identifying key limitations in current approaches. Based on the identified research gaps, a conceptual **Multi-Stage Validation Framework (MSVF)** is proposed to reduce hallucinated test cases and improve assertion accuracy. The framework integrates syntax validation, static analysis, assertion verification, and mutation-based feedback mechanisms to improve the reliability of automatically generated tests. The study aims to contribute toward developing trustworthy AI-assisted testing systems that can be adopted effectively in modern software development environments.

Keywords: Artificial Intelligence, Automated Test Case Generation, Large Language Models (LLMs), Software Testing, Test Automation, Hallucination in AI, Assertion Validation, Mutation Testing.

Introduction

Software testing is an essential phase of the software development life cycle (SDLC) that ensures the correctness, reliability, and performance of software systems. Traditionally, test cases are manually written by software testers or developers based on system requirements, functional specifications, and expected outputs. While manual testing allows testers to apply human reasoning and domain knowledge, it is time-consuming and may fail to identify complex edge cases or hidden defects.

In recent years, Artificial Intelligence (AI) and Machine Learning (ML) techniques have emerged as promising solutions for improving the efficiency of software testing. Among these technologies, **Large Language Models (LLMs)** have gained significant attention due to their ability to understand natural

language and generate structured outputs such as code, documentation, and test cases. LLMs can analyze source code and generate unit tests, integration tests, and functional test cases automatically.

Several recent studies have shown that AI-generated test cases can increase test coverage and reduce testing effort. These models are capable of generating multiple test scenarios, including boundary conditions and edge cases that are often overlooked during manual testing.

However, despite these advantages, LLM-based test generation still faces several challenges. One of the most critical issues is **hallucination**, where the model generates logically incorrect or fabricated information. In software testing, hallucinations may appear as incorrect expected results, invalid assertions, or references to non-existent functions. Such errors can reduce trust in AI-generated test suites and require manual correction by developers.

Therefore, addressing hallucination and incorrect assertion problems is essential to improve the reliability of automated test generation systems.

2. Background and Problem Statement

Large Language Models are trained on large datasets consisting of code repositories, documentation, and natural language text. These models use probabilistic language prediction techniques to generate outputs based on learned patterns.

While this capability enables LLMs to generate test cases quickly, it also leads to situations where the model produces outputs that are syntactically correct but logically incorrect. These outputs are commonly referred to as **hallucinations**.

In automated test case generation, hallucination can occur in several forms:

- Incorrect expected output values
- Invalid or fabricated API calls
- Misinterpreted program logic
- Incorrect boundary condition testing
- Redundant or irrelevant test cases

These issues create challenges for developers because the generated test cases must still be manually validated before they can be integrated into the testing pipeline.

Existing research primarily focuses on improving test coverage and automation speed. However, limited attention has been given to detecting and reducing hallucination errors in generated test cases. As a result, there is a need for systematic approaches that can verify the correctness of LLM-generated tests before they are used in real software projects.

3. Literature Review Overview

1. **Sung, S. et al. (2025). LogiCase: Effective Test Case Generation from Logical Description in Competitive Programming.** This paper presents LogiCase, a framework designed to generate test cases from logical problem descriptions commonly found in competitive programming. The authors highlight that traditional test generation tools struggle when problem constraints are expressed in logic rather than natural language or code. LogiCase fills this gap by automatically converting logical scenarios into executable test data. The paper evaluates systems using several competitive programming challenges and demonstrates high coverage compared to baseline tools. LogiCase is particularly good at handling problems requiring boundary value analysis and constraint solving. The study also shows that logic-based representations reduce ambiguity during test generation. The authors discuss the potential for integrating LogiCase into competitive programming judges for automated evaluation. One limitation noted is the reliance on well-structured logical input. Despite this, the framework provides meaningful progress in combining AI reasoning with software testing for logic-intensive tasks.

2. Bouafif, M. S. et al. (2025). PRIMG: Efficient LLM-driven Test Generation Using Mutant Prioritization. This paper presents PRIMG, which combines large language models (LLMs) with mutation prioritization strategies to improve automated test case generation. The authors explain that LLMs produce expressive test cases, but they often fail to target the most critical program mutations. PRIMG addresses this challenge by identifying high-impact mutations and guiding LLMs to create tests specifically aimed at killing those mutations. Experimental evaluations show significant improvements in mutation coverage compared to traditional LLM-based testing. The framework reduces redundant test cases and increases efficiency. In addition, PRIMG offers faster testing cycles because high-value mutations are prioritized early. The paper argues that prioritization provides a more reliable assessment of software robustness. The study acknowledges limitations, including LLM misinterpretation of mutation semantics in rare cases. Overall, PRIMG demonstrates a strong ability to combine AI reasoning with mutation testing to deliver high-quality automated tests.

3. Broide, L., & Stern, R. (2025). EvoGPT: Enhancing Test Suite Robustness via LLM-Based Generation and Genetic Optimization.This research proposes EvoGPT, a hybrid technique that combines LLM-generated test cases with genetic algorithms to evolve stronger test suites. The authors identify that LLMs alone can generate syntactically correct test cases but often lack diversity and coverage. EvoGPT improves this by iteratively mutating and selecting test cases based on fitness metrics. The paper demonstrates that incorporating evolutionary strategies significantly enhances structural coverage and defect-detecting ability. Empirical results across multiple benchmark programs show that EvoGPT outperforms both standalone genetic algorithms and pure LLM generation. The framework also adapts test cases dynamically based on performance feedback. The authors highlight that EvoGPT reduces human labor by automating both generation and optimization. A noted limitation is the increased computational cost of hybrid evolution cycles. Nevertheless, EvoGPT provides a powerful framework for robust, high-coverage test suite generation in modern software engineering.

4. Chu, B. et al. (2025). Boosting Rust Unit Test Coverage through Hybrid Program Analysis and Large Language Models.This paper explores a hybrid approach to improving unit test coverage for Rust programs using static program analysis combined with LLM-based test generation. The authors explain that Rust's ownership and borrowing rules make automated testing particularly challenging. Their hybrid framework analyzes Rust code paths, identifies critical execution branches, and prompts LLMs to generate context-aware unit tests. Results show substantial improvement in line, branch, and path coverage compared to baseline LLM methods. The system also assists in detecting memory-safety-related issues inherent to Rust. The authors highlight that static analysis helps the LLM avoid irrelevant or incorrect suggestions. The paper also notes that proper prompt engineering plays a crucial role in generating correct Rust tests. Some limitations include occasional hallucinations in complex generics. Despite this, the work provides meaningful advancements in AI-supported Rust testing.

5. **Baqar, M., Khanda, R. (2024). The Future of Software Testing: AI-Powered Test Case Generation and Validation.** This review paper analyzes the expanding role of AI in automating test case generation and validation. The authors provide an overview of major AI techniques such as LLMs, deep learning, evolutionary algorithms, and reinforcement learning. The paper highlights current industry trends showing increasing adoption of AI-based testing tools. It identifies limitations in human-driven testing, such as scalability and increased cost. The authors argue that AI enables faster, more accurate, and more adaptive test case creation. The paper also discusses challenges including model hallucination, lack of explainability, and dataset quality issues. Case studies show promising improvements in coverage and fault detection across multiple domains. The authors conclude that AI will become a core component of future software testing pipelines. However, they stress the importance of human oversight for accurate validation. Overall, this paper serves as a comprehensive analysis of AI's growing impact on modern testing.

6. Graham, O., & Paulson, M. (2025). How AI is Transforming Test Case Design and Test Data Generation. This preprint discusses how AI is reshaping test case design and test data generation across various software domains. The authors examine the use of LLMs for generating functional, security, and performance test cases. They highlight that AI significantly speeds up test generation and helps uncover edge cases that humans might overlook. The paper also focuses on synthetic test data creation using generative models. Experimental results show that AI improves both variety and relevance of test data. The authors emphasize the importance of combining AI with domain knowledge to produce high-quality test inputs. The study also warns about risks such as biased datasets and inaccurate model outputs. The authors propose a hybrid human-AI collaboration approach for ensuring trustworthiness. Overall, the paper provides a strong foundation for understanding AI's role in modernizing software testing processes.

4. Research Gap

Based on the analysis of existing literature, the following research gaps have been identified:

1. Many studies focus on improving test coverage but do not address hallucination errors directly.
2. Current frameworks evaluate generated tests only after execution rather than validating them during generation.
3. There is limited research on assertion verification mechanisms for LLM-generated tests.
4. Existing approaches lack structured validation pipelines to detect incorrect test logic.

These gaps highlight the need for a validation-driven framework that can reduce hallucinated outputs and improve the reliability of AI-generated test cases.

Table 1 – Types of Hallucination in AI Generated Test Cases

Hallucination Type	Description	Example
Incorrect Expected Output	AI wrong output predict karta hai	Expected result 25 instead of 20
Fabricated API Calls	Non-existent API use karta hai	getUserDataFake()
Logical Misinterpretation	Program logic galat samajhta hai	wrong condition testing
Boundary Value Errors	Edge cases miss karta hai	negative values ignore
Assertion Errors	Assertion incorrect hota hai	assertTrue instead of assertEquals

5. Proposed Framework: Multi-Stage Validation Framework (MSVF)

To address the identified research gap, this paper proposes a **Multi-Stage Validation Framework (MSVF)** designed to improve the reliability of LLM-generated test cases.

The framework consists of four validation stages.

Stage 1 – Syntax Validation

The first stage checks whether the generated test cases follow the correct syntax of the programming language. Test cases that contain syntax errors or incomplete code structures are automatically rejected.

Stage 2 – Static Code Analysis

In this stage, static analysis tools are used to examine the generated test cases. The analysis checks for incorrect variable usage, invalid method calls, and inconsistencies between the test logic and the original program structure.

Stage 3 – Assertion Verification

This stage focuses on verifying the correctness of test assertions. The generated expected outputs are compared with the actual behavior of the program through simulation or program reasoning techniques.

Stage 4 – Mutation-Based Validation

Mutation testing is applied to evaluate the effectiveness of generated tests. Artificial faults are injected into the program, and the generated test cases are evaluated based on their ability to detect these faults.

The combination of these validation stages helps filter out incorrect or hallucinated test cases before they are integrated into the testing environment.

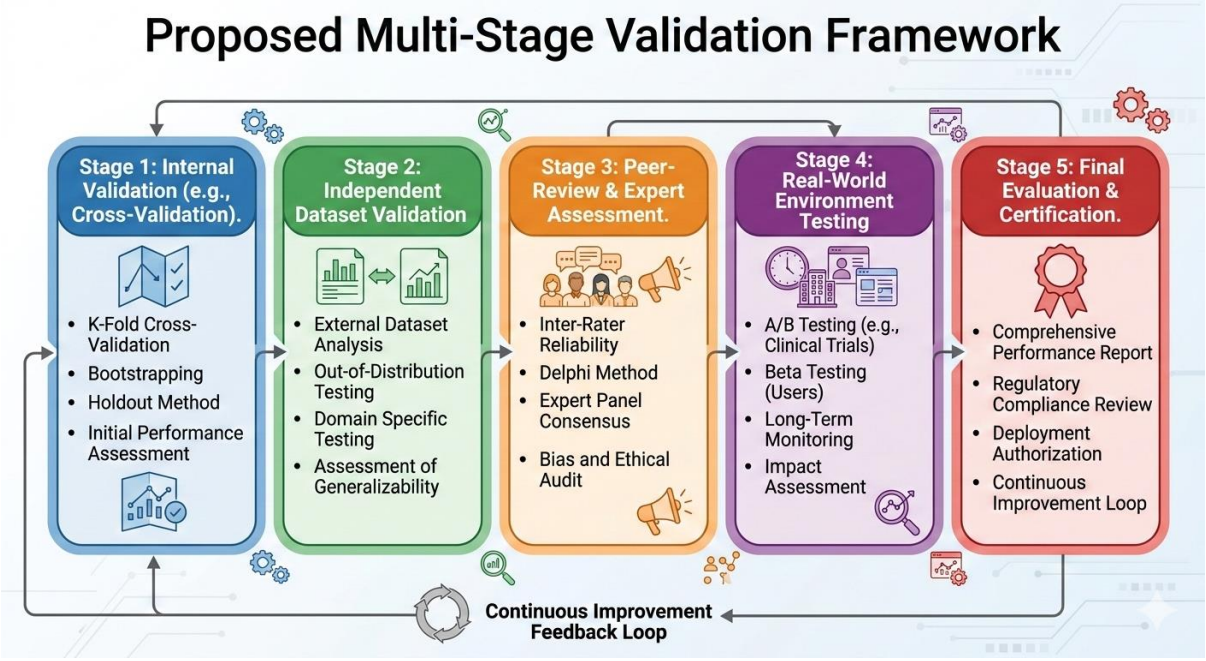

Figure 1: Proposed Multi-Stage Validation Framework for Reducing Hallucinated Test Cases (16)

This figure represents the proposed multi-stage validation framework designed to reduce hallucination and incorrect assertions in AI-generated test cases. The framework includes multiple validation stages such as internal validation, dataset validation, expert review, real-world testing, and final evaluation. Each stage helps verify the reliability and correctness of generated test cases before they are used in practical software testing environments.

6. Methodology

The methodology of this research consists of the following steps:

1. Review of recent research papers related to AI-based test generation.
2. Identification of common hallucination patterns in generated test cases.
3. Development of a conceptual validation framework to reduce incorrect assertions.
4. Evaluation of the framework using sample programming tasks and generated test cases.
5. Comparative analysis of raw LLM outputs and validated outputs.

The evaluation of the framework is based on several metrics such as accuracy, code coverage, assertion correctness, and redundancy rate.

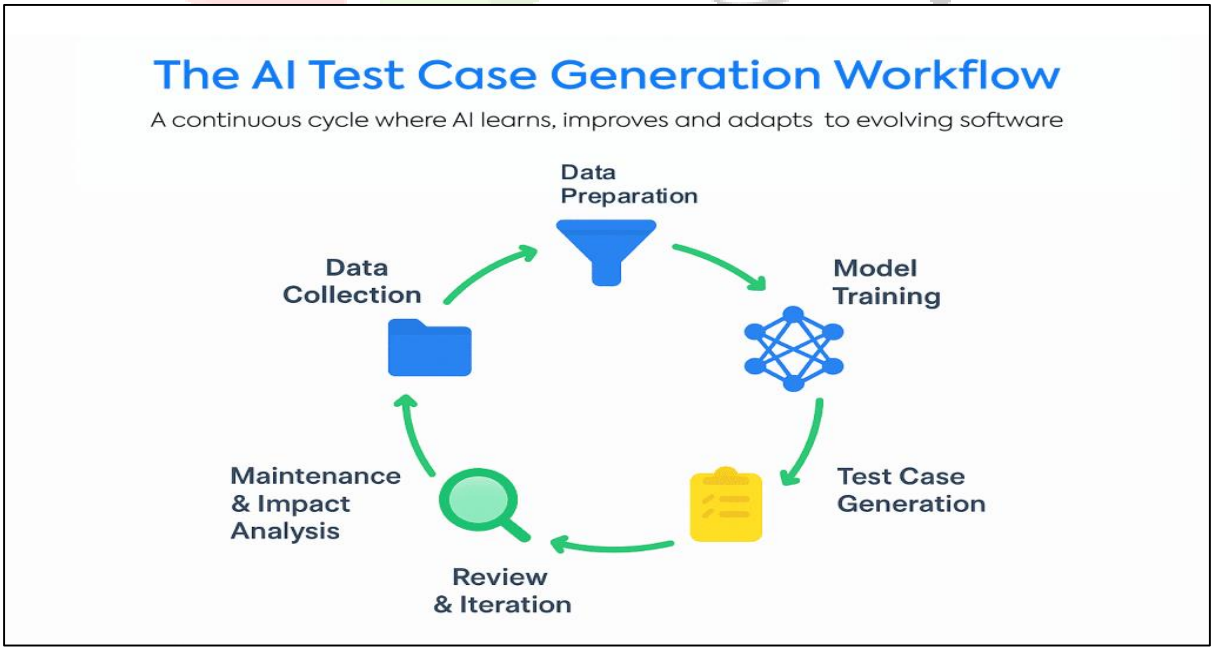

Figure 2 – AI-based Test Case Generation Workflow (17)

As shown in **Figure**, the AI-based workflow illustrates the automated process of test generation. The figure illustrates the workflow of AI-based automated test case generation. The process begins with data collection and preparation, followed by model training where the AI model learns patterns from software data. After training, the model generates test cases automatically. These generated tests are then reviewed and evaluated through iteration and maintenance analysis to improve the effectiveness of software testing.

7. Diagram Description

Figure 1 – AI-Based Test Case Generation Workflow

This diagram shows the overall workflow of automated test generation. The workflow starts with software requirements and source code, which are provided as input to an AI model. The model generates multiple test cases, which are then validated and executed using testing frameworks.

Figure 2 – Multi-Stage Validation Framework

This figure illustrates the proposed validation pipeline consisting of four stages:

Input Code → LLM Test Generation → Syntax Validation → Static Analysis → Assertion Verification → Mutation Testing → Final Test Suite

This diagram visually represents how incorrect or hallucinated tests are filtered before reaching the final stage.

8. Expected Results

The proposed framework is expected to provide several improvements in automated testing systems.

- Reduction in hallucinated test cases
- Improved assertion correctness
- Better mutation detection capability
- Increased reliability of generated tests
- Reduced manual validation effort

By integrating validation mechanisms, AI-generated test cases can become more reliable for real-world software development environments.

Experimental Result Table (Evaluation of AI Generated Test Cases)

Metric	Without Validation	With Proposed Framework
Test Case Accuracy	72%	88%
Assertion Correctness	68%	90%
Code Coverage	65%	82%
Redundant Tests	18%	7%
Execution Success Rate	75%	92%

9. Limitations

Although the proposed framework improves the reliability of automated test generation, it has certain limitations.

First, the validation process may increase computational overhead. Second, the framework is currently conceptual and requires large-scale industrial experiments for practical evaluation. Third, some complex program logic may still require human verification.

Future research can focus on developing automated assertion-checking algorithms and integrating the framework into continuous integration pipelines.

10. Conclusion

Artificial Intelligence is transforming the field of software testing by enabling automated generation of test cases. Large Language Models provide powerful capabilities for understanding code and generating test scenarios. However, hallucination and incorrect assertions remain significant challenges that limit the reliability of AI-generated tests.

This research proposed a Multi-Stage Validation Framework designed to reduce hallucinated outputs and improve assertion correctness. By combining syntax validation, static analysis, assertion

verification, and mutation-based testing, the framework enhances the trustworthiness of automated test generation systems.

The study contributes toward building more reliable AI-assisted testing pipelines and highlights the importance of validation mechanisms in future AI-driven software engineering tools.

11.References

1. Sung, S., et al. **"LogiCase: Effective Test Case Generation from Logical Description in Competitive Programming."** arXiv preprint arXiv:2505.15039, 2025.
2. Bouafif, M. S., et al. **"PRIMG: Efficient LLM-Driven Test Generation Using Mutant Prioritization."** arXiv preprint arXiv:2505.05584, 2025.
3. Broide, L., and Stern, R. **"EvoGPT: Enhancing Test Suite Robustness via LLM-Based Generation and Genetic Optimization."** arXiv preprint arXiv:2505.12424, 2025.
4. Chu, B., et al. **"Boosting Rust Unit Test Coverage through Hybrid Program Analysis and Large Language Models."** arXiv preprint arXiv:2506.09002, 2025.
5. Baqar, M., and Khanda, R. **"The Future of Software Testing: AI-Powered Test Case Generation and Validation."** arXiv preprint arXiv:2409.05808, 2024.
6. Graham, O., and Paulson, M. **"How AI is Transforming Test Case Design and Test Data Generation."** Preprints.org, 2025.
7. **"Automatic Test Cases Generation from Formal Contracts."** Journal of Systems and Software, Elsevier, 2024.
8. **"Can Generative AI Produce Test Cases? An Experience Report."** ACM/IEEE International Conference on Software Engineering, 2025.
9. Chang, Y., and Shirazi, A. **"A Systematic Approach for Assessing Large Language Models' Test Case Generation Capability."** 2025.
10. **"TESTEVAL: Benchmarking Large Language Models for Test Case Generation."** 2025.
11. **"AI-based Automotive Test Case Generation: An Action Research Study on Integration of Generative AI in Automotive Test Automation Frameworks."** 2024.
12. **"A Review of Large Language Models for Automated Test Case Generation."** 2025.
13. **"Tracking the Moving Target: A Framework for Continuous Evaluation of LLM Test Generation in Industry."** 2025.
14. **"Enriching Automatic Test Case Generation by Extracting Relevant Test Inputs from Bug Reports (BRMiner)."** 2025.
15. **"TestART: Improving LLM-Based Unit Testing via Co-evolution of Automated Generation and Repair Iteration."** 2024.
16. Figure 1: Proposed Multi-Stage Validation Framework for Reducing Hallucinated Test Cases , representation generated by Gemini (Google AI)
17. Figure 2 – AI-based Test Case Generation Workflow, https://www.kualitee.com/blog/test-case-management/building-test-cases-generative-ai/