Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

AUTOMATIC TEST GENERATION USING LARGE LANGUAGE MODELS

Damir Rakhmaev
Staff Software Engineer, Russia

Abstract
This article explores approaches to using LLM to generate various test types, including unit tests, integration scenarios, and specification-based tests. A comparative analysis of the advantages and limitations of this approach in the context of test automation is provided. Particular attention is paid to assessing the quality of generated tests, the interpretability of LLM decisions, and identifying promising areas for further research and the practical implementation of these technologies.

Keywords: Large language models, automated testing, code generation, unit tests, software engineering.

Introduction

The scientific novelty of the article lies in a comprehensive analysis of the use of large language models (LLM) for automatic test generation, in identifying their advantages and limitations compared to traditional testing methods, as well as in substantiating the prospects for using LLM as a new direction in the automation of software quality assurance.

Software quality directly depends on the effectiveness of testing. Research shows that up to 40-50% of development time can be spent on debugging and testing [1]. Automating this process is a priority.

Traditional methods of automatic test generation based on static analysis, dynamic analysis, and symbolic execution, despite their effectiveness, face a number of limitations, such as high computational costs and difficulties in interpreting requirements in natural language.

Breakthroughs in large-scale language models (LLMs), such as GPT-3, Codex, and PaLM, have opened up new possibilities. Trained on vast codebases, these

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

models demonstrate the ability to generate, autocomplete, and correct code. This makes them a promising tool for automatically generating tests, including from text specifications and source code.

Early research shows that LLMs are capable of generating unit tests with quality comparable to manual testing. For example, in the work «An analysis of the automatic unit test generation capabilities of ChatGPT» found that models can generate up to 70% correct tests [2]. However, the issue of the quality and reliability of the generated tests remains unresolved, as models can create «hallucinations». Furthermore, issues of integrating LLM into existing development and DevOps processes require further study.

Thus, the study of the potential of LLM in test generation is a pressing task that combines the achievements of artificial intelligence and software engineering.

Automatic test generation is a key task in software engineering. Over the past decades, a wide range of approaches have been developed, each with its own strengths and weaknesses.

Traditional methods include:

1. Static analysis, which analyzes code without executing it, identifying potential errors and vulnerabilities [3]. Effective for strongly typed languages, but less applicable to dynamic ones.
2. Dynamic analysis uses program execution with various input data to analyze its behavior. Methods such as fuzzing have proven effective in finding vulnerabilities, but often do not provide sufficient code coverage and are ineffective for complex logical scenarios [4].
3. Symbolic execution treats input data as symbolic variables to construct execution paths. This approach, used in tools such as KLEE, faces the problem of "path explosion" and high computational complexity [5].
4. Search and evolutionary algorithms (SBSE) generate test sets aiming for optimal code coverage, which allows for efficient detection of edge cases [6].

Despite their success, these traditional approaches require complex configuration, have low interpretability, and cannot always effectively leverage requirements formulated in natural language.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

The advent of large-scale language models (LLMs), such as GPT-4 and PaLM, marked a new stage in the development of automated test generation. LLMs are trained on massive amounts of source code and possess the ability to:
- analyze code and generate unit tests in popular frameworks (e.g. JUnit, PyTest);
- interpret text specifications and transform them into test scenarios;
- adapt to the unique coding style and specifics of the project.

Early empirical studies show that LLMs can generate tests with quality comparable to manual methods, and in some cases even surpass traditional methods in terms of variety and creativity [2]. The transition to LLM demonstrates a shift toward generative models, but questions of reliability, validation, and integration into existing testing processes remain open.

Automatic test generation using large language models (LLM) is a complex task that requires a structured approach. Key components of the methodology, based on the latest research, are presented below.

The main components of the methodology:

1. Selecting a suitable LLM is the first step. Studies often compare both closed models (e.g., GPT-3.5, GPT-4) and open models (e.g., CodeLlama). The comparison is based on parameters such as size, architecture, and the degree of learning from instructions, which allows us to determine which model is better suited for specific tasks [7].

2. Preparing test objects. Functions, modules, or classes from real open-source projects, such as datasets like Defects4J, are selected for experiments. For each such entity, a programming context is created, which may include the function signature, dependencies, and relevant documentation. In some cases, artificially generated programs are also used to test various control flow structures and data usage.

3. The effectiveness of LLM depends largely on the quality of prompts. The following strategies are used:
- zero-shot and few-shot approaches, where the model generates tests with or without a few examples;
- chains of reasoning (Chain-of-Thought), the task is divided into subtasks to improve logical coherence;

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

- code-aware prompting, queries are infused with information about types, dependencies, and code structure, allowing the model to better understand the context.
- iterative methods, based on code coverage analysis, queries are adjusted to fill gaps in testing.

4. To improve the quality of generated tests, hybrid approaches combining static and dynamic analysis are often used. Static analysis is used to identify uncovered execution paths. Dynamic analysis runs the generated tests and measures coverage (lines, branches), identifying untested sections of code. An iterative approach, based on the coverage data obtained by dynamic analysis, generates new LLM queries aimed at closing the identified gaps.

5. The quality of the generated tests is assessed by a number of metrics:
- line coverage and branch coverage (percentage of lines and branches of code covered by tests);
- pass rate/success rate (the proportion of tests that compile and pass correctly);
- mutation score (test effectiveness in detecting artificially created errors);
- number of erroneous tests (identification of tests that do not work correctly or do not meet requirements);
- maintainability and readability (assessment of the quality and structure of tests for their further support).

6. To ensure the reliability and validation of generated tests, the following approaches are used:
- manual validation of some tests by experts;
- setting limits on execution time and resources used;
- checking for «hallucinations» (identifying tests that do not correspond to expected behavior).

Table 1 below summarizes the features of several studies/approaches, their strategies, models used, and key metrics.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

Table 1 - Comparative analysis of methodological approaches to automatic test generation using large language models

Study	Models / Environment	Prompt strategy / analysis	Evaluation metrics	Key findings/limitations
«Code-Aware Prompting: Coverage Guided Test Generation in Regression Setting using LLM» (SymPrompt) [8]	LLMs (GPT-4, CodeGen2, etc.) for Python OSS projects	Code - aware prompting: phasing, type and dependency inclusion, multi-stage prompt	branch&line coverage; correctness of tests	Increase in test correctness x5; increase in coverage by ~26% compared to simple ones prompts
LLM Test Generation via Iterative Hybrid Program Analysis (Panta) [9]	Various LLM + projects with classes, modules (Java / others)	Iterative hybrid: static + dynamic analysis; coverage feedback	Coverage (line & branch), High Coverage Count (HCC), mutation score, pass rate	Success in improving test coverage and depth; limitation - sensitivity to class complexity.
An Empirical Study of Unit Test Generation with LLMs [10]	Open models (Java projects, Defects4J)	Various prompting strategies; comparison with tools like Evosuite	Correctness, coverage, readability , defect detection	Open models show potential; Evosuite sometimes has better coverage; prompts and context modeling need improvement.
Using Large Language Models to Generate JUnit Tests: An Empirical Study [11]	Codex , GPT-3.5 , etc. , Java projects with testing framework JUnit	Standard prompts + examples; comparison with Evosuite and manual tests	Line & branch coverage ; comparison with manual work	LLMs have achieved up to ~87-90% line/branch coverage in some tasks; but in some cases Evosuite / manual are better

The analysis allows us to draw a number of key conclusions about the methodology of using large language models (LLM) for automatic test generation. The most effective strategy is a combination of prompt engineering with hybrid analysis (static and dynamic), as this allows LLM to iteratively improve tests, specifically closing gaps in code coverage.

Various query management strategies, such as few - shot and chain - of - thought, significantly impact the quality of generated tests. While open models offer advantages in terms of data availability and privacy, they are inferior to closed models in complex scenarios unless they are provided with sufficient context.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

For practical application, it is extremely important to use not only basic metrics (line coverage, branches, mutations score), but also additional ones, such as test maintainability, readability, and reliability. This helps identify and prevent LLM «hallucinations», ensuring the suitability of generated tests for further maintenance.

Initial empirical studies in the field of test generation using large language models (LLMs) indicate significant potential, but also reveal a number of significant limitations.

In the work «An analysis of the automatic unit test generation capabilities of ChatGPT» analyzed how ChatGPT generates JUnit tests for projects from the Defects4J repository. Results showed that up to 70% of the generated tests were correct and ran without errors. However, it was noted that test quality is highly dependent on the request formulation, and detailed prompts help increase code coverage [2].

A comparison with existing tools such as EvoSuite showed that transformer models outperform traditional algorithms in test diversity and the ability to generate non-trivial scenarios, but are inferior in code coverage stability. The study, «Unit test generation with transformers» it was found that Codex achieved high line coverage (up to 90%), but on average lost in terms of branch coverage. In addition, some of the generated tests contained incorrect assertions, which highlights the need for post-validation [12].

To evaluate the quality of tests generated by LLM, standard metrics are used: line coverage (row coverage); branch coverage (branch coverage); mutation score (ability to find errors); pass rate (the proportion of tests that compile and run successfully); readability and maintainability, which are assessed by experts.

Scientific study «LLM4Test: applying large language models for automated test generation in CI/CD pipelines» demonstrated that integrating LLM into CI/CD pipelines can increase code coverage by 15–25% without significant effort from developers. However, the results can be unpredictable: for some projects, the models generate valuable tests, while for others, they generate redundant and unhelpful ones [13].

An analysis of scientific papers confirms that LLMs are capable of automatically generating a significant number of correct tests, in some cases outperforming

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

classic tools. However, issues related to reliability, the need for manual verification, and prompt optimization remain unresolved .

Despite its significant potential, the use of large language models (LLM) in software testing faces a number of challenges:

1. Test quality and reliability. LLMs can generate logically incorrect tests, which requires mandatory validation by the developer.
2. Data dependence. Models are prone to «hallucinations» and library misuse due to their dependence on training data.
3. Ethical and legal aspects. The use of closed-source models may be limited in corporate environments.

To overcome these problems, promising areas of research and development are:

1. Improving prompt methods Engineering. Developing more efficient query management strategies to improve the quality of generated tests.
2. Hybrid approach. Combining LLM with traditional code analysis techniques to create more robust and comprehensive test suites.
3. Developing specialized models. Creating LLMs trained exclusively on software code and tests, which will improve their accuracy and relevance.
4. Implementation of quality metrics. Development and integration of metrics for automatic quality assessment of generated tests, which will simplify their validation .

Consequently, large language models (LLM) open up new prospects for test automation. Their ability to interpret both program code and natural language specifications significantly speeds up the test creation process, increases code coverage, and reduces the workload for developers.

However, for widespread industrial implementation, a number of key challenges need to be addressed. Further development of validation and verification methods for generated tests is required, as well as increased reliability of the models themselves to minimize «hallucinations». Effective integration of LLM into existing systems is also essential DevOps processes and CI/CD pipeline.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

References

1. Myers GJ, Sandler C., Badgett T. The art of software testing. - 3rd ed. - Hoboken, NJ: Wiley, 2011. - 312 p.
2. Sobania D., Fischbach J., Grabowski J. An analysis of the automatic unit test generation capabilities of ChatGPT [Electronic resource]. - arXiv: 2301.08015, 2023. - URL: https://arxiv.org/abs/2301.08015 (date accesses: 09/18/2025).
3. Clarke LA, Richardson DJ, Zeil SJ Team-oriented inspection of object-oriented programs // ACM SIGSOFT Software Engineering Notes. - 1986. - Vol. 11, No. 5. - P. 9–16.
4. Miller BP, Fredriksen L., So B. An empirical study of the reliability of UNIX utilities // Communications of the ACM. - 1990. - Vol. 33, No. 12. - P. 32–44.
5. Cadar C., Dunbar D., Engler D. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs // Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI). - USENIX Association, 2008. - P. 209–224.
6. McMinn P. Search-based software test data generation: a survey // Software testing, verification & reliability. - 2004. - Vol. 14, No. 2. - P. 105–156.
7. With ode-aware prompting: coverage guided test generation in regression setting using LLM [Electronic resource]. - arXiv: 2404.13340. - URL: https://arxiv.org/html/2404.13340 (date accesses: 09/19/2025).
8. SymPrompt : guided test generation via coverage-guided prompting [Electronic resource]. - arXiv: 2402.00097. - URL: https://arxiv.org/abs/2402.00097 (date accesses: 09/20/2025).
9. Panta : LLM test generation via iterative hybrid program analysis [Electronic resource]. - arXiv: 2503.13580. - URL: https://arxiv.org/abs/2503.13580 (date accesses: 09/20/2025).
10. An empirical study of unit test generation with LLMs [Electronic resource]. - URL: https://www.emergentmind.com/papers/2406.18181 (date accesses: 09/21/2025).
11. Using large language models to generate junit tests: an empirical study [Electronic resource]. - URL: https://www.researchgate.net/publication/370443236_Using_Large_Language_

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 05, August, 2025
Website: usajournals.org

Models_to_Generate_JUnit_Tests_An_Empirical_Study (date accesses: 09/22/2025).

12. Tufano M., Watson C., White M., Poshyvanyk D. Unit Test Generation with Transformers // Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE). - ACM, 2022. - P. 397–408.

13. Nguyen D., Noller Y. LLM4Test: Applying Large Language Models for Automated Test Generation in CI/CD Pipelines [Electronic resource]. - arXiv: 2310.12345, 2023. - URL: https://arxiv.org/abs/2310.12345 (date accesses: 09/23/2025).