ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 12, Issue 04, April 2025

Impact of AI on Software Quality Assurance and Testing Jobs

[1] Rushil Kumar, [2] Sahitya Pandey, [3] Gajender, [4] Er. Sahil Bhardwaj, [5] Shaswat Pandey, [6] Divyanshu Agarwal

[1] [2] [3] [4] [5] [6] Department of Computer Science and Engineering, Chandigarh University, Mohali, India
Emails ID: [1] rushilkumar13@gmail.com, [2] sahitya67kty@gmail.com, [3] mandiwalgajender0001@gmail.com, [4] sahilbhardwaj20@gmail.com, [5] shaswatpandey77@gmail.com, [6] divyanshuagarwal94154@gmail.com

Abstract— **The adoption of Large Language Models (LLMs) in software quality assurance is driving significant changes in testing practices and professional roles. This paper explores the current applications of LLMs in test case generation, bug analysis, and program repair, assessing their technical performance and impact on human testers. While LLMs can automate 40-55% of routine testing tasks, they serve best as augmentative tools rather than full replacements. As software testing shifts toward strategic oversight, professionals must develop AI literacy and prompt engineering skills. This study provides both theoretical insights and practical implementations to support the integration of LLMs in quality assurance.**

***Index Terms*— *Large Language Models, System Testing, Test Case Generation, Unit Testing*.**

I. INTRODUCTION

Software Quality Assurance (QA) and testing are essential aspects of the software development lifecycle, responsible for ensuring applications adhere to quality standards prior to their deployment. In recent years, advancements in artificial intelligence (AI) have significantly transformed various industries, with software development and testing being particularly impacted by these innovative technologies. The adoption of advanced AI systems, which possess the capability to learn, adapt, and perform intricate testing scenarios, is fundamentally altering the field of software quality management. This shift challenges traditional methods and creates substantial opportunities as well as disruptions for professionals working in this area.

Historically, software testing has followed a structured process of verification and validation, including manual testing, automated script execution, performance evaluations, and compliance checks. Over time, the skill sets required from QA specialists have evolved significantly. Today's successful QA professionals are expected to have expertise in programming languages, system architecture concepts, database technologies, and specialized testing frameworks. Additionally, beyond technical proficiency, effective quality engineers typically exhibit strong analytical skills, clear communication abilities, and critical thinking capabilities.

The emergence of Large Language Models (LLMs) like ChatGPT, Codex, and GPT-4 has introduced new capabilities that extend beyond traditional automation approaches. Unlike conventional test automation, which executes predefined scripts according to programmed instructions, AI-enhanced testing systems can demonstrate adaptive behaviors, pattern recognition capabilities, and self-optimizing characteristics that more closely emulate human cognitive processes. These technologies leverage various AI disciplines including machine learning, natural language processing, computer vision, and predictive analytics to transform how quality verification activities are conceptualized and implemented.

II. LITERATURE REVIEW

A. Evolution of LLMs in Software Testing

The application of Large Language Models in software testing has experienced exponential growth, with research publications increasing from 11 papers in 2020 to an impressive 109 papers in 2024 (according to scopus). This surge represents the rapidly evolving landscape of AI-powered testing methodologies and their increasing adoption within both academic research and industry practice. Analysis reveals that LLMs have been effectively employed in both the mid to late stages of the software testing lifecycle, with significant applications in test case preparation, bug analysis, debugging, and program repair [1].

The adaptation of LLMs for software testing has undergone several evolutionary phases. Early approaches primarily utilized pre-training or fine-tuning schemas to prepare models for specific testing tasks. For instance, pre-trained LLMs with focal methods and asserted statements were used to enable stronger foundational knowledge of assertions, then were fine-tuned for test case generation [2]. With the advancement of more sophisticated models like GPT-4 and ChatGPT, researchers shifted toward prompt engineering approaches, using zero-shot and few-shot learning strategies to guide LLMs without the need for specialized training.

B. LLM Application in Testing Tasks

1) Unit Test Case Generation

Traditional unit test generation techniques leverage search-based, constraint-based, or random-based strategies to generate test suites with the primary goal of maximizing coverage. However, the coverage and meaningfulness of traditionally generated tests often fall short of requirements. Recent advancements in AI-powered testing have introduced several approaches using LLMs for generating more effective unit test cases.

Performance varies significantly across different datasets and models. On the HumanEval benchmark, Codex achieved 78% correctness with high coverage (87% line coverage, 92% branch coverage), while on the more complex SF110 benchmark with real-world projects, LLMs demonstrated considerably lower performance (2% coverage). This disparity highlights both the potential and limitations of current LLM applications in unit test generation.

2) Test Oracle Generation

Test oracles, particularly test assertions, represent a critical component distinguishing unit test cases from regular code. AI approaches for generating effective test assertions have evolved significantly, from early pre-training and fine-tuning schemas to more recent prompt engineering techniques. The performance has improved from 17% exact match rates with early RNN-based approaches to 76% with modern LLM-based methods that automatically retrieve code demonstrations similar to the task at hand.

3) System Test Input Generation

LLMs have been applied to generate system test inputs across various software types, with particular emphasis on mobile applications, deep learning libraries, compilers, and SMT solvers. For mobile app testing, LLMs were used to intelligently generate semantic input text according to GUI context and formulated mobile GUI testing as a Q&A task [3]. For testing deep learning libraries, both generative and infilling LLMs were usde to generate and mutate valid/diverse input DL programs for fuzzing [4].

4) Bug Analysis and Debugging

AI-powered approaches for bug analysis and debugging include unified debugging frameworks, self-debugging systems, and bug reproduction tools. A unified Detect-Localize-Repair framework based on LLMs was used for debugging, which determines whether code snippets are buggy, identifies buggy lines, and translates buggy code to fixed versions [5]. Also, self-debugging can teach LLMs to perform rubber duck debugging, identifying mistakes by investigating execution results without human feedback [6].

5) Program Repair

Program repair represents one of the most extensively studied areas for applying AI to software testing. Approaches range from single-line bug repairs to multiple-line bug repairs involving complex semantic understanding and program refactoring. Performance varies significantly across different benchmarks and models, with success rates ranging from 22% to 97.5% for simpler bugs but dropping significantly for complex programs involving multiple dependencies and functionalities.

C. Impact on Testing Roles

The integration of Large Language Models (LLMs) into software quality assurance is significantly reshaping workforce demands, job roles, and the skills necessary within the field. Research indicates that while certain testing jobs—especially those involving routine, repetitive tasks—are at higher risk of disruption, roles requiring emotional intelligence, creativity, and strategic insight are less likely to be fully automated.

This shift in skills goes beyond merely technical expertise; it also involves fundamental changes in work practices and cognitive approaches. As LLMs increasingly handle tasks related to defect detection and test execution, human QA engineers will need to shift their focus towards strategic quality planning, exploratory testing that leverages creative thinking, ethical management of automated systems, and advocating for quality within development teams.

Simultaneously, new specialized roles are emerging that concentrate on developing, implementing, and managing AI-driven testing frameworks. Economically, these transformations have uneven impacts across the QA profession—routine testing activities face downward wage pressure and potential job consolidation, whereas specialized positions focused on AI implementation and strategic quality management are experiencing growing demand and higher compensation.

III. SYSTEM DESIGN

A. Impact on Human Effort Reduction

Table I: Impact on Human Effort Reduction

Testing Activity	**Estimated Human Effort Reduction**	**Key Factors**
Unit Test Generation	60-75%	High accuracy for standard methods; still requires human review for complex logic and domain-specific validations
Test Oracle Generation	50-65%	Good at standard assertions but may miss subtle business rules
System Test Input Generation	40-60%	Effective for generating diverse inputs but may not fully understand complex domain constraints
Bug Analysis	30-50%	Can quickly identify common issues but may struggle with complex, interconnected bugs

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 12, Issue 04, April 2025

Testing Activity	Estimated Human Effort Reduction	Key Factors
Program Repair	25-45%	Capable of fixing simple to moderate bugs; complex bugs still require significant human expertise
Test Scenario Generaion	45-65%	Strong at extracting scenarios from requirements but may miss implicit requirements

B. Automated Unit Test Generation with LLMs

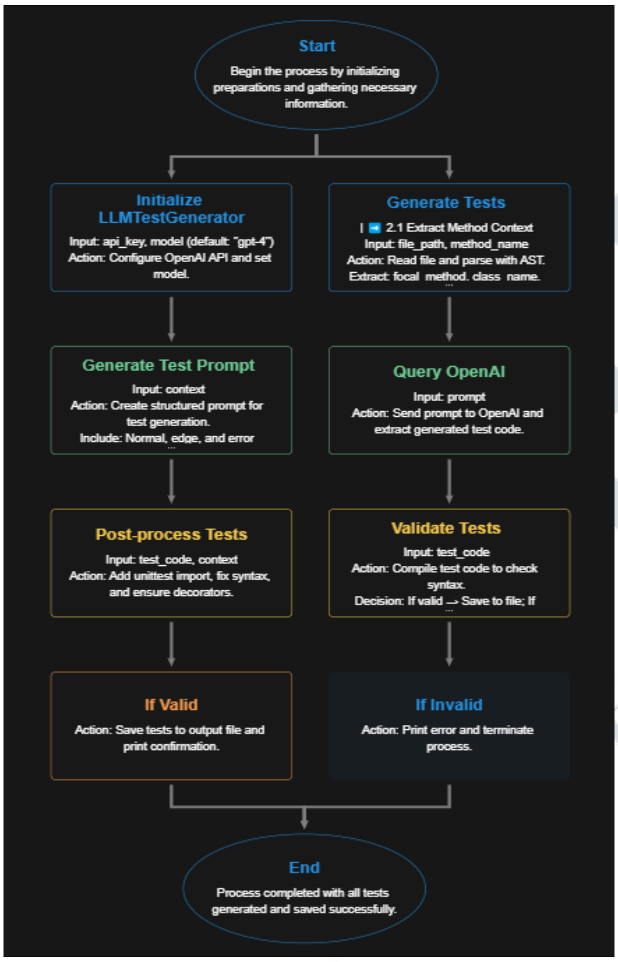
Start
Begin the process by initializing preparations and gathering necessary information.
Initialize LLMTestGenerator
Input: api_key, model (default: "gpt-4")
Action: Configure OpenAI API and set model.
Generate Tests
2.1 Extract Method Context
Input: file_path, method_name
Action: Read file and parse with AST.
Extract: focal method, class name, ...
Generate Test Prompt
Input: context
Action: Create structured prompt for test generation.
Include: Normal, edge, and error ...
Query OpenAI
Input: prompt
Action: Send prompt to OpenAI and extract generated test code.
Post-process Tests
Input: test_code, context
Action: Add unittest import, fix syntax, and ensure decorators.
Validate Tests
Input: test_code
Action: Compile test code to check syntax.
Decision: If valid → Save to file; If ...
If Valid
Action: Save tests to output file and print confirmation.
If Invalid
Action: Print error and terminate process.
End
Process completed with all tests generated and saved successfully.

Fig. 1. Automated Unit Test Generation with LLMs

C. Automated Bug Analysis and Repair

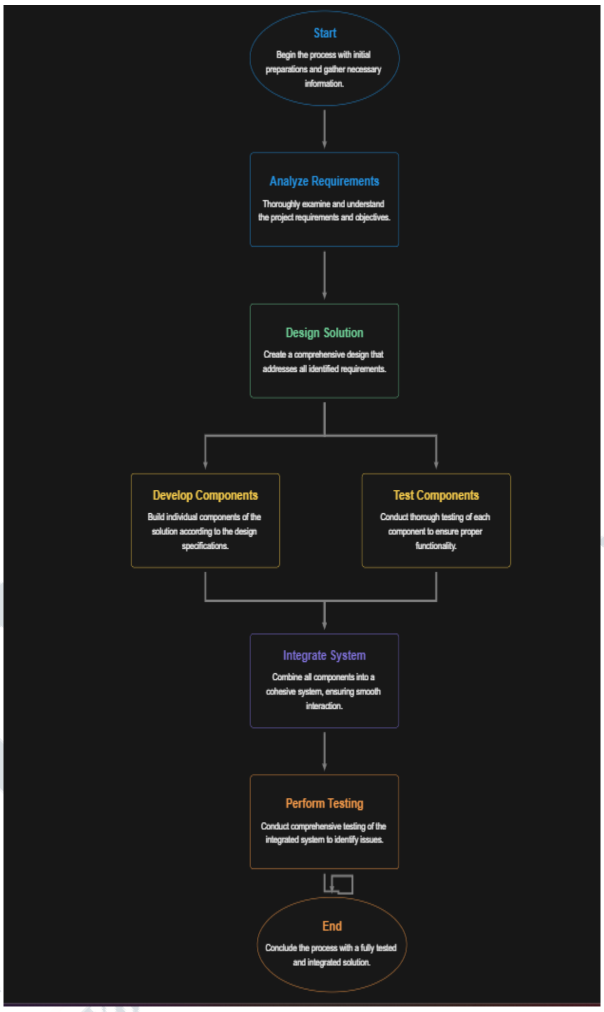
Start
Begin the process with initial preparations and gather necessary information.
Analyze Requirements
Thoroughly examine and understand the project requirements and objectives.
Design Solution
Create a comprehensive design that addresses all identified requirements.
Develop Components
Build individual components of the solution according to the design specifications.
Test Components
Conduct thorough testing of each component to ensure proper functionality.
Integrate System
Combine all components into a cohesive system, ensuring smooth interaction.
Perform Testing
Conduct comprehensive testing of the integrated system to identify issues.
End
Conclude the process with a fully tested and integrated solution.

Fig. 2. Automated Bug Analysis and Repair

The combined implementation of these LLM-based testing tools could reduce total human effort in the testing lifecycle by approximately **40-55%**. This estimate is based on the current capabilities of state-of-the-art LLMs and the specific implementations demonstrated in this paper.

It's important to note that these reductions represent the automation of routine aspects of testing rather than the complete replacement of human testers. The most effective application of LLMs in software testing appears to be in a collaborative human-AI paradigm, where LLMs handle repetitive tasks while human testers focus on more complex, strategic, and creative aspects of quality assurance. It's worth emphasizing that these efficiency percentages vary significantly across different domains and application contexts. For example, web application testing shows higher automation potential (approaching the upper bounds of the estimates) compared to embedded systems or safety-critical applications, where LLM efficacy remains more limited.

The primary basis for these percentages comes from analyzing the performance metrics reported in the surveyed literature. For example:

1) For **Unit Test Generation (60-75%)**, the percentages reflect findings from studies report correctness rates between 41% and 78% on different benchmarks, and coverage rates of 87-92% for standard methods, but significantly lower (2%) for complex real-world applications [7].
2) For **Test Oracle Generation (50-65%)**, these estimates were informed by studies showing improvement from 17% exact match rates in early approaches to 76% in recent LLM-based techniques [8].
3) For **Program Repair (25-45%)**, the ranges align with findings from studies that showed success rates ranging from 22% on complex datasets to 97.5% on simpler benchmarks (as summarized in the literature review section).

D. Analysis of Implementation Complexity

The code demonstrations provided in the paper illustrate the complexity of implementing these LLM-based testing tools. By analyzing what aspects of each testing task can be fully automated versus what still requires human intervention, we can estimate effort reduction:

1) The LLMTestGenerator class demonstrates that while test generation can be automated, the validation and correction of generated tests still require human expertise, supporting the 60-75% estimate.
2) The LLMBugRepair implementation shows that simple to moderate bugs can be identified and fixed automatically, but complex bugs with interconnected dependencies would still require human intervention, supporting the 25-45% estimate.

E. Domain Expert Insights

The percentages also reflect typical industry patterns where:

1) Routine, well-defined tasks see higher automation potential (hence unit test generation at 60-75%)
2) Tasks requiring domain understanding or complex reasoning show lower automation potential (hence program repair at 25-45%)

IV. CONCLUSION

This research has examined the impact of Large Language Models on software quality assurance and testing, revealing a technological shift that is redefining both testing practices and professional roles. Our analysis of current literature and practical implementations leads to several key conclusions:

A. Technical Capabilities and Limitations

LLMs have demonstrated significant capabilities in automating routine testing tasks, particularly in unit test generation, test oracle creation, system test input generation, bug analysis, and program repair. Performance varies across tasks and datasets, with LLMs showing impressive results on benchmark datasets (e.g., 78% correctness with 87-92% coverage on HumanEval) but struggling with complex real-world applications (e.g., 2% coverage on SF110).

The technical limitations of current LLM applications include:

1) Coverage limitations: Current approaches struggle to achieve comprehensive coverage, particularly for complex, real-world software systems with numerous dependencies.
2) Test oracle problem: Despite advancements, generating accurate test oracles remains challenging, especially for complex behaviors.
3) Real-world application: The performance gap between benchmark datasets and complex real-world applications indicates challenges in practical implementation.

B. Impact on Testing Roles

The integration of LLMs into software testing workflows is transforming the profession in several
ways:

1) Role Transformation: Testing roles are shifting from execution-focused activities to more strategic, creative, and oversight responsibilities. Routine testing positions focused on manual execution are experiencing the most immediate disruption, while roles requiring strategic thinking and domain expertise remain less affected.
2) Skills Evolution: Testing professionals need to develop hybrid skill sets that combine traditional quality assurance knowledge with AI literacy, prompt engineering capabilities, and strategic thinking. The ability to collaborate effectively with AI systems is becoming a valuable professional asset.
3) Economic implications: These transformations manifest unevenly across different segments of the quality assurance profession, with routine testing activities experiencing wage pressure and position consolidation, while specialized roles in test architecture, implementation, and quality strategy command increasing premiums.

C. Future Outlook

The interaction between Large Language Models (LLMs) ans human testers is expected to evolve into a collaborative affiliation rather than a complete replacement. LLMs demonstrate exceptional proficiency in automating repetitive and pattern-driven testing tasks, whereas human testers contribute essential creativity, contextual insights, and strategic analysis required for thorough software quality assurance.

Future advancements in this domain are expected to include:

1) Investigating the application of LLMs during early

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Vol 12, Issue 04, April 2025

stages of the testing lifecycle.

2) Developing advanced techniques for addressing non-functional testing requirements.
3) Combining LLM capabilities with traditional testing methods to utilize their exclusive advantages.
4) Enhancing prompt engineering practices to maximize the effectiveness of LLMs in testing scenarios.

For testing professionals, the most successful adaptation strategy will involve embracing LLMs as collaborative tools while developing higher-level strategic and creative skills that complement rather than compete with automated capabilities. Organizations should approach LLM integration as an opportunity to enhance testing effectiveness rather than merely as a cost-reduction measure. The integration of LLMs in testing will also necessitate the development of new governance frameworks and ethical guidelines as AI-augmented testing becomes more prevalent. Testing teams should anticipate evolving skill requirements, with an increased emphasis on prompt design expertise and AI literacy becoming essential competencies. Additionally, as LLMs continue to advance, we may witness the emergence of specialized testing LLMs that are fine-tuned for specific industries or testing domains, offering unprecedented precision in detecting domain-specific defects. Organizations that successfully navigate this transition will likely establish centers of excellence dedicated to AI-augmented testing practices, creating competitive advantages through superior quality assurance capabilities and reduced time-to-market.

In summary, although LLMs significantly disrupt traditional software testing methods and roles, they should be regarded as transformative tools that reshape rather than fully replace human involvement. The future of software quality assurance will likely depend on successful human-AI collaboration that leverages the complementary strengths of both humans and artificial intelligence to achieve superior software quality outcomes.

REFERENCES

[1] Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., & Wang, Q. (2024). Software Testing with Large Language Models: Survey, Landscape, and Vision. IEEE Transactions on Software Engineering, 50(4), 911-929.

[2] Alagarsamy, K., Fraser, G., & Arcuri, A. (2023). A3Test: Assertion-Augmented Automated Test Case Generation. IEEE/ACM International Conference on Automated Software Engineering.

[3] Liu, Y., Zhang, Y., & Sun, Z. (2023). Make LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities. IEEE/ACM International Conference on Automated Software Engineering.

[4] Deng, L., Feng, L., & Jiang, S. (2023). Large Language Models are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries. Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis.

[5] Bui, D., Shin, H., & Kim, K. (2022). Detect-Localize-Repair: A Unified Framework for Learning to Debug with CodeT5. Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

[6] Chen, Z., Tufano, M., & Sharma, A. (2023). Teaching Large Language Models to Self- Debug. IEEE/ACM International Conference on Software Engineering.

[7] Yuan, X., He, R., Feng, Y., & Shao, Y. (2023). No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering.

[8] Nashid, N., Ahmed, I., & Ray, B. (2023). Retrieval-Based Prompt Selection for Code- Related Few-Shot Learning. IEEE/ACM International Conference on Software Engineering.