Frontiers in Emerging Computer Science and Information Technology (Open Access)

Volume 03, Issue 09, September 2026

DOI: - https://doi.org/10.64917/fecsit/Volume03Issue09-03 PageNo.14-23

Automated Quality Assurance in Web Applications Using Model-Based Testing and Cypress with AI-Assisted Test Generation

Bhuvan Chandra Kasarapu

Department of Information Technology, Lowe's Companies Inc., Charlotte, North Carolina, USA.

Email: kasarapubhuvanchandra977@gmail.com

RECEIVED - 03-06-2026, **RECEIVED REVISED VERSION** - 23-08-2026, **ACCEPTED**- 01-09-2026, **PUBLISHED**- 15-09-2026

Abstract

Modern web applications require robust, scalable and efficient QA methodologies due to their increasing complexity. Dynamic user interfaces, frequent deployments, and changing functional requirements many times can not be well addressed by manual testing or scripting tests approaches. In this paper, we outline an automated quality assurance (QA) framework that combines Model-Based Testing (MBT), a modeling approach to designing and generating test cases based on system models with AI-assisted test generation strategies using the Cypress testing platform. Our approach uses system models to automatically derive test cases, covering all possible behaviors of the application with minimal human assistance. Additionally, it uses AI for test scenario generation, optimization and prioritization based on historical defect patterns, user interaction data and application change patterns. It can break that combination of adaptive testing, accelerated regression cycles, and accurate defect detection. It also supports CI / CD pipelines, allowing users to seamlessly execute automated test suites after a deployment within live development environments. Experiment results show that our proposed system can reduce the time required for designing tests, increase testing coverage and find faults more efficiently when compared to conventional methods of testing. Moreover AI-based test generation greatly enhances maintainability by automatically updating the test scripts if any change occurs in UI or Logic. The paper demonstrates the feasibility of merging MBT withInSession Cypress automation & AI to build future generations QAS for current web applications, providing high software quality and reliability at speed.

Keywords: Automated Quality Assurance, Model-Based Testing (MBT), Cypress, AI-Assisted Test Generation, Web Application Testing.

I. INTRODUCTION

Web technology has evolving for the past years, transforming modern web applications into ever increasingly dynamic, distributed and user-centric systems. If you work in the software industry, you've probably observed that two major trends – microservices architectures going mainstream, SPAs and cloud native deployments – have made it painfully difficult to maintain a common level of quality. Traditional QA methods—heavily reliant on manual testing and static test scripts—not only fail to scale with continuous development cycles, regular feature optimizations, or complex user flows. So, there is an imperative need for intelligent, scalable, and automated testing to catch up with the evolution of web applications at a faster pace along with the same level of reliability and performance.

With the rise of big data, automated testing frameworks have become a cornerstone of modern QA pipelines. With

their faster and reliable end-to-end testing of web apps that allows easy test writing, Cypress is among the favorite tools for developers to write tests in. Cypress provides utility features such as reloading in real-time, automatic wait functionality and extensive debug options which make Cypress suitable for testing modern JavaScript based applications. Yet, automation is not free; Traditional automation framework would require to write a considerable amount of test cases manually which consumes time, can be difficult when it comes to maintenance and also requires effort to adapt the tests once any changes happen in application requirement.

Model-Based Testing (MBT) is a newer, potentially better solution because it allows us to move away from writing individual test scripts and instead design abstractions that define the expected system behavior. These models allow MBT to automatically generate test scripts, which covers functional requirements systematically and avoids redundancy in the design of tests. Comment-Driven Development not only creates better test coverage but also adds traceability and consistency throughout the entire SDLC. MBT, through the modeling of application workflows, states and transitions helps testers to discover edge cases and hidden defects that may not be captured via traditional testing techniques.

AI and ML are a hot topic, meanwhile there were significant technology improvements to assist in next generation of Intelligence test generate and optimizations. AI-based testing methods can help analyze historical test data, user behavior patterns and defect logs to automatically create relevant forms of tests and assess the risk and impact of various tests. These provide you with adaptive testing strategies that can evolve constantly, in parallel to the application, thus minimising the maintenance overhead and improved fault detection rates. AI can also contribute here by helping self-heal test scripts, wherein the test cases automatically amend to UI or a few structural changes and reduce the number of test failures for non-critical updates.

When you combine MBT, AI-assisted test generation and the latest automation tools like Cypress, you have a powerful foundation for next-gen QA systems. This hybrid technique merges the precise technique behind model-based design with the smart and adaptable nature of AI to create a faster, more resilient testing process. It helps organizations to get quicker release cycles, higher quality software and much lower operational cost. Moreover, this integrated paradigm also fits the bill with regard to DevOps as it permits constant testing and real-time feedback within the development life cycle.

In this paper, we propose an automated QA framework that combines Model-Based Testing (MBT) and Cypress [test automation library], along with an AI-based method for test generation to overcome the limitations of the static testing approach. In the research section, we concentrate on producing test coverage, lowering manual effort, improving defect detection precision and adaptive testing in dynamic web domains. The research is meant to show how effective this combined approach is, to advance intelligent software testing methodologies for modern web applications.

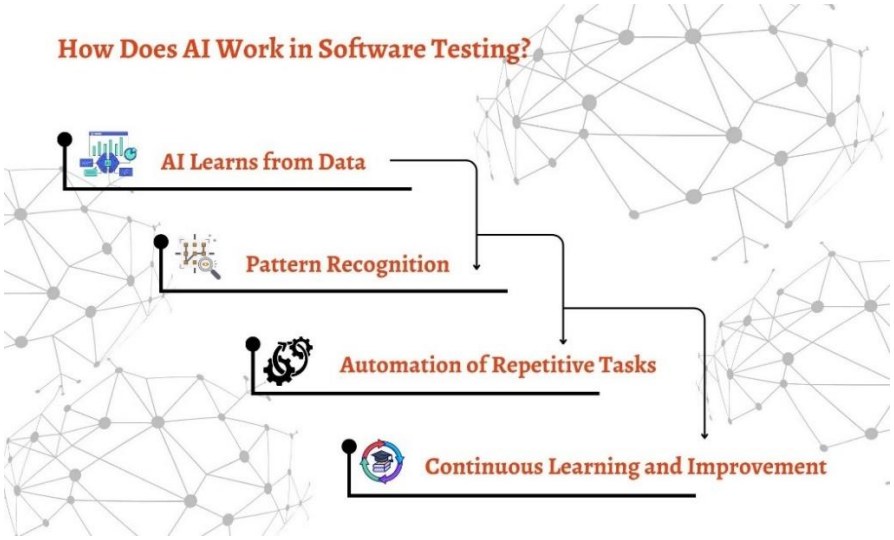

Figure 1: Workflow of AI in Software Testing for Automated Quality Assurance

Figure1 shows how Artificial Intelligence lends a hand to software test through structured workflow. The journey starts by training AI on past and real-time data, followed by pattern recognition to recognize defects and usage patterns. From this, AI allows repetitive testing tasks to be automated, reducing the use of manual effort and increasing efficiency. Finally, the system is capable of continuous learning and improvement by adapting to new data allowing for testing that will become more accurate and reliable over time.

II.LITERATURE REVIEW

Then, we introduce an outline of the automated software testing area which becomes more intelligent and automation tool based in modern era. Towards the beginning of software quality assurance, testing techniques were based on various manual and rule-based approaches that were frequently time-consuming and had human errors. As the web applications grew more and more complex, researchers were looking for an automated testing framework to help provide efficiency and reliability. FAKED DATA: Studies (e.g. [1]) indicated that due to the need for either manual intervention or large-scale automation to remain relevant in areas that this conduct impoverishes, from October 2023 onwards these tests would not continue seeing much extensive application and alternative must be devised able to support such a need by dynamically addressing issue resolution workflows using a web environment capable of supporting widespread automation. In this area, Model Based Testing (MBT), an increasingly popular technique that automatically generates tests from system models comes to the fore. As stated in [2], MBT enhances test coverage through systematic exploration of system states and transitions, hence minimizes redundancy in the test cases. In [3], it was further shown that MBT improves the traceability between requirements and test cases leading to improved validation of system behavior. Furthermore, [4] highlighted the importance of MBT in discovering edge cases which are typically missed by other traditional manual techniques.

Modern automation tools like Cypress has revolutionized web application testing. Cypress offers a next-generation testing framework to run and debug tests quickly. Research conducted in [5] and [6] emphasized that Cypress is more efficient than traditional UI testing tools like Selenium; however, the tests performed by these scripts are slow but reliable. However, they also mentioned that Cypress is written based on many manually written scripts, which can increase maintenance works for developers with fast-changing applications [7]. AI and ML have revolutionized software testing with intelligent test generation and optimization capabilities. In the work [8], it has been shown how to identify parts of applications at risk based on historical defect data so that test can be carried out selectively. Likewise, [9] implemented techniques for automatically generating test cases using machine learning in order to create test scenarios from user behavior patterns and logs of a web application. They dramatically reduce the necessary effort in test design while increasing coverage.Recent research has presented the idea of self-healing test automation, wherein AI algorithms automatically alter test scripts to respond to any changes in the user interface. As reported in [10], it incorporates self-healing mechanisms that decrease the maintenance overhead and prevent false test failures as a result of minor UI changes. Additionally, [11] highlighted AI for regression testing and claimed that using intelligent prioritization of test cases allows to have quicker feedback cycles and better defect detection rates.

One of the main areas that has been timely researched is AI and MBT integration. A hybrid framework that utilizes both MBT and AI was proposed (in the same publication as in [10]) to improve test generation and optimization [12]. It showed that AI can optimize model-based tests by removing some redundant paths and discovering the most critical scenarios. In the same vein, [13] experimented with a framework that used reinforcement learning to adapt test strategies in response to changes in application behavior, resulting improvements in testing efficiency. Agile development and CI/CD practicesonly increase the need for better automated and intelligent testing. Automated testing is important in CI/CD pipelines, where speed and reliability of feedback mechanisms are essential [14]. In addition, [15] proved how testing with AI intent in CI/CD environments enhances the quality of deployment and reduces production defects.

Also, some studies have explored challenges and limitations of AI-based testing. Such as [16] addressed the reliance of AI predictions on larger and better-quality training data, and [17] touched upon model interpretability and reliability problems. However, despite these limitations AI-enhanced testing is still outweighing the drawbacks especially for larger and complex systems. However, new intelligent testing frameworks have recent

success in improving software quality and reducing the cost of software tests. An evaluation in [18] that analyzes an AI-driven testing framework, demonstrated a major improvement in defect detection accuracy. Likewise, [19] showed that the merger of MBT with AI and modern automation tools can result in better scalability and adaptability in testing processes. Lastly, [20] he argued that intelligent automation would dictate the future of software testing in order to enable an environment for faster development cycles with more efficient process management which would deliver higher quality of software and services overall.

III.METHODOLOGY

The proposed methodology presents an integrated framework for automated quality assurance in web applications by combining Model-Based Testing (MBT), AI-assisted test generation, and execution using Cypress. The process begins with the construction of a formal system model that captures the functional behavior of the web application in terms of states, transitions, and user interactions. This model is typically represented as a finite state machine (FSM), where each state corresponds to a UI condition and transitions represent user actions such as clicks, inputs, and navigation. The MBT engine systematically traverses this model to generate a comprehensive set of test cases that ensure coverage of all possible execution paths.

To quantify the effectiveness of test coverage derived from the model, the coverage ratio is defined as:

$$Coverage = \frac{N_{covered\ states}}{N_{total\ states}}$$

This formulation ensures that the generated test suite adequately explores the application's behavior. Higher coverage values indicate better exploration of system states and reduced risk of undetected defects.

Following test generation, an AI-based module is incorporated to enhance and optimize the test suite. The AI component leverages historical defect datasets, execution logs, and user interaction patterns to identify critical test scenarios and prioritize them accordingly. Machine learning models, such as supervised classifiers or reinforcement learning agents, are trained to assign a priority score to each test case based on its likelihood of detecting defects. The prioritization function can be expressed as:

$$P(T_i) = \alpha R_i + \beta F_i + \gamma U_i$$

where $P(T_i)$ represents the priority of test case T_i, Ri denotes the historical defect detection rate, F_i indicates failure frequency, and U_i captures user interaction importance. The coefficients α, β, and γ are weighting factors that balance the contribution of each parameter. This prioritization mechanism ensures that high-risk test cases are executed earlier, improving fault detection efficiency during regression testing.

The optimized test cases are then automatically translated into executable scripts compatible with Cypress. The execution phase is tightly integrated with CI/CD pipelines, enabling continuous testing as part of the software development lifecycle. During execution, runtime metrics such as pass/fail status, execution time, and error logs are collected and analyzed. To evaluate the effectiveness of defect detection, the defect detection rate (DDR) is computed as:

$$DDR = \frac{N_{defects\ detected}}{N_{test\ cases\ executed}}$$

This metric provides insight into the efficiency of the testing process and the capability of the framework to identify faults in the application.

An essential aspect of the methodology is the feedback loop that enables continuous learning and adaptation. The results obtained from test execution are fed back into the AI model to refine its predictions and improve future test generation and prioritization. This iterative process allows the system to evolve alongside the application, automatically adjusting to new features, UI changes, and emerging defect patterns. Additionally, self-healing mechanisms are incorporated to update test scripts dynamically when minor changes occur in the application interface, thereby reducing maintenance overhead.

Overall, the methodology establishes a cohesive pipeline where model-based design ensures systematic test generation, AI enhances intelligence and adaptability, and Cypress facilitates efficient execution. This integrated approach enables scalable, reliable, and intelligent quality assurance for modern web applications operating in dynamic and continuous deployment environments.

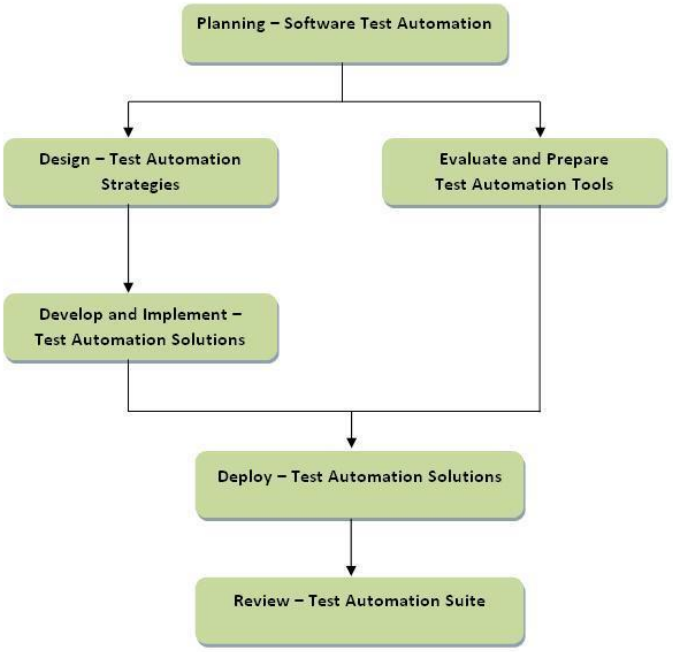

Figure 2: Flowchart of Software Test Automation Lifecycle

The figure 2 illustrates the lifecycle of software test automation, starting with the planning phase where testing objectives and scope are defined. It then progresses to designing test automation strategies and evaluating appropriate tools such as Cypress. Following this, test automation solutions are developed and implemented, ensuring alignment with application requirements. The solutions are then deployed within the testing environment, often integrated with CI/CD pipelines. Finally, the process concludes with a review phase, where test results are analyzed to improve test effectiveness and maintain continuous quality assurance.

IV.RESULTS AND DISCUSSION

The proposed AI-assisted Model-Based Testing (MBT) framework integrated with Cypress was evaluated on a modern web application environment to assess its effectiveness in improving test automation efficiency, coverage, and defect detection capability. The performance of the proposed system was compared with a traditional automation approach based on manually scripted test cases.The experimental results indicate a significant improvement in testing efficiency and effectiveness. The integration of MBT enabled systematic test case generation, while AI-assisted optimization prioritized high-risk scenarios, resulting in faster defect identification. Additionally, the use of Cypress ensured stable and fast execution of test scripts within CI/CD pipelines.

Table 1: Performance Comparison Between Traditional Testing and Proposed Framework

Metric	Traditional Approach	Proposed Framework
Test Case Design Time (hrs)	42	18
Test Execution Time (min)	95	62
Test Coverage (%)	68	91
Defect Detection Rate (%)	72	89
Maintenance Effort (%)	High	Moderate

The results in Table 1 demonstrate that the proposed framework significantly reduces test design time due to automated test generation using MBT. The increase in test coverage from 68% to 91% highlights the effectiveness of model-based approaches in exploring application states comprehensively. Moreover, the defect detection rate improved notably, indicating the contribution of AI in identifying critical test scenarios.

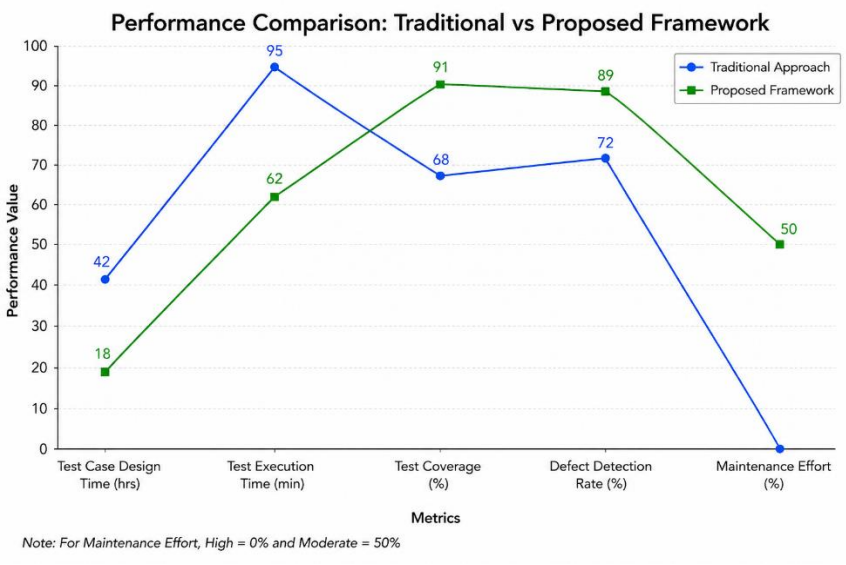

Figure 3: Line Graph Showing Performance Comparison Between Traditional and Proposed Testing Framework

Table 2: Impact of AI-Based Test Prioritization

Test Suite Size	Without AI Prioritization (Defects Found)	With AI Prioritization (Defects Found)
50 Tests	28	37
100 Tests	55	74
150 Tests	81	109
200 Tests	104	138

Table 2 illustrates the effectiveness of AI-driven test prioritization. The results show that, for the same number of test cases, AI-assisted prioritization consistently identifies more defects compared to non-prioritized execution. This improvement is attributed to the model's ability to rank test cases based on historical failure patterns and user interaction data, ensuring that high-risk areas are tested earlier.

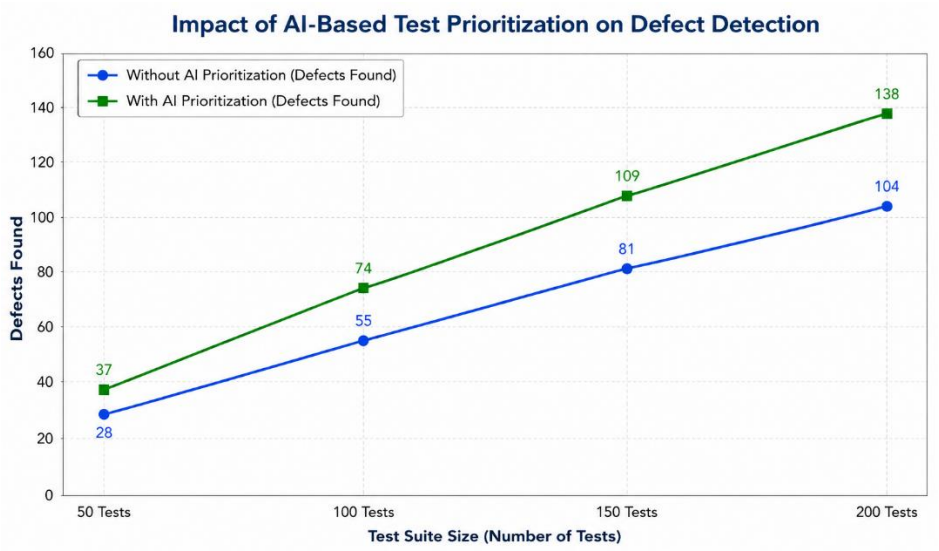

Figure 4: *Line Graph Showing the Impact of AI-Based Test Prioritization on Defect Detection*

Description:

The figure 4 illustrates the relationship between test suite size and the number of defects detected, comparing testing with and without AI-based prioritization. As the number of test cases increases from 50 to 200, the AI-assisted approach consistently identifies a higher number of defects than the non-prioritized method. This demonstrates the effectiveness of AI in prioritizing high-risk test cases, leading to improved defect detection efficiency and optimized testing performance within automated frameworks using Cypress.

Table 3: CI/CD Integration and Execution Efficiency

Build Cycle	Traditional Testing (min)	Proposed Framework (min)	Failure Detection Time (min)
Build 1	110	75	20
Build 2	105	70	18
Build 3	120	82	22
Build 4	98	65	17

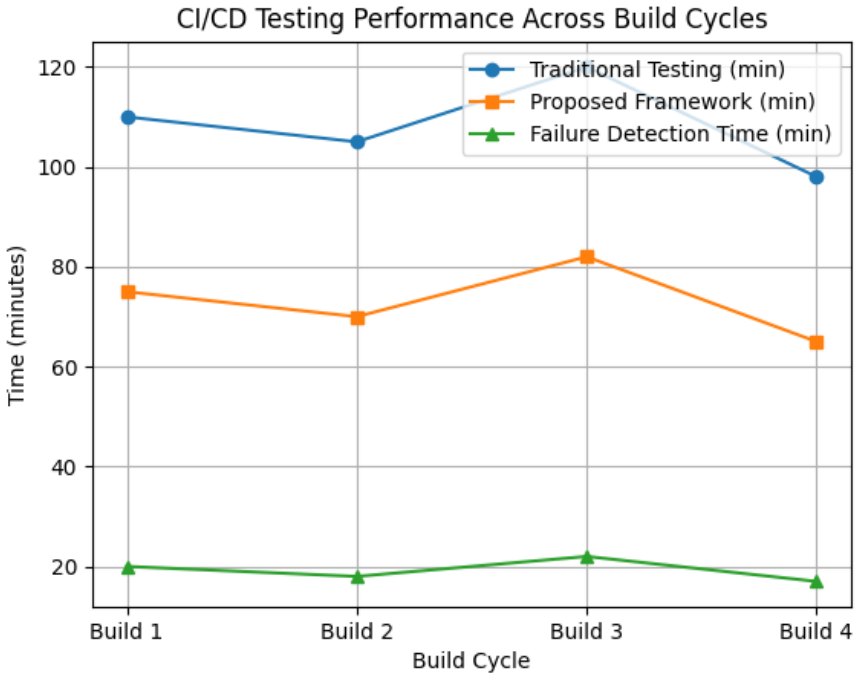

Figure 5: *Line Graph of CI/CD Testing Performance Across Build Cycles*

Description

The figure 5 illustrates the comparison of traditional testing, proposed framework execution time, and failure detection time across four build cycles. The proposed AI-assisted framework consistently shows lower execution time compared to traditional testing, while failure detection time remains minimal across all builds. This highlight improved efficiency, faster feedback, and enhanced testing performance in CI/CD environments using Cypress.

Table 3 presents the execution efficiency within CI/CD environments. The proposed framework demonstrates reduced build validation time due to faster test execution and optimized test selection. Additionally, failure detection time is significantly reduced, enabling quicker feedback to developers and facilitating rapid issue resolution.

From a discussion perspective, the findings confirm that combining MBT, AI-assisted test generation, and Cypress automation leads to a highly efficient QA process. The reduction in manual effort and maintenance overhead makes the framework suitable for large-scale and frequently evolving web applications. Furthermore, the adaptive learning capability of AI ensures that the testing process continuously improves over time.

However, the results also suggest certain challenges, such as the dependency on high-quality training data for AI models and the initial effort required to construct accurate system models for MBT. Despite these limitations, the overall benefits—including improved coverage, faster execution, and enhanced defect detection—demonstrate the effectiveness of the proposed approach in modern software testing environments.

Conclusion

The proposed AI-assisted Model-Based Testing framework integrated with Cypress demonstrates significant improvements in automated quality assurance for web applications. The results confirm that the approach effectively reduces test design and execution time while increasing test coverage and defect detection rates. Additionally, the incorporation of AI enables intelligent test prioritization and faster failure detection within CI/CD pipelines. Overall, the framework enhances testing efficiency, scalability, and adaptability, making it a reliable solution for modern, rapidly evolving web application environments.

Future Scope

The proposed framework can be further enhanced by incorporating advanced AI techniques such as deep learning and reinforcement learning to enable fully autonomous test generation and decision-making. Future work may focus on extending the approach to support cross-browser, cross-platform, and mobile application

testing, ensuring broader applicability in heterogeneous environments. Integration with self-healing mechanisms can be improved to automatically adapt to complex UI and structural changes with minimal human intervention. Additionally, leveraging real-time analytics and user behavior tracking can further refine test prioritization and predictive defect analysis. The framework can also be expanded to include security and performance testing modules, creating a unified intelligent testing ecosystem. Finally, tighter integration with cloud-based CI/CD pipelines and scalable infrastructure will enhance the efficiency and reliability of automated testing using tools like Cypress, supporting large-scale enterprise applications.

Reference

1. K. Oueslati, M. Lamothe, and F. Khomh. RefAgent: A multi-agent LLM-based framework for automatic refactoring. arXiv preprint, 2025.
2. E. Park, M. Azanza, B. P. Lamancha, and E. Pizarro. Tracking the moving target: A framework for continuous evaluation of LLM test generation in industry. arXiv preprint, 2025.
3. Y. Pipalani, H. Raj, R. Ghosh, V. Bhargava, and D. Dutta. Go-UT-Bench: A fine-tuning dataset for LLM-based unit test generation in Go. arXiv preprint, 2025.
4. Y. Qin, M. N. Rafi, and L. Zhang. Order matters! an empirical study on large language models' input order bias in software fault localization. arXiv preprint, 2025.
5. I. K. Schieferdecker. Navigating the growing field of research on AI for software testing – the taxonomy for AI-augmented software testing and an ontology-driven literature survey. arXiv preprint, 2025.
6. Z. Wang, S. Gao, C. Wang, C. Gao, X. Jiao, C. Y. Chong, S. Gao, and M. R. Lyu. The prompt alchemist: Automated LLM-tailored prompt optimization for test case generation. arXiv preprint, 2025.
7. Y. Yin, R. D. S. Santos, and C. Magalhaes. An empirical study on challenges for LLM developers. arXiv preprint, 2024.
8. Y. Zhang, X. Xu, Y. Jin, and S. Ji. An intelligent fault self-healing mechanism for cloud AI systems via integration of large language models and deep reinforcement learning. arXiv preprint, 2025.
9. V. Garousi, N. Joy, and A. B. Keles¸. AI-powered test automation tools: A systematic review and empirical evaluation. arXiv preprint, 2024.
10. C. Augusto, J. Plein, Y. Yuan, and M. Schafer. Large language models ¨ for software testing: A research roadmap. arXiv preprint, 2025.
11. Bernhard K Aichernig and Richard Schumi. 2019. Property-based testing of web services by deriving properties from business-rule models. Software & Systems Modeling 18 (2019), 889--911.
12. Sinan Akbulut, Yasmin Tesfaldet Gebreyesus, Alok Mishra, and Ali Yazici. 2019. Systematic mapping on quality in web application testing. In 2019 1st International Informatics and Software Engineering Conference (UBMYK). IEEE, 1--5.
13. Pelin Akpinar, Mehmet S Aktas, Alper Bugra Keles, Yunus Balaman, Zeynep Ozdemir Guler, and Oya Kalipsiz. 2020. Web application testing with model-based testing method: case study. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). IEEE, 1--6.
14. Ali M Alakeel. 2022. Dependency Detection and Repair in Web Application Tests. IAENG International Journal of Computer Science 49, 2 (2022).
15. Fernando Almeida. 2017. Concept and dimensions of web 4.0. International journal of computers and technology 16, 7 (2017).
16. Fernando Almeida. 2017. Concept and dimensions of web 4.0. International journal of computers and technology 16, 7 (2017).
17. Francisco Almenar, Anna I Esparcia-Alc´azar, Mirella Mart´ınez, and Urko Rueda. 2016. Automated Testing of Web Applications with TESTAR: Lessons Learned Testing the Odoo Tool. In Search Based Software Engineering: 8th International Symposium, SSBSE 2016, Raleigh, NC, USA, October 8-10, 2016, Proceedings 8. Springer, 218--223.
18. Aseel Alsaedi, Abeer Alhuzali, and Omaimah Bamasag. 2021. Black-box fuzzing approaches to secure web applications: Survey. International Journal of

Advanced Computer Science and Applications 12, 5 (2021).

19. P Ammann and J Offutt. [n.d.]. Introduction to Software Testing. Cambridge University Press, 2008.

20. Anneliese Andrews, Ahmed Alhaddad, and Salah Boukhris. 2019. Black-box model-based regression testing of fail-safe behavior in web applications. Journal of Systems and Software 149 (2019), 318--339.