International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(REVIEW ARTICLE)

Check for updates

Generative AI for software testing: Harnessing large language models for automated and intelligent quality assurance

Subham Dandotiya *

Independent Researcher, MS in Information Systems, University of Utah, Salt Lake City, Utah.

International Journal of Science and Research Archive, 2025, 14(01), 1931-1935

Publication history: Received on 14 December 2024; revised on 27 January 2025; accepted on 30 January 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.14.1.0266

Abstract

Software testing is indispensable for ensuring that modern applications meet rigorous standards of functionality, reliability, and security. However, the complexity and pace of contemporary software development often overwhelm traditional and even AI-based testing approaches, leading to gaps in coverage, delayed feedback, and increased maintenance costs. Recent breakthroughs in Generative AI, particularly Large Language Models (LLMs), offer a new avenue for automating and optimizing testing processes. These models can dynamically generate test cases, predict system vulnerabilities, handle continuous software changes, and reduce the burden on human testers. This paper explores how Generative AI complements and advances established AI-driven testing frameworks, outlines the associated challenges of data preparation and governance, and proposes future directions for fully autonomous, trustworthy testing solutions.

Keywords: Artificial Intelligence; Generative AI; Large Language Models (LLMs); Software Testing; Test Automation; Quality Assurance; DevOps

1. Introduction

Software testing plays a critical role in ensuring the quality of software before release. Yet, as systems grow in complexity—encompassing web services, mobile apps, and Internet of Things (IoT) devices—traditional testing approaches struggle to maintain adequate coverage within shortened development cycles. AI-driven solutions have already made considerable headway in addressing these limitations, especially in automated testing workflows, predictive analytics, and targeted defect detection [1].

Despite these advancements, several challenges remain. Conventional AI methods can be overfit to particular data sets or fail to adapt to rapidly changing requirements. Consequently, test suites often require extensive manual maintenance to keep pace with new features, code refactoring, and user interface overhauls [2]. Additionally, specialized machine learning (ML) techniques like neural networks, genetic algorithms, or ant colony optimization have each brought value to software testing, but they do not fully exploit natural language understanding and context-driven reasoning—key strengths offered by Large Language Models [3].

Generative AI addresses these gaps by leveraging language-based learning and text-generation capabilities to interpret specifications, generate realistic test data, and adapt oracles in real time. These models can process diverse text-based artifacts—ranging from requirements documentation to commit messages—and automatically generate or refine test scenarios. This paper explores how Generative AI refines AI-driven testing methods, highlighting best practices and governance considerations. The discussion draws from systematic reviews of AI in test automation [4], analyses of AI's

* Corresponding author: Subham Dandotiya.

impact on software testing [5], and practical use cases of machine learning–infused test automation [6], [7]. We also investigate how new governance frameworks ensure safety and reliability in scaling AI-driven solutions [8].

2. Generative AI for software testing

2.1. Evolution of AI-Driven Test Automation

AI-based solutions have profoundly influenced test automation. Early implementations utilized machine learning for tasks like defect prediction and test prioritization, notably reducing defect leakage and boosting test coverage [1]. These solutions often relied on training models on historical bug reports, code metrics, or user behavior logs. Still, many solutions were limited by the dependency on large, high-quality labeled datasets and faced difficulties when software requirements evolved swiftly.

Generative AI builds on these foundations by offering the ability to interpret unstructured information (e.g., textual specifications, user stories) and produce test artifacts that are contextually aligned with the current state of the software. Through language modeling, these systems can capture semantic nuances in requirements, user interactions, and system architectures, thereby transcending conventional AI's reliance on narrowly defined feature sets [2]. The outcome is a form of testing automation that not only executes tests but also understands and generates them in line with developer intent.

2.2. Capabilities of Large Language Models in Testing

An LLM trained on large corpuses of code, documentation, and discussion forums can learn patterns of expected behavior in software systems. It can, for instance, generate test cases that reflect real-world edge conditions—such as performance bottlenecks or atypical user journeys—without the need for intricate rule-based scripting [3]. When integrated into CI/CD pipelines, these models can automatically update and refine test suites to keep pace with incremental code changes.

In addition, LLMs can assist in test oracle creation, a task often plagued by ambiguities. By parsing design documents and code comments, Generative AI can propose logical expected outcomes for specific inputs, reducing the overhead in defining pass/fail criteria [4]. The ability to generate synthetic yet plausible data sets further helps in stress testing or security testing scenarios, especially when real production data is unavailable or must be anonymized for privacy reasons [5].

2.3. Towards Real-Time Adaptation and Continuous Testing

Continuous integration practices demand that test suites respond rapidly to new features, bug fixes, and evolving deployment environments. Generative AI accommodates this requirement in two ways. First, it can perform dynamic test generation by analyzing recent commits or merge requests, identifying code "hotspots," and proposing relevant tests. Second, it can incorporate real-time feedback from test outcomes, user telemetry, or operational metrics to improve subsequent generation cycles [6]. This interplay between generation and feedback fosters a self-evolving testing ecosystem that aligns with the DevOps emphasis on continuous feedback loops.

2.4. Comparison with Conventional AI Approaches

Machine learning models—like reinforcement learning and genetic algorithms—have been employed to optimize test coverage, prioritize test suites, and detect defects early [3]. While these approaches excel at specific tasks, they often require domain-specific features, extensive tuning, or manual updates to accommodate new functionalities. Generative AI, conversely, can interpret natural language instructions and code structures on the fly, thus reducing the upfront feature-engineering effort [7].

Furthermore, LLMs can integrate domain-specific vocabularies or rules by fine-tuning on curated repositories of code, bugs, and user requirements. This hybrid approach bridges the gap between domain-dependent AI solutions and the flexibility demanded by modern software ecosystems. As a result, Generative AI can produce test scenarios that reflect both the general best practices in software testing and the unique traits of a specific application.

3. Key challenges and best practices

3.1. Data Preparation and Model Training

A primary challenge in deploying Generative AI for testing lies in data preparation. High-quality training data—encompassing a broad set of code repositories, user stories, and historical defect logs—is essential. Yet, many organizations struggle with data availability, labeling, and cleaning. If the training corpus includes outdated or erroneous code and documentation, the model risks generating inaccurate or irrelevant test cases [3]. To mitigate these risks, practitioners should:

- Curate diverse training sets: Incorporate code from different domains (e.g., microservices, mobile apps) to produce more generalized models.
- Apply data augmentation: Use techniques that generate additional variations of user journeys or bug scenarios, thereby enriching model exposure to edge cases.
- Continuously fine-tune: Periodically retrain or fine-tune the model on recent code and defect data to capture evolving project contexts.

3.2. Complexity and Maintenance Overhead

Although Generative AI automates much of test generation, it does not remove all complexity. Teams must establish processes for reviewing, validating, and potentially overriding the AI's outputs [4]. For example, if the model produces a test script that is syntactically correct but semantically misaligned with business logic, human intervention is needed. Best practices include:

- Version control integration: Store generated tests in the same repository as source code, enabling quick reviews through standard code review tools.
- Explainability efforts: Develop interpretability layers or logs that help testers and developers understand why certain test scenarios were generated.
- Automated gating: Integrate dynamic thresholds (e.g., coverage improvement, pass rate stability) to automatically accept or reject newly generated tests.

3.3. Governance and Ethical Considerations

The adoption of AI in testing introduces ethical and regulatory concerns. Generative AI could inadvertently generate test data resembling private or sensitive information, leading to compliance risks [5]. Moreover, as LLMs become integrated into broader workflows, robust governance frameworks become essential for managing the risks of automation at scale.

A proactive governance model involves using guardrails such as scope constraints, baseline coverage requirements, and automated vulnerability checks on generated test scripts [8]. Reactive moderation tools, on the other hand, help organizations swiftly address anomalies or errors when discovered in the test automation process. A hybrid approach might fuse ongoing monitoring of AI outputs with preemptive constraints on sensitive data usage, ensuring both developer agility and product safety.

3.4. Scalability and Performance

Generative AI models, especially larger LLMs, can be computationally expensive to run. During test generation, they may require significant processing power and memory, which can slow down integration pipelines. This challenge grows as projects expand to hundreds of microservices or daily build cycles. Some best practices include:

- Model optimization: Use techniques like model pruning, quantization, or knowledge distillation to reduce resource overhead without losing essential test-generation capabilities.
- Caching and incremental generation: Cache partial model inferences so repeated queries for similar code segments do not re-trigger large-scale computations.
- Resource-aware orchestration: Dynamically scale the AI testing environment (e.g., using container orchestration or serverless architectures) based on the size of the test suite or the number of concurrent changes.

3.5. Skilled Workforce and Cross-Disciplinary Collaboration

Generative AI can significantly reduce manual effort, but it requires testers and developers to acquire new competencies in AI tooling and model interpretation. Without the right skill sets, organizations may misuse or misinterpret the model's outputs, resulting in either under-testing or over-trusting the AI's suggestions [1]. Effective integration involves:

- Training and education: Offer formal training on how the AI generates test scenarios, relevant biases, and interpretability practices.
- Cross-functional collaboration: Encourage synergy between QA engineers, data scientists, and developers. This collaboration improves the training data's quality and the relevance of generated tests.
- Feedback-driven evolution: Create a feedback loop where human experts continuously refine the AI model's performance by labeling incorrect outputs and reinforcing correct generation patterns.

4. Future directions

Generative AI's capacity to handle both structured and unstructured inputs positions it well for ongoing innovation in automated testing. Several avenues for future exploration are particularly promising:

- Self-Healing Test Suites: As code changes, test scripts often break due to changes in identifiers or system flows. LLMs can detect these changes and autonomously "heal" affected scripts by updating steps or expected outcomes on the fly [2].
- Hybrid Testing Methodologies: Combining LLM-driven test generation with specialized ML techniques—like reinforcement learning for coverage maximization—could yield robust synergy. One system could determine which areas of the application are high risk, while another generates targeted test cases [3].
- Continuous Monitoring and Adaptation: Beyond the static generation of test cases, Generative AI could integrate feedback loops from production. Observing real user behavior, performance logs, and anomalies would allow the model to update and refine the test suite continually [6].
- Ethical AI and Regulatory Compliance: Future frameworks may impose stricter regulations on AI usage in software pipelines. Incorporating traceability, fairness checks, and transparent decision-making processes into the Generative AI pipeline could become mandatory [8].
- Domain-Specific Customization: Fine-tuning LLMs to specific domains like healthcare, finance, or automotive software could improve the relevance and reliability of generated tests, as these domains come with stringent compliance and security requirements.

5. Conclusion

Generative AI stands at the forefront of a transformative shift in software testing, enabling real-time adaptation, reduced maintenance, and improved coverage. Building upon earlier AI-based approaches, it leverages natural language understanding to interpret evolving requirements and user stories, producing context-rich test suites that resonate with real-world usage. However, deploying Generative AI effectively requires organizations to address challenges in data preparation, model maintenance, governance, and workforce training. By adopting best practices—such as robust data curation, proactive guardrails, and optimized model serving—teams can integrate Generative AI into continuous testing pipelines, ultimately achieving a more agile, accurate, and trustworthy approach to quality assurance. As research evolves, especially in self-healing systems, hybrid methodologies, and regulatory compliance, Generative AI is likely to become a cornerstone of modern software testing, delivering unprecedented efficiency and reliability in complex software ecosystems.

References

[1] Nama, Prathyusha. (2024). Integrating AI in testing automation: Enhancing test coverage and predictive analysis for improved software quality. World Journal of Advanced Engineering Technology and Sciences. 13. 769–782. 10.30574/wjaets.2024.13.1.0486.

[2] Khan, Md Fokrul Islam & Mahmud, Farhad & Hossen, Arif & Masum, Abdul. (2024). A NEW APPROACH OF SOFTWARE TEST AUTOMATION USING AI. Yingyong Jichu yu Gongcheng Kexue Xuebao/Journal of Basic Science and Engineering. 21. 559-570.

[3] Sugali, Kishore, Software Testing: Issues and Challenges of Artificial Intelligence & Machine Learning (October 24, 2021). IJAIA 2021, Available at SSRN: https://ssrn.com/abstract=3948930

[4] Battina, Dhaya Sindhu, Artificial Intelligence in Software Test Automation: A Systematic Literature Review (December 12, 2019). International Journal of Emerging Technologies and Innovative Research (www.jetir.org | UGC and issn Approved), ISSN:2349-5162, Vol.6, Issue 12, page no. pp1329-1332, December-2019

[5] Khaliq Z, Farooq SU, Khan DA. Artificial intelligence in software testing: Impact, problems, challenges and prospect. arXiv preprint arXiv:2201.05371. 2022 Jan 14.

[6] Nama, Prathyusha & Meka, Harika & Pattanayak, Suprit Kumar. (2021). Leveraging machine learning for intelligent test automation: Enhancing efficiency and accuracy in software testing. International Journal of Science and Research Archive. 3. 152-162. 10.30574/ijsra.2021.3.1.0027.

[7] Yadav PS. Enhancing Software Testing with AI: Integrating JUnit and Machine Learning Techniques. North American Journal of Engineering Research. 2023 Mar 17;4(1).

[8] Dutta Gupta, Shreyam. (2024). Building Trustworthy AI: Proactive Guardrails and Reactive Moderation for Scalable Governance. International Journal of Innovative Research in Science, Engineering and Technology. 13. 10.15680/IJIRSET.2024.1312181