IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

AI-Driven Adaptive Test Automation Framework with Self-Healing and Predictive Capabilities

Atul Kumar Jha, Avinash Kumar, Kushagra Sinha, Prateek Bohra, Prof. Prashanth J
Assistant Professor, Department of Information Science and Engineering
Global Academy of Technology, Bengaluru, Karnataka, India
atulkumarjha88@gmail.com, avinashkumar1224@gmail.com, kushagrasi1301@gmail.com,
erprateekbohra@gmail.com, prashanthj@gat.ac.in

Abstract: *The continuous acceleration of software delivery pipelines mandates highly resilient validation mechanisms. However, UI-centric web automation remains fundamentally hindered by structural Document Object Model (DOM) instability, a vulnerability frequently termed locator brittleness. While contemporary paradigms propose the integration of Large Language Models (LLMs) for dynamic script remediation, such generative architectures introduce prohibitive latency and computational bloat, rendering them misaligned with high-velocity agile environments. This paper proposes a lightweight, hybrid test automation ecosystem that mitigates locator fragility while optimizing execution pathways. Operating on a JavaScript-based orchestration backend (Node.js/Express) coupled with a Python-powered analytical engine, the framework establishes a dual-layered defense against test failure. First, a deterministic, rule-based fallback heuristic ensures instantaneous, zero-latency element recovery during live Selenium execution. Second, a Logistic Regression model processes historical NoSQL execution metadata to calculate real-time failure probabilities, dynamically prioritizing high-risk automated checks. The proposed architecture significantly minimizes false-positive alerts, conserves server compute cycles, and provides transparent, actionable diagnostics via an interactive React.js interface*

Key Words: Adaptive Automation, Continuous Integration, Logistic Regression, MERN Architecture, Predictive Analytics, Software Quality Assurance, DOM Resilience

1. INTRODUCTION

The paradigm shift toward hyper-agile software engineering has entrenched Continuous Integration and Continuous Deployment (CI/CD) as the foundational backbone of modern development. Within these pipelines, automated web testing serves as the primary gatekeeper for software quality. Yet, frameworks like Selenium WebDriver suffer from an inherent architectural flaw: a strict dependency on static UI locators (e.g., CSS Selectors, XPaths). Routine aesthetic or structural updates to web interfaces inadvertently invalidate these locators, triggering cascading test suite failures even when the core business logic remains entirely intact.

To resolve this fragility, the software testing domain has increasingly pivoted toward Artificial Intelligence (AI). Recent explorations favor deep learning networks and generative AI to autonomously synthesize new locator paths. While theoretically sound, these models function as opaque processing entities (black boxes) and impose severe execution delays due to heavy remote API interactions. In enterprise-scale environments where thousands of validation checks run concurrently, such latency is a critical operational bottleneck.

This study presents a highly optimized, explainable, and adaptive test automation framework designed to reconcile the demand for script resilience with the necessity for execution speed. Deviating from resource-intensive generative models for real-time healing, this system relies on a localized, deterministic heuristic to bypass DOM mutations instantaneously. Concurrently, it addresses server inefficiencies by employing a lightweight mathematical classifier —

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

Logistic Regression — to analyze historical execution logs, structurally prioritizing tests that exhibit the highest probabilistic risk of failure.

A. Problem Statement

Existing test automation infrastructures are plagued by high maintenance overhead due to UI volatility. Current AI-integrated healing solutions impose unacceptable execution delays in CI/CD pipelines, while the indiscriminate execution of massive, unoptimized test suites wastes critical server computing resources. There is an urgent requirement for a high-speed, transparent testing framework that combines rapid script recovery with intelligent, data-driven execution prioritization.

B. Objectives

- **Zero-Latency Resilience:** Engineer a sequential, in-memory fallback mechanism capable of recovering compromised DOM interactions without halting the underlying automation engine.
- **Intelligent Execution Routing:** Develop a Python-based machine learning pipeline utilizing Logistic Regression to evaluate historical MongoDB test telemetry, enabling the real-time prioritization of highly volatile test cases.
- **Full-Stack Diagnostic Visibility:** Construct a scalable Node.js and React.js ecosystem to orchestrate test deployments and visualize predictive health metrics for quality assurance analysts.

II. RELATED WORK

The trajectory of automated testing has evolved from brittle, sequential scripts to increasingly adaptive ecosystems. However, the integration of intelligence often compromises execution velocity.

Early algorithmic interventions utilized deep autoencoders and Reinforcement Learning (RL) agents to mimic human navigational patterns during testing. While these strategies achieve high coverage, they are burdened by the "Cold Start" dilemma — requiring massive historical datasets before yielding accurate predictions. Furthermore, their black-box nature discourages widespread enterprise adoption, as quality engineers cannot easily trace the logic behind a generated test step.

Our proposed methodology is fundamentally shaped by evaluating the trade-offs present in two specialized research domains:

- **Mechanisms of Execution Recovery:** Recent comparative analyses contrasting LLMs against rule-based methodologies for Selenium test stabilization indicate a stark divergence in utility. LLMs demonstrate exceptional capability in synthesizing novel locators but suffer from severe network and inference latency. Conversely, deterministic rule-based systems — which iterate through a localized array of predefined fallback attributes — achieve near-identical recovery rates for standard UI mutations while executing in milliseconds. Our architecture strictly adopts this heuristic approach to preserve pipeline velocity.
- **Predictive Suite Optimization:** Research investigating ML-driven test optimization validates the efficacy of deploying shallow classification networks. Studies confirm that extracting temporal and historical features from CI/CD telemetry significantly improves fault detection rates. By implementing Logistic Regression, which is mathematically tailored for binary outcomes (Pass/Fail), our framework dynamically isolates fragile scripts without incurring the computational tax of complex ensemble models.

III. METHODOLOGY

The system is conceptualized as a decoupled, multi-tier architecture, ensuring independent scalability across data ingestion, predictive modeling, and automation execution.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

A. Telemetry Ingestion and Orchestration (Node.js & MongoDB)

The operational core of the ecosystem is a Node.js runtime environment leveraging the Express.js framework. This layer acts as the primary orchestrator, managing incoming test requests and governing API traffic. As test suites execute, multi-dimensional metadata — encompassing execution duration, temporal timestamps, DOM state snapshots, and ultimate binary outcomes — is ingested into a MongoDB database. The schema-less nature of this NoSQL repository seamlessly accommodates highly variable telemetry structures.

B. Predictive Threat Engine (Python)

Prior to the initialization of a testing cycle, the execution queue must be optimized. A specialized Python microservice interfaces with the MongoDB cluster to extract consolidated historical feature vectors. Given that software validation results are intrinsically binary, the system deploys a Logistic Regression algorithm. This classifier assesses the historical fragility footprint of each script, calculating a precise failure probability score, thereby reordering the queue to execute vulnerable tests immediately.

C. Adaptive Execution Module (Selenium WebDriver)

The optimized queue is subsequently dispatched to the Selenium execution engine. To neutralize locator brittleness, all native UI interactions (e.g., clicks, text inputs) are encapsulated within a dynamic error-handling wrapper. Should the primary locator trigger a NoSuchElementException, the execution does not terminate. Instead, the heuristic engine instantaneously traverses a prioritized matrix of secondary locators (e.g., CSS IDs, DOM hierarchies, partial text strings). Upon a successful tertiary interaction, the anomaly is flagged as "Healed" within the analytics payload, and the execution proceeds unhindered.

IV. MATHEMATICAL FORMULATION

The predictive intelligence governing test prioritization relies on Logistic Regression, a robust statistical method for binary classification problems where the objective is to predict the probability of a dichotomous outcome. In our framework, we model the probability of a specific test case failing based on its historical execution telemetry.

Let $y \in \{0, 1\}$ represent the binary response variable, where $y = 1$ denotes a test failure or required healing, and $y = 0$ denotes a clean pass.

Let $x = [x_1, x_2, \ldots, x_n]^T$ represent the feature vector comprising historical test metrics extracted from MongoDB, such as average execution duration, recent failure rate, and historical self-healing frequency.

The core operation of the model calculates a linear predictor, denoted as z, which is a weighted sum of the input features:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = \beta_0 + \beta^T x \quad (1)$$

where β_0 is the intercept term and β is the vector of learned coefficients representing the impact of each feature on the likelihood of failure.

To map this linear combination to a valid probability distribution between 0 and 1, the linear predictor z is transformed using the standard logistic (sigmoid) function. Thus, the probability p that a test case will fail ($y = 1$) given the input features x is defined as:

$$P(Y = 1 \mid x) = 1 / (1 + e^{-(\beta_0 + \beta^T x)}) \quad (2)$$

During the training phase, the optimal parameter values (β_0, β) are determined using Maximum Likelihood Estimation (MLE), effectively minimizing the Log-Loss cost function over the n historical samples in the dataset:

$$\text{Log-Loss} = -(1/n) \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (3)$$

where $p_i = P(Y = 1 \mid x_i)$.

During execution routing, any test case where the calculated predicted probability $P(Y = 1 \mid x)$ exceeds a dynamically configurable risk threshold (typically > 0.5) is classified as high-risk and aggressively prioritized at the apex of the execution queue.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 1, November 2026
Impact Factor: 8.2

V. DISCUSSION

The implemented ecosystem effectively circumvents the standard operational dichotomy between structural interpretability and execution velocity. By consciously excluding massive deep neural networks from the live testing pathway, the localized fallback heuristic successfully processed continuous DOM mutations without triggering script timeouts or pipeline bottlenecks. Transitioning the complex decision-making to the pre-execution phase — via the Logistic Regression engine — proved highly effective in optimizing server loads, ensuring that volatile components are validated first.

Furthermore, translating raw automation logs into the React-based visual dashboard transforms the testing pipeline from a rigid execution channel into an explainable, data-rich diagnostic tool, empowering developers to permanently rectify chronically brittle UI components.

VI. FUTURE SCOPE

Future enhancements to this architecture will focus on expanding the scope of its predictive capabilities:

- **Visual Regression Integration:** Augmenting the framework with pixel-based comparison algorithms to detect superficial CSS anomalies alongside structural DOM failures.
- **Automated Code Patching:** Developing a reverse-feedback loop that not only heals the script during runtime but automatically commits the successful fallback locator back into the central source code repository.
- **Multi-Browser Telemetry:** Scaling the Logistic Regression engine to account for browser-specific rendering variances across Chromium and WebKit environments.

VII. CONCLUSION

This paper introduced a highly scalable, AI-driven adaptive test automation framework explicitly engineered to resolve locator brittleness without sacrificing CI/CD pipeline velocity. By orchestrating a robust Node.js backend alongside a Python-driven ML microservice, the system seamlessly blends predictive execution routing with deterministic, zero-latency self-healing. The application of Logistic Regression ensures lightweight, mathematically rigorous threat prioritization, while the localized rule-based fallback array eliminates the excessive overhead associated with generative AI models. Ultimately, this framework delivers a transparent, resource-efficient solution that drastically improves the reliability and maintainability of enterprise software validation protocols.

REFERENCES

[1] S. Silva, “Reviewing and Proposing Approaches for Self-adaptive Testing in the Field,” 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), 2023.
[2] S. C. P. Saarathy, S. Bathrachalam, and B. K. Rajendran, “Self-Healing Test Automation Framework using AI and ML,” International Journal of Strategic Management, vol. 3, no. 3, pp. 45-77, 2024.
[3] M. Nass, E. Alégroth, R. Feldt, M. Leotta, and F. Ricca, “Similarity-based Web Element Localization for Robust Test Automation,” ACM Trans. Softw. Eng. Methodol., 2023.
[4] M. Bagherzadeh, N. Kahani, and L. Briand, “Reinforcement Learning for Test Case Prioritization,” IEEE Transactions on Software Engineering, vol. 48, no. 8, pp. 2836, 2022.
[5] A. Sharif, D. Marijan, and M. Liaaen, “DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration Testing,” 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2021.
[6] K. Schneid, S. Thöne, and H. Kuchen, “Modification-Impact based Test Prioritization for Process-Driven Applications,” 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2023.
[7] X. Chang et al., “A Reinforcement Learning Approach to Generating Test Cases for Web Applications,” 2023 IEEE/ACM International Conference on Automation of Software Test (AST), 2023.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 1, November 2026 Impact Factor: 8.2

[8] J. C. Alonso, A. Martin-Lopez, S. Segura, J. M. García, and A. Ruiz-Cortés, "ARTE: Automated Generation of Realistic Test Inputs for Web APIs," IEEE Transactions on Software Engineering, vol. 49, no. 1, 2023.

[9] N. Tihanyi, R. Jain, M. A. Ferrag, Y. Charalambous, and L. C. Cordeiro, "A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification," 2025 IEEE/ACM International Conference on Automation of Software Test (AST), 2025.

[10] S. M. Saleh, "DevSecLogs: AI-Powered, Tamper-Evident Log Intelligence for Secure CI/CD Pipelines," 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2025.

[11] M. Havakeshian, Y. Labiche, S. Nejati, S. Desjardins, and K. Haghighi, "Test Cost Reduction for 5G and Beyond using Machine Learning," 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2023.

[12] K. Singh and S. K. Singh, "Web Automation Testing in Industry: Lifecycle, Framework Challenges, and the Future with AI," 2025 Seventeenth International Conference on Contemporary Computing (IC3), 2025.

[13] S. Sharma and S. V. Chande, "Optimizing test case prioritization using machine learning algorithms," Journal of Autonomous Intelligence, vol. 6, no. 2, 2023.

[14] D. R. Seelam, "Automated Test Case Generation using Machine Learning," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 4, no. 3, 2024.

[15] V. S. P. Karnam, "Leveraging AI for Intelligent Decision-Making in Test Automation Engineering," Journal of Computer Science and Technology Studies, 2025.

[16] C. B. C and C. B. Ram, "Generative AI for Self-Healing Selenium Tests: Comparative Evaluation of Multiple Open-Source LLMs and Rule-Based Methods," 2025 International Conference on Emerging Computation and Information Technologies (ICECIT), 2025.

[17] A. Mehmood, Q. M. Ilyas, M. Ahmad, and Z. Shi, "Test Suite Optimization Using Machine Learning Techniques: A Comprehensive Study," IEEE Access, 2024.

[18] T. O. Olaleye, O. T. Arogundade, S. Misra, A. Abayomi-Alli, and U. Kose, "Predictive Analytics and Software Defect Severity: A Systematic Review and Future Directions," Scientific Programming, 2023.

[19] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, "Empirically evaluating flaky test detection techniques combining test case rerunning and machine learning models," Empirical Software Engineering, vol. 28, 2023.

[20] T. Hunko, "Adaptive Approaches to Software Testing with Embedded Artificial Intelligence in Dynamic Environments," International Journal of Current Science Research and Review, vol. 8, no. 5, 2025.