iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VII **Month of publication:** July 2026

DOI: https://doi.org/10.22214/ijraset.2026.84120

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VII July 2026- Available at www.ijraset.com

Intelligent Web Application Automated Testing Framework with AI-Assisted Test Case Optimization

Amireddy Sainath Reddy[1], N Naveen Kumar [2]
[1]*M.Tech Scholar, Dept of CSE, JNTUH UCESTH, Hyderabad, India*
[2]*Associate Professor, Dept of CSE, JNTUH UCESTH, Hyderabad, India*

Abstract: ***Modern software development demands rapid delivery with continuous integration, making intelligent testing essential for maintaining quality. Traditional automation tools like Selenium and JUnit execute all test cases without prioritization, resulting in resource inefficiency and longer testing cycles. To address this, AI-based intelligent frameworks are emerging that dynamically optimize test execution by predicting critical areas and prioritizing impactful tests based on historical results.***
The project aims to design and implement an intelligent web application testing framework that automates functional testing while learning from past results to enhance efficiency. It employs AI-based models to prioritize and optimize test case execution, provides visual analytics on performance trends, and supports seamless integration with CI/CD tools like Jenkins and GitHub Actions. This framework offers developers and testers a scalable, adaptive, and data-driven solution for smarter and faster software testing.
Existing frameworks like Selenium, JUnit, and TestNG offer robust automation but lack intelligent test prioritization. Key limitations include redundant execution of stable tests, extended testing time, high maintenance overhead, absence of AI-driven optimization, and limited analytical insights. These gaps highlight the need for AI-powered automation solutions that enhance efficiency, reduce redundancy, and support data-driven test management.
The proposed framework integrates AI-assisted test case optimization with automated web testing. Key features include automated Selenium-based test execution, AI-driven prioritization using historical data, dynamic adjustment of test order, a result visualization dashboard, and seamless CI/CD integration. This approach reduces redundant tests, enhances coverage, and accelerates feedback, enabling faster and more reliable software releases.
The framework is implemented using Python with Selenium for automated web testing and scikitlearn, TensorFlow, and PyTorch for AI-driven test case optimization. Historical test execution data is collected and preprocessed to train machine learning models such as Random Forest and Logistic Regression to predict test case failures and prioritize execution. Dynamic prioritization reorders tests based on recent results and code changes. Visualization dashboards developed with Plotly or Dash display pass/fail ratios, execution time, and optimization outcomes. The system integrates with CI/CD pipelines like Jenkins and GitHub Actions to enable continuous testing and rapid feedback in web application development.
Keywords: ***Intelligent Software Testing , Automated Test Case Prioritization , Machine Learning-Based Testing, Selenium Web Automation, Test Case Optimization.***

I. INTRODUCTION

The increasing adoption of Agile software development methodologies, Continuous Integration (CI), and Continuous Delivery (CD) practices has fundamentally transformed the software engineering landscape. Modern software systems are expected to evolve rapidly through frequent feature enhancements, bug fixes, security updates, and performance improvements. While these practices accelerate software delivery, they also introduce significant challenges in maintaining software quality and reliability. Consequently, regression testing has become a critical activity that must be executed repeatedly throughout the software development lifecycle to ensure that newly introduced changes do not adversely affect existing functionalities.

Automated testing frameworks such as Selenium, JUnit, and TestNG have emerged as indispensable tools for supporting large-scale software validation. These frameworks enable the execution of extensive regression test suites with minimal human intervention, thereby reducing manual effort and improving testing consistency.

However, despite their widespread adoption, conventional automation frameworks continue to rely on static execution strategies in which all test cases are executed sequentially regardless of their historical behavior, business criticality, or probability of failure. As software systems grow in complexity, such approaches often lead to prolonged execution cycles, delayed feedback, inefficient utilization of computational resources, and increased maintenance overhead.

Recent advancements in Artificial Intelligence (AI) and Machine Learning (ML) have created new opportunities for transforming traditional software testing practices into intelligent and adaptive processes. By leveraging historical execution data, machine learning models can identify patterns associated with software failures and predict the likelihood of future defects. This capability enables the prioritization of high-risk test cases, thereby improving fault detection efficiency and accelerating feedback during development activities. Motivated by these advancements, this research proposes an Intelligent Web Application Automated Testing Framework with AI-Assisted Test Case Optimization that integrates predictive analytics, automated testing, and continuous integration technologies within a unified framework. The proposed approach aims to improve regression testing effectiveness, reduce testing costs, and support data-driven software quality assurance in modern development environments.

II. LITERATURE REVIEW

A. Development of a Web-Based Insurance Management System

The study was centered on the enhancement of insurance operations by developing a Web based management platform. The authors found that a number of insurance companies were still relying on conventional record-keeping and processing systems for customer requests, which often led to delays and higher administrative workloads. To address these concerns, a single online system was developed to control customer information, policy records, premium details, as well as activities related to claims. The proposed solution allowed customers to receive insurance services through the internet, and not have to go to branches often. The system allowed users to access policy information, make updates to their personal information, and monitor the status of their policy as needed. A centralized database made it easier to manage information and ensured accurate record keeping. The study found that the web-based platform helped to streamline operations and enhance the customer experience. The authors also pointed out that ongoing maintenance and good security practices are required to provide long-lasting reliability and security of sensitive data.

B. Cloud Computing Approaches for Insurance Information Management

This research explores the potential of cloud technologies in the modern insurance service. The researchers found that processing huge amount of insurance-related data with the traditional infrastructure can be costly and sometimes difficult to manage. To address these shortcomings, they proposed a cloud environment that would enable the insurance activities to be carried out in a more flexible and scalable way.In the proposed architecture, customers data, policy records and claim information were kept on cloud servers and services were provided to authenticated users from different locations as and when needed. The cloud-based solution enhanced data accessibility and streamlined backup and recovery procedures. The study identified that organizations can increase the scope of their services more effectively with a lesser amount of investments in infrastructure. The results showed that the system had a number of operational benefits but the researchers noted the need to have a proper security system to ensure customer privacy and data confidentiality.

C. Information Secure Database Management for Insurance Applications

The objective of this research was to strengthen the management of insurance information through a secure database framework. Insurance organizations generate and maintain large amounts of data related to customers, policies, premium transactions, and claims. Managing this information efficiently while maintaining security is a significant challenge.To address this requirement, the researchers designed a centralized database environment that organized all insurance-related information in a structured manner. The proposed framework improved data consistency, reduced duplication, and allowed faster retrieval of records when required. Security was treated as a key component of the system. Authentication mechanisms, controlled access privileges, and data protection techniques were incorporated to prevent unauthorized access. The study concluded that an effective database management system is essential for maintaining operational efficiency and ensuring the reliability of insurance services.

D. Automated Claim Processing in Insurance Management Systems

The focus of this research was on automating the claim settlement process. The traditional approach to claim processing may include several manual steps, a lot of paperwork, and a lengthy wait time, impacting both the efficiency of the process and the satisfaction of the customers. The researchers suggested an automated claim management framework to make these activities easier.

The system provided for the policyholders to file claims electronically and submit supporting documents via the Internet. Policy information and the completeness of the information submitted was automatically verified before further processing was done by automated validation procedures. The system eliminated manual processing, thus minimizing processing errors and making the processing more efficient. The outcomes showed that automating claims reduced the settlement duration and gave the customers better visibility on the status of the claims. The authors recommended that the automated claim processing can greatly enhance the quality of service and ease the burden of administration.

III. PROPOSED SYSTEM

This research proposes an Intelligent Web Application Automated Testing Framework that integrates automated functional testing, machine learning-based prediction, and continuous integration technologies to improve the effectiveness of regression testing activities. The primary objective of the framework is to overcome the limitations of conventional test automation approaches by introducing predictive intelligence into the test execution process. Unlike traditional testing environments where all test cases are executed sequentially irrespective of their historical behavior, the proposed framework utilizes historical execution knowledge to identify failure-prone test cases and optimize their execution order. The framework establishes a data-driven testing environment capable of adapting to changing software conditions and supporting efficient software quality assurance practices.

A. AI-Assisted Test Optimization Framework

The proposed system incorporates a robust automated testing infrastructure designed to validate web application functionality across multiple operational scenarios. Automated test execution is performed using Selenium WebDriver, which enables simulation of user interactions and verification of application behavior under various conditions. The framework systematically executes regression test suites and records detailed execution information, including execution status, runtime characteristics, failure occurrences, and module-specific outcomes.The automation layer serves as the foundation of the framework by generating reliable execution data required for predictive analysis. Through continuous execution and monitoring, the system establishes a comprehensive repository of testing information that reflects the evolving behavior of the software under test. This approach minimizes manual intervention, improves testing consistency, and provides a scalable mechanism for validating software functionality throughout the development lifecycle.

1) Automated Test Execution

The automated test execution module is responsible for validating web application functionality through browser-based testing.
Selenium WebDriver is used to simulate user interactions and execute predefined test scenarios across multiple application modules. The framework automatically performs regression testing activities and records execution outcomes including pass/fail status, execution duration, and defect information. The automation process reduces manual effort, improves testing consistency, and enables frequent validation of software functionality throughout the development lifecycle. Continuous execution also generates valuable testing information that serves as the foundation for intelligent optimization activities.

2) Historical Execution Analysis

The framework maintains a centralized repository of historical execution records collected from previous testing cycles.
This repository stores execution outcomes, runtime statistics, failure occurrences, module information, and testing trends observed over time. Historical analysis helps identify recurring failure patterns and unstable components within the application. The accumulated testing information provides valuable insights that support predictive modeling and enable data-driven decision-making during regression testing activities.

3) Machine Learning-Based Failure Prediction

The proposed framework incorporates machine learning techniques to predict potential test failures before execution begins.
Historical execution data is preprocessed and transformed into meaningful features that represent testing behavior and execution characteristics. Predictive models such as Random Forest and Logistic Regression analyze these features to estimate the probability of future failures. The generated prediction scores enable the framework to identify high-risk test cases and support intelligent testing decisions based on historical evidence.

4) Intelligent Test Case Prioritization

The test prioritization module utilizes prediction results generated by machine learning models to dynamically optimize test execution order.

Each test case is assigned a risk score according to its historical behavior and predicted failure probability.Test cases with higher risk scores are executed before lower-risk alternatives, allowing critical defects to be discovered earlier in the testing cycle. This adaptive prioritization strategy improves fault detection efficiency while reducing unnecessary execution of stable test cases.

5) Visualization and Reporting

The framework provides an interactive visualization environment that presents testing information through dashboards, charts, and analytical reports.

Execution trends, failure distributions, prediction outcomes, and testing performance metrics are displayed in an understandable format. These visual insights assist developers and testers in monitoring software quality, evaluating testing effectiveness, and identifying areas that require additional attention. Analytical reporting also supports informed decision-making and continuous improvement of testing strategies.

6) Continuous Integration and Continuous Testing

The proposed system supports seamless integration with Continuous Integration and Continuous Delivery environments.

Automated testing activities are triggered whenever software changes are introduced into the development repository. The framework automatically performs predictive analysis, prioritizes test cases, executes optimized test suites, and generates execution reports during each development cycle. This continuous testing approach accelerates feedback, improves software quality, and supports reliable software delivery processes.

B. System Architecture

The design of the proposed architecture is composed of several interconnected modules that collaborate to automate, optimize, and analyze software testing activities. The process starts with the Test Case Repository where all the automated test scripts written in selenium are stored. If a test is started, the Test Execution Engine fetches the test cases that are available and pulls out the appropriate historical test execution data from the Test History Database. Historical data is cleaned and prepared in the Data Processing Module for later analysis in machine learning. These attributes are extracted and transformed into a structured format suitable for the model training, including important attributes such as execution time, defect history, pass/fail ratio, and code change frequency.

The Dashboard and Reporting Module provides an overall dashboard of execution summaries, defect trends, prioritization effectiveness, pass/fail distributions and performance trends. Integration modules for CI/CD are included in the framework and allow automation of test runs during the software development lifecycle, for example, with Jenkins or GitHub Actions.

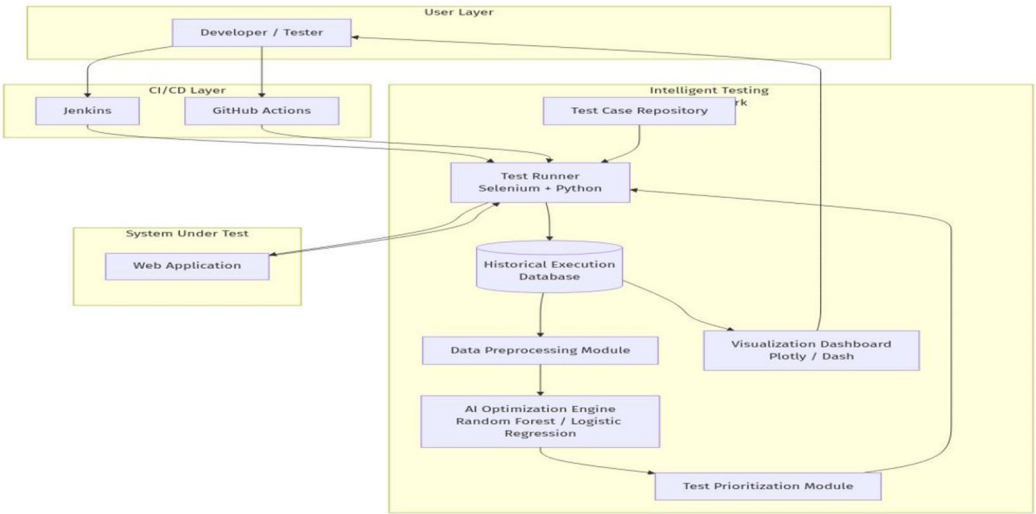


Fig 1 – System Architecture

- *User Layer:* The User Layer provides an interface for developers and testers to interact with the framework. It enables users to execute test cases, monitor testing activities, and analyze execution results.
- *CI/CD Layer:* The CI/CD Layer automates testing activities throughout the software development lifecycle. It supports continuous integration and continuous delivery by ensuring that testing is performed whenever software changes occur.
- *Jenkins:* Jenkins acts as an automation server that manages build processes and testing workflows. It supports continuous testing and helps identify defects early during development.
- *GitHub Actions:* GitHub Actions automates testing workflows directly within the source code repository. It ensures continuous validation of software changes and improves overall software quality.
- *Test Case Repository:* The Test Case Repository stores automated test scripts and testing configurations in a centralized location. It supports efficient test management and scalability of the testing framework.

C. *Working Flow of the Proposed System*

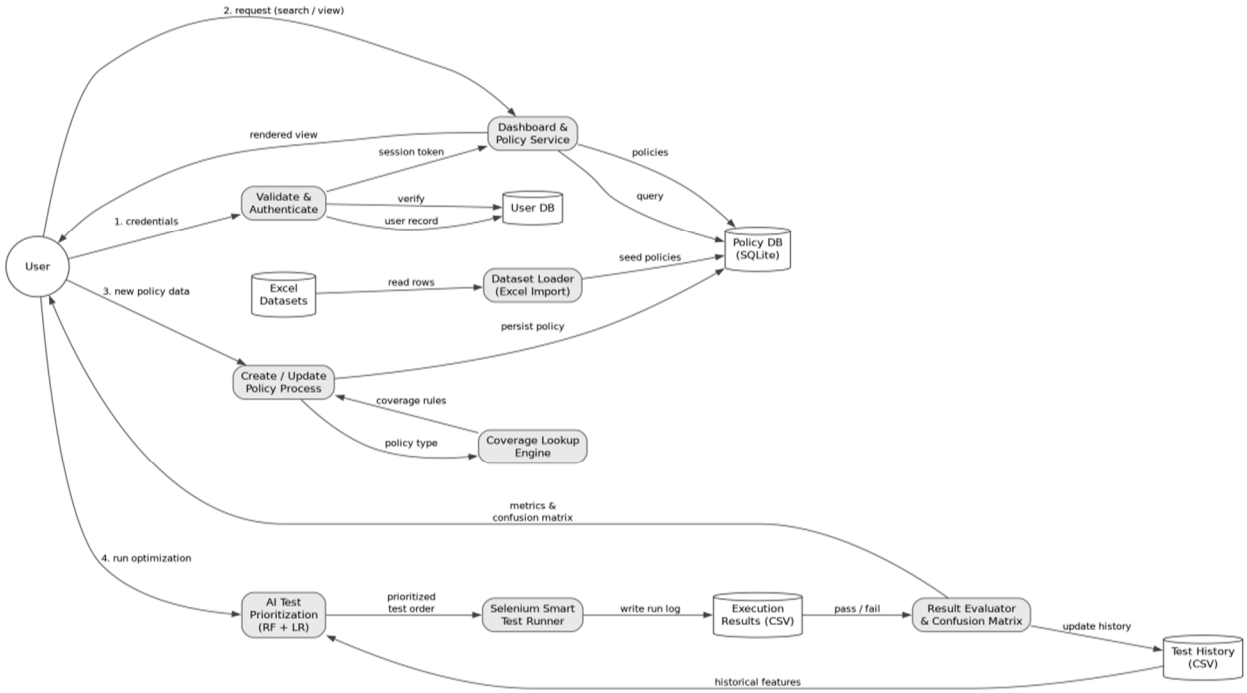


Fig 2 – Workflow of the proposed system

The diagram illustrates the flow of data within the proposed system. The user acts as the primary entity and interacts with the framework to access application functionalities, manage records, and execute testing activities. User information is maintained in the User Database, while application records and historical execution information are stored in dedicated databases.

Initially, the user is authenticated by the system before accessing application services. The framework allows data import, record management, and application monitoring through the dashboard. Historical execution information is collected and analyzed using machine learning models to identify testing patterns and predict potential failures.

Based on these predictions, the system prioritizes test cases and forwards them to the Selenium-based test execution engine. The generated results are evaluated, stored in the historical repository, and displayed through analytical dashboards. This continuous feedback mechanism enables intelligent test optimization, faster defect detection, and improved software quality assurance.

System Flow:

User → Authentication → Dashboard & Data Management → Historical Data Analysis → AI Test Prioritization → Automated Test Execution → Result Evaluation → Historical Data Storage → Dashboard Reporting

D. System Performance

Testing Activity	Test Cases	Average Execution Time (s)	Optimization Time (s)	Accuracy (%)
Login Testing	50	8.2	2.1	92.4
Policy Management Testing	75	12.5	3.4	93.1
Coverage Validation Testing	60	10.8	2.8	92.7
Dashboard Testing	40	6.9	1.9	91.8
Complete Regression Suite	225	38.4	8.7	93.5

The performance of the proposed intelligent testing framework was evaluated by analyzing automated test execution, optimization efficiency, and prediction accuracy. Experimental results demonstrate that the framework effectively prioritizes test cases based on historical execution information and predicted failure probabilities. Machine learning models were able to identify high-risk test cases and optimize the execution sequence without affecting overall test coverage.

For individual application modules such as login management, policy management, coverage validation, and dashboard functionalities, the framework achieved prediction accuracy above 90%. The optimization process required only a small amount of additional processing time while significantly improving the order of test execution. This enabled critical test cases to be executed earlier, resulting in faster identification of potential defects.

The complete regression test suite consisting of multiple automated test cases was executed through the Selenium-based testing engine. Historical execution records were analyzed using Random Forest and Logistic Regression models to generate prioritization scores. The optimized execution process reduced unnecessary testing effort and improved testing efficiency while maintaining reliable prediction performance.

Overall, the results indicate that the proposed framework successfully combines automated testing and machine learning techniques to improve regression testing effectiveness. The integration of AI-based prioritization, execution analytics, and continuous learning mechanisms contributes to faster feedback, better resource utilization, and improved software quality assurance.

IV. RESULTS

The proposed Intelligent Web Application Automated Testing Framework was successfully developed and evaluated using automated web testing, machine learning-based prediction, and test case optimization techniques. The experimental observations demonstrate that the framework can effectively execute automated test cases, analyze historical execution data, and prioritize high-risk test scenarios to improve regression testing efficiency. By integrating Selenium-based automation with predictive machine learning models, the system enables faster defect identification and more efficient utilization of testing resources.

The implementation also demonstrates the effectiveness of using historical execution information for intelligent decision-making. Machine learning algorithms such as Random Forest and Logistic Regression were utilized to analyze testing patterns and estimate failure probabilities, allowing the framework to dynamically optimize test execution order. Furthermore, the integration of analytical dashboards and CI/CD support provides continuous monitoring, real-time insights, and automated feedback, making the framework a scalable and reliable solution for modern software quality assurance environments.

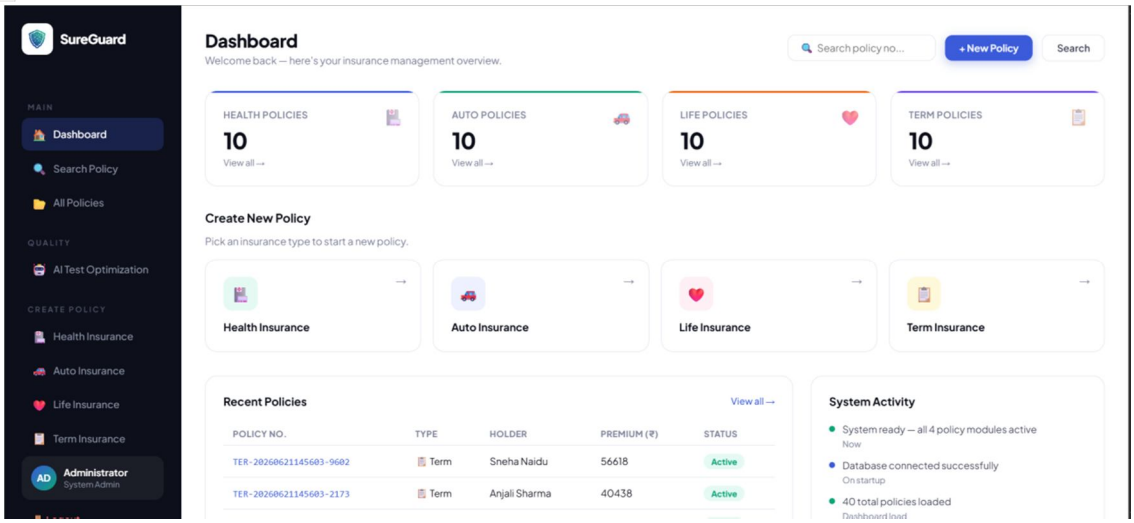


Fig 2 – Machine Learning-Based Test Prioritization and Risk Analysis Dashboard

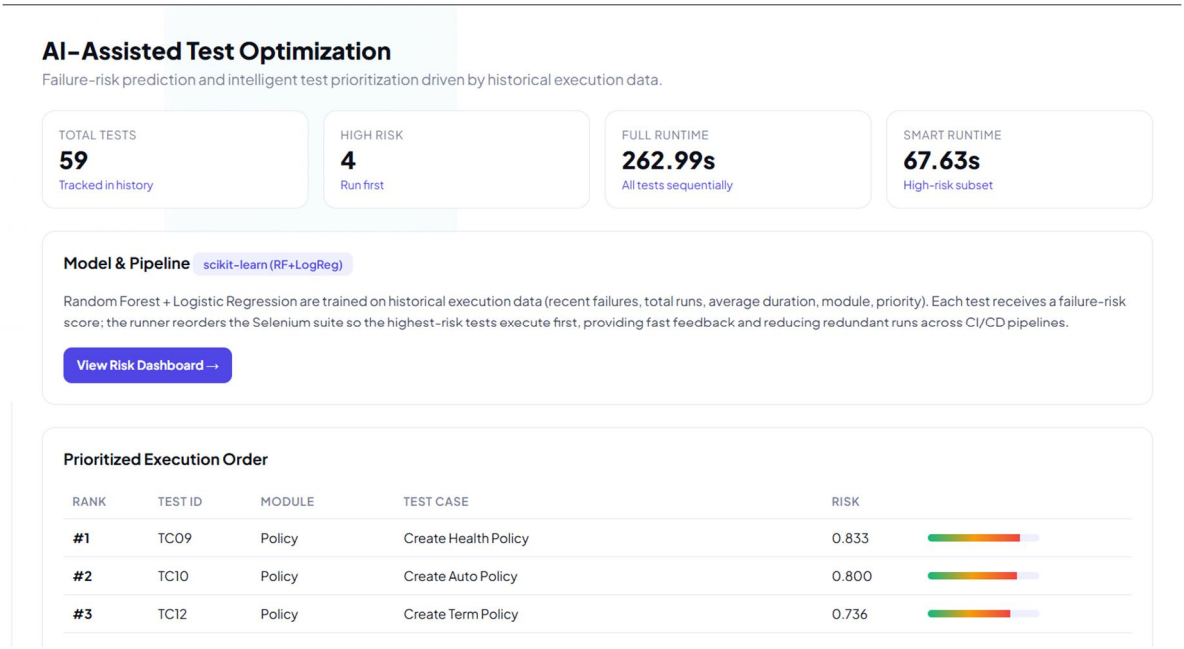


Fig 2 – Dashboard Interface for Policy Management and System Monitoring

V. CONCLUSION

The increasing complexity of modern software applications and the widespread adoption of Agile, Continuous Integration, and Continuous Delivery practices have created a growing demand for intelligent and efficient software testing solutions. Although traditional automation frameworks have significantly reduced manual testing effort, they often rely on fixed execution strategies that do not consider the likelihood of test failures or the criticality of individual test cases. This results in longer regression testing cycles, delayed feedback, and inefficient utilization of testing resources. To address these challenges, this project proposed and implemented an Intelligent Web Application Automated Testing Framework that combines automated testing with machine learning techniques to enhance the effectiveness and efficiency of software quality assurance activities.

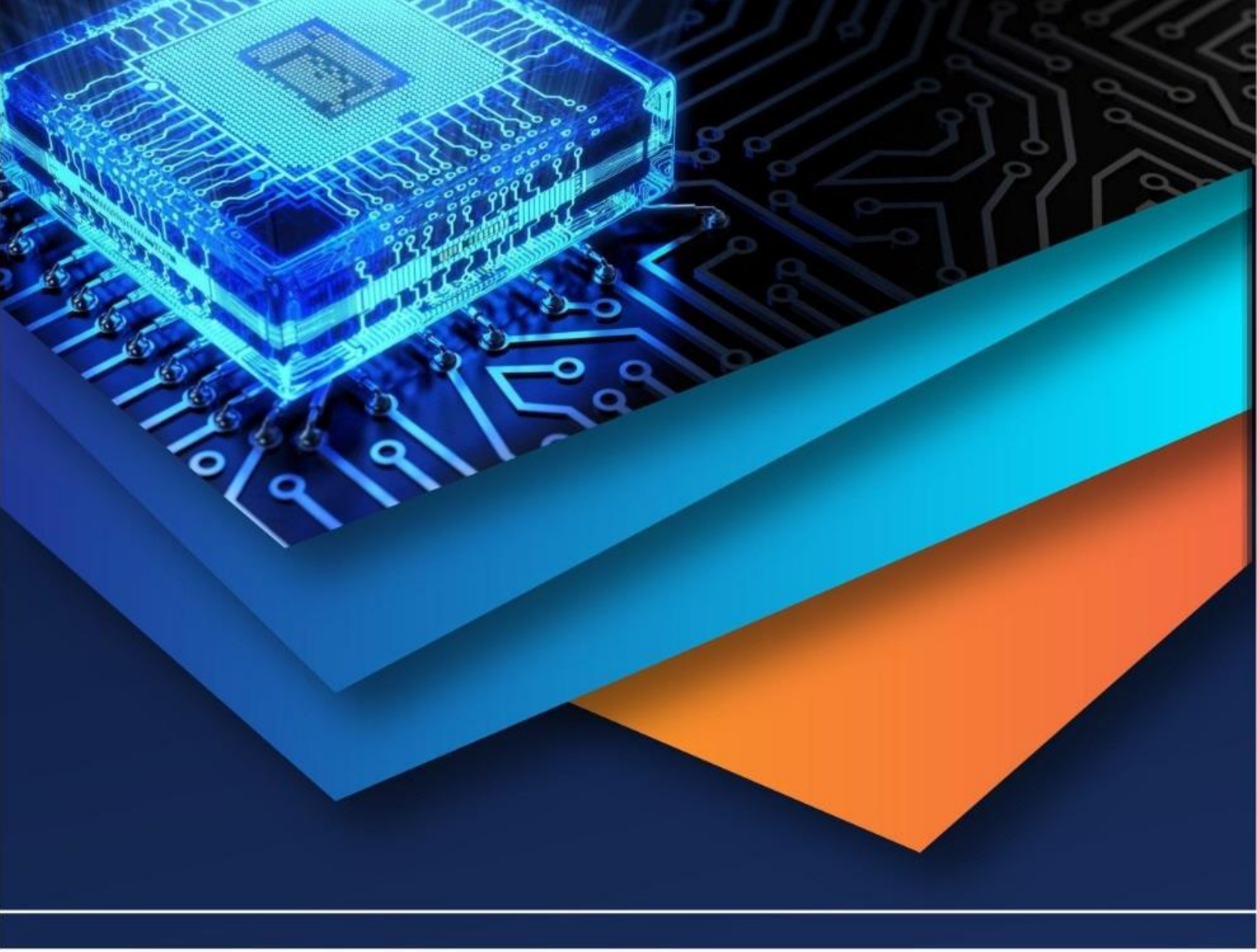
The developed framework integrates Selenium-based web automation, historical execution analysis, machine learning-driven prediction, intelligent test case prioritization, visualization, and CI/CD support within a unified environment. Historical testing data is continuously collected and analyzed using predictive models such as Random Forest and Logistic Regression to estimate the probability of future test failures. Based on these predictions, the framework dynamically prioritizes test cases and executes high-risk tests earlier in the testing cycle.

This approach improves fault detection efficiency, reduces redundant test execution, and accelerates the feedback process for developers and testers. The inclusion of analytical dashboards and automated reporting further enhances visibility into testing activities by providing meaningful insights into execution trends, testing performance, and optimization outcomes.

The results obtained from the implementation demonstrate that the proposed framework successfully transforms conventional automated testing into a more adaptive, intelligent, and data-driven process. By combining predictive analytics with automated test execution, the framework improves resource utilization, supports faster identification of software defects, and contributes to more reliable software releases. Furthermore, seamless integration with continuous development environments enables continuous testing and rapid quality validation throughout the software development lifecycle. Overall, the proposed framework establishes a strong foundation for the application of Artificial Intelligence in software testing and highlights the potential of intelligent automation to improve software quality, testing efficiency, and development productivity in modern software engineering environments.

REFERENCES

[1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, “Prioritizing Test Cases for Regression Testing,” IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929–948, 2001.
[2] M. J. Harrold, “Testing: A Roadmap,” Proceedings of the Conference on The Future of Software Engineering, ACM Press, pp. 61–72, 2000.
[3] T. Menzies, J. Greenwald, and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2–13, 2007.
[4] J. Campos, A. Riboira, A. Perez, and R. Abreu, “An Empirical Evaluation of Similarity Coefficients for Software Fault Localization,” IEEE Transactions on Software Engineering, vol. 39, no. 11, pp. 1487–1510, 2013.
[5] D. Marijan and M. Liaaen, “Machine Learning-Based Test Case Prioritization for Continuous Integration Testing,” Software Quality Journal, vol. 31, no. 2, pp. 487–515, 2023.
[6] Y. Zhao, D. Hao, and L. Zhang, “Revisiting Machine Learning-Based Test Case Prioritization for Continuous Integration,” Empirical Software Engineering, vol. 29, no. 1, pp. 1–28, 2024.
[7] A. Sharif, D. Marijan, and M. Liaaen, “DeepOrder: Deep Learning for Test Case Prioritization in Continuous Integration,” Proceedings of the International Conference on Software Testing, Verification and Validation, 2021.
[8] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, “Test Case Selection and Prioritization Using Machine Learning: A Systematic Literature Review,” ACM Computing Surveys, vol. 54, no. 5, pp. 1–38, 2021.
[9] S. Lachmann, S. Gotz, and C. Schaefer, “System-Level Test Case Prioritization Using Machine Learning Techniques,” IEEE International Conference on Machine Learning and Applications, pp. 1–8, 2016.
[10] SeleniumHQ, Selenium WebDriver Documentation, Selenium Project Documentation, Latest Edition.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY