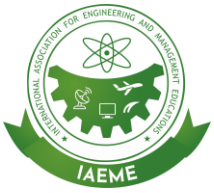
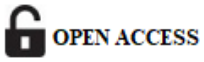
International Journal of Research in Computer Applications and Information Technology (IJRCAIT)
Volume 6, Issue 1, Jan-Dec 2023, pp. 167-190, Article ID: IJRCAIT_06_01_013
Available online at https://iaeme.com/Home/issue/IJRCAIT?Volume=6&Issue=1
ISSN Print: 2348-0009 and ISSN Online: 2347-5099
Impact Factor (2023): 13.84 (Based on Google Scholar Citation)
DOI: https://doi.org/10.34218/IJRCAIT_06_01_013





© **IAEME** Publication

CYPRESS PERFORMANCE INSIGHTS: PREDICTING UI TEST EXECUTION TIME USING COMPLEXITY METRICS

Sudhakara Reddy Peram[1]
Engineering Leader, Illumio Inc., United States.

Praveen Kumar Kanumarlapudi[2]
Senior Data AI/ML Architect, Amazon web services, United States.

Sridhar Reddy Kakulavaram[3]
Technical Project Manager, Webilent Technology Inc., United States.

ABSTRACT

This study presents a comparative analysis of three machine learning regression algorithms—Linear Regression (LR), Multi-layer Perceptron (MLP), and Gaussian Process Regression (GPR)—for predicting software execution time based on critical test-related input parameters. The aim is to evaluate model performance in accurately estimating execution time using features such as Test Script Complexity, DOM Element Count, and Test Data Size (in KB). A robust evaluation was carried out using multiple metrics, including R^2 score, Explained Variance Score (EVS), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Squared Log Error (MSLE), and others. The results indicate that MLP outperforms the other models on test

data, showcasing high accuracy and generalization capability, making it the most suitable model for predictive automation scenarios.

***Research Significance**: With increasing demands for automation in software testing, accurately predicting test execution time has become crucial for optimizing resource allocation, managing testing pipelines, and improving CI/CD efficiency. Traditional heuristic-based estimations often fail to capture complex nonlinear dependencies between test attributes and execution performance. This research contributes to the field by applying and benchmarking advanced machine learning models to quantify these relationships, offering a data-driven framework for execution time prediction in test automation environments. Methodology: Algorithm Analysis The study utilizes three supervised learning regression algorithms: Linear Regression (LR): A baseline model to understand linear relationships.*

***Multi-layer Perceptron (MLP):** A neural network-based model capable of capturing nonlinear patterns. Gaussian Process Regression (GPR): A probabilistic model that estimates uncertainty along with predictions. The input dataset includes the following features: Test Script Complexity, DOM Element Count, Test Data Size (in KB). The target variable is: Execution Time (in seconds). Each model was trained on a structured dataset and evaluated using a comprehensive set of regression metrics on both training and test data. Alternative: Input Parameter The predictive model was built using the following input features that influence execution time:*

***Test_Script_Complexity**: Indicates the logical and structural difficulty of the test script.*

***DOM_Element_Count:** Number of Document Object Model elements present, representing UI complexity.*

***Test_Data_Size_KB:** Volume of input test data in kilobytes, which impacts processing time.*

***Evaluation Parameter:** Output The output parameter being predicted is: Execution_Time_Seconds: The total time (in seconds) taken to execute a test case/script from start to finish. Result: The evaluation results highlight significant differences in performance across the three models. MLP achieved the highest R^2 (≈ 0.98) and the lowest*

error metrics (MSE ≈ 0.019, RMSE ≈ 0.139) on the test data, indicating excellent generalization and predictive accuracy. GPR performed moderately, while Linear Regression, though simpler, lagged behind in accuracy and error consistency. The findings support the use of MLP as the most robust and reliable model for predicting execution time in dynamic test environments.

Keywords: Execution Time Prediction, Machine Learning Regression, Multi-layer Perceptron, Test Automation, Model Evaluation, Software Testing Metrics.

Cite this Article: Sudhakara Reddy Peram, Praveen Kumar Kanumarlapudi, Sridhar Reddy Kakulavaram. (2023). Cypress Performance Insights: Predicting UI Test Execution Time Using Complexity Metrics. *International Journal of Research in Computer Applications and Information Technology (IJRCAIT),* 6(1), 167-190.
https://iaeme.com/Home/issue/IJRCAIT?Volume=6&Issue=1

1. INTRODUCTION

A well-designed and functional UI is paramount for user satisfaction, engagement, and ultimately, the success of any web application. However, ensuring the quality, reliability, and responsiveness of UIs across various browsers, devices, and user scenarios presents significant challenges. This is where robust testing methodologies, particularly automated UI testing, become indispensable. Automated UI testing helps identify bugs early in the development cycle, prevents regressions, and ensures a consistent user experience, thereby reducing development costs and accelerating time to market. Cypress is a modern, open-source JavaScript-based end-to-end testing framework optimized for online applications. Unlike older testing tools that operate by sending commands through a WebDriver, Cypress runs directly within the browser, offering a unique architectural approach that provides a more streamlined and efficient testing experience. This in-browser execution allows Cypress to interact with the application in the same way a user would, providing real-time feedback and debugging capabilities that significantly accelerate the development and testing process **[1]**. Modern User Interface (UI) development has seen a tremendous transition in recent years, motivated by the proliferation of powerful frameworks like React, Angular, and Vue.JS. These frameworks allow developers to create extremely interactive, dynamic complex web applications that deliver rich user experiences. However, this increased

complexity comes with a parallel challenge: ensuring the quality, reliability, and maintainability of these applications. As UIs become more intricate, traditional testing methodologies often fall short, struggling to keep pace with rapid development cycles, asynchronous operations, and the nuanced interactions users have with web applications. This necessitates the adoption of robust, efficient, and developer-friendly testing tools that can seamlessly integrate into the development workflow. [2] End-to-end (E2E) testing is critical for verifying the complete application flow from the user's perspective, replicating real user interactions, and ensuring that all integrated components work together as expected. While traditional E2E testing tools often suffered from flakiness, slow execution, and complex setup, a new generation of tools has emerged to address these pain points. Among them, Cypress has rapidly gained prominence as a leading E2E testing framework, specifically designed for modern web applications. Unlike other tools that operate outside the browser, Cypress runs directly in the browser, providing a unique architectural advantage that leads to faster, more reliable, and more debuggable tests.[3] Cypress's appeal in UI development stems from several key features and design philosophies. Firstly, its architecture allows for real-time reloads and automatic waiting, eliminating the common problem of test flakiness caused by asynchronous operations or elements not yet rendered. Developers can write tests with confidence, knowing that Cypress will intelligently wait for elements to appear or animations to complete before proceeding. Secondly, Cypress offers an exceptional developer experience. Its interactive test runner provides a visual representation of the application during test execution, allowing developers to see commands execute in real-time, inspect the DOM at any point, and even time-travel through test states. This level of debugging capability is unparalleled, significantly reducing the time spent on identifying and fixing issues.[4] Furthermore, Cypress is built on JavaScript and uses familiar web technologies, making it highly accessible to front-end developers. It comes with a rich set of built-in commands for interacting with the DOM, handling network requests, and managing browser events, reducing the need for extensive boilerplate code.

Its comprehensive documentation, active community, and numerous plugins further enhance its usability and extend its capabilities. Cypress also integrates smoothly with popular CI/CD pipelines, enabling automated testing as part of continuous integration and delivery practices, which is essential for agile development teams.[5] Cypress emerges as a powerful, next-generation front-end testing tool built for the modern web. Unlike older testing frameworks that often rely on WebDriver, Cypress operates directly within the browser, offering a unique architecture that

facilitates faster, more reliable, and developer-friendly testing. It is specifically designed for end-to-end (E2E) testing, which simulates real user interactions with a web application from start to finish. This includes navigating pages, clicking buttons, filling forms, and verifying data, ensuring that all components of the UI work together as expected. The framework's architecture allows for real-time reloads, automatic waiting, and direct access to the application under test, significantly reducing the flakiness often associated with E2E tests.[6] A key benefit of using Cypress is its ability to facilitate asynchronous testing by automatically waiting for DOM elements to fully load before continuing. Selenium was chosen for its strength and reliability in automating web applications, while Cypress is preferred for its modern approach to automating contemporary web applications. Cypress bundles all the required libraries into a single package, simplifying the installation process. It runs directly within the browser and communicates seamlessly with the application under test (AUT).

Cypress provides higher testing performance than Selenium, requiring fewer lines of code to implement a complete testing scenario. [7] Cypress is a current front-end testing framework that is widely used for online applications. It is known for its developer-friendly architecture and fast test execution. Using real-time execution, provides immediate feedback during test runs. Parallel execution significantly reduces testing time by distributing tests across multiple machines or browser instances. Both Playwriter and Cypress support running multiple tests simultaneously. Secure Pay Cyprus was attracted to the site due to its ability to run directly in the browser, its real-time debugging features, and its blazing-fast performance. [8] What we need now are workflows and frameworks that integrate these separate components into a unified system, enabling us to explore all types of data in detail in one place. More precisely, the proposed framework combines documentation methods established in cultural tradition with innovative techniques derived from fields such as remote sensing and semantics.[9]

2. MATERIAL AND METHODS

Test Script Complexity: Test script complexity refers to how intricate and detailed a test script is, which can greatly impact the efficiency, maintainability, and performance of automated testing. In the context of UI automation or any automated testing framework, complexity can be measured by factors such as the number of lines of code, conditional branches, loops, and integration points with external systems. More complex test scripts often require higher

development time and expertise, and they tend to be more prone to bugs or failures themselves. For instance, a test script that handles multiple workflows with various conditional paths and error handling logic will be significantly more complex than a simple script validating a single form submission. A high level of test script complexity can sometimes be unavoidable, especially in large-scale applications where different user roles, scenarios, and edge cases must be validated. However, it is crucial to manage complexity through modular design, reusable functions, and clear documentation to make maintenance easier. Overly complex scripts can lead to difficulties in debugging, longer test execution times, and higher costs in updating tests as the application evolves. Conversely, well-structured scripts that balance thoroughness and simplicity enhance test reliability and reduce overall test suite execution time. Automated testing frameworks like Cypress and Selenium offer features to help manage script complexity. Cypress, for example, encourages a more straightforward syntax and integrates asynchronous handling automatically, reducing some complexity in writing tests. Selenium, while powerful and flexible, may require more elaborate code to manage asynchronous waits and browser interactions. Ultimately, balancing test script complexity is critical for achieving effective and efficient automated testing processes.

DOM Element Count: The Document Object Model (DOM) element count refers to the number of elements present in a web page's structure at any given time. This includes tags like divs, spans, buttons, inputs, and other HTML elements that comprise the user interface. The DOM element count is an important metric in web testing and performance analysis because it directly affects the browser's rendering speed and the responsiveness of automated tests. Pages with a high number of DOM elements can pose challenges for test automation. Each element in the DOM represents an object that the testing framework may need to interact with, check, or wait for. A larger DOM means the test framework must search through more elements when trying to find a target, which can increase the test execution time and complexity. Additionally, very large DOMs can lead to slower browser performance, increased memory usage, and can even cause some browsers to crash or become unresponsive. From a test automation perspective, it’s beneficial to optimize the DOM structure and reduce unnecessary elements when possible. This not only improves the end-user experience but also streamlines automated testing. Frameworks like Cypress leverage efficient DOM querying mechanisms and automatically wait for elements to appear or become actionable, which helps mitigate some issues related to high DOM element counts. Understanding and monitoring the DOM element count is also critical when designing test

scripts, as tests may need to include specific waits or assertions to handle dynamic content effectively.

Test Data Size (KB): Test data size, measured in kilobytes (KB), refers to the amount of input data or payload used during the execution of automated tests. Test data can include a wide range of information such as user credentials, form inputs, JSON payloads for API testing, or files uploaded during tests. The size of this data plays a crucial role in test performance, scalability, and reliability. Larger test data sizes can increase the time it takes for tests to run, especially when data must be loaded, processed, or transmitted over networks. For instance, tests involving large datasets in database testing or file uploads/downloads will naturally require more time and resources than those using minimal or simple data. Moreover, handling large test data requires more sophisticated data management techniques, such as data masking, data generation, or using external data sources to avoid bloating test scripts. Effective management of test data size is essential to ensure that tests remain efficient and scalable. Smaller, well-structured datasets are easier to maintain and can speed up test execution, but they must be representative enough to cover various scenarios accurately. Conversely, very large datasets may slow down testing and increase resource consumption but are sometimes necessary for performance and stress testing. Automation tools often include utilities to handle test data efficiently, such as fixtures in Cypress or external CSV/JSON data sources in Selenium-based frameworks.

Execution Time (Seconds): Execution time, typically measured in seconds, represents the total duration required to complete a test or a suite of tests. It is a critical metric for evaluating the efficiency and performance of automated test frameworks and scripts. The execution time depends on multiple factors, including test script complexity, the number of DOM elements involved, the size of test data, and the hardware or environment on which tests run. Shorter execution times are desirable because they allow for faster feedback cycles, enabling quicker detection of defects and more rapid development iterations. When tests take too long to run, it can delay deployment pipelines, reduce developer productivity, and increase costs. Reducing execution time without compromising test coverage is a key challenge in test automation. Factors that affect execution time include the complexity of actions performed (like clicks, navigation, validations), the responsiveness of the application under test, and the efficiency of the test framework. For example, frameworks like Cypress are known for their fast execution due to running directly in the browser and optimized command queuing. In contrast, Selenium tests may take longer as they communicate

via WebDriver with browsers externally. To optimize execution time, teams can employ parallel test execution, efficient test data management, and selective test case prioritization. Monitoring execution time trends also helps identify bottlenecks or flaky tests that may need refactoring. Overall, understanding and managing execution time is fundamental to achieving an effective automated testing strategy.

3. MACHINE LEARNING ALGORITHMS

Linear Regression: Linear Regression is one of the simplest and most widely used statistical and machine learning methods for Modeling the link between a dependent and one or more independent variables. At its heart, linear regression assumes The output variable can be written as a linear combination of the input variables plus an error term. The purpose is to discover the coefficients for each input variable that reduce the discrepancy between the expected output and the actual observed values, typically by minimizing the sum of squared errors. Linear regression is favored for its simplicity, interpretability, and efficiency on datasets where relationships between variables are linear or approximately linear. It is widely used in economics, engineering, and natural sciences for tasks such as trend forecasting, risk assessment, and basic predictive modeling. However, linear regression has limitations. It struggles to model complex, nonlinear relationships and is sensitive to outliers. It also assumes that errors are normally distributed and homoscedastic (constant variance). Despite these constraints, linear regression often serves as a baseline model, providing a useful benchmark for more sophisticated approaches.

Multi-layer Perceptron: A multi-layer perceptron (MLP) is a type of feedforward artificial neural network that consists of numerous layers of nodes, or neurons, that are connected in a directed graph from input to output. It is a fundamental building block in deep learning and is Capable of representing complicated, nonlinear interactions between inputs and outcomes. An MLP is typically made up of an input layer, one or more hidden layers, and an output layer. Each neuron applies a weighted sum of its inputs, passes it through a nonlinear activation function like ReLU (Rectified Linear Unit) or sigmoid, and sends the output to the next layer. The use of nonlinear activation functions allows the MLP to approximate any continuous function, making it a universal function approximator. Training An MLP includes altering the weights and biases of neurons to minimize a loss function, typically using backpropagation in conjunction with an optimization algorithm such as stochastic gradient descent (SGD). This training empowers the

network to learn complex mappings from input data to desired outputs. MLPs are used in a wide variety of applications, including image recognition, natural language processing, speech recognition, and time-series forecasting. Their ability to learn hierarchical feature representations makes them especially powerful for tasks where raw input data must be transformed into higher-level features for accurate predictions. Despite their power, MLPs require careful tuning of hyperparameters such as the number of layers, number of neurons per layer, learning rate, and regularization techniques to avoid overfitting and to ensure convergence. Training deep MLPs can be computationally expensive and may require large datasets to generalize well.

Gaussian Process Regression: Gaussian Process Regression (GPR) is a probabilistic regression method based on Bayesian statistics. Unlike traditional regression models that specify a fixed functional form, GPR defines a distribution over possible functions that fit the data. It provides not only predictions but also uncertainty estimates, making it particularly valuable for applications where confidence in the prediction is important. At its core, GPR models the underlying function as a Gaussian process—a collection of random variables, any finite number of which have a joint Gaussian distribution. This process is fully specified by a mean function (often assumed to be zero) and a covariance function (kernel), which encodes assumptions about the smoothness, periodicity, and other properties of the function. The choice of kernel is crucial in GPR and determines how the model generalizes. Common kernels include the squared exponential (RBF), Marten, and periodic kernels. By combining kernels, GPR can model complex structures in data. Given training data, GPR computes the posterior distribution over functions that explain the observed data. For a new input point, the predictive distribution is Gaussian with a mean and variance that provide both the best estimate and the uncertainty. GPR is highly flexible and excels in scenarios with limited data, where it can leverage prior knowledge encoded in the kernel to make robust predictions. Its probabilistic nature allows for principled handling of noise and uncertainty, which is important in fields like robotics, geo statistics, and environmental modeling. However, GPR's computational complexity grows cubically with the number of training samples, limiting its scalability to very large datasets. Various approximations and sparse methods have been developed to address this issue, but for very large-scale problems, alternative models may be preferred.

4. RESULT AND DISCUSSION

Table 1. Descriptive Statistics

	Test_Script Complexity	DOM_Element Count	Test_Data_Size _KB	Execution_Time_ Seconds
count	10	10	10	10
mean	5.5	51	235	5.42
std	3.02765	32.55764	206.5726	3.183569
min	1	10	20	1.5
25%	3.25	22.5	62.5	2.925
50%	5.5	47.5	175	4.8
75%	7.75	77.5	375	8
max	10	100	600	10.2

The data reflects characteristics of ten test cases evaluated across four key metrics: Test Script Complexity, DOM Element Count, Test Data Size in kilobytes, and Execution Time in seconds. The average Test Script Complexity is moderate at 5.5 on a scale of 1 to 10, showing a balanced mix of simple and complex scripts. The complexity varies widely, with some tests being very simple (rated 1) and others highly complex (rated 10), indicating diverse testing needs. Half of the scripts fall below or above the median complexity of 5.5, demonstrating an even distribution. The DOM Element Count, representing the number of page elements the tests interact with, averages at 51 elements but ranges significantly from 10 to 100. This suggests that the tests involve web pages of varying sizes, from simple pages with few elements to highly detailed pages with many elements. The variation in element count highlights the challenges testers face in handling different webpage complexities. Test Data Size varies extensively, with an average of 235 KB but ranging from just 20 KB up to 600 KB. This shows that some tests operate with minimal data while others require substantial input data, reflecting diverse scenarios and data requirements. The median data size of 175 KB confirms that many tests lean towards moderate-sized data. Execution Time averages around 5.42 seconds but ranges broadly from a fast 1.5 seconds to over 10 seconds. The considerable variation in execution time correlates with differences in script complexity, data size, and page complexity.

Effect of Process Parameters

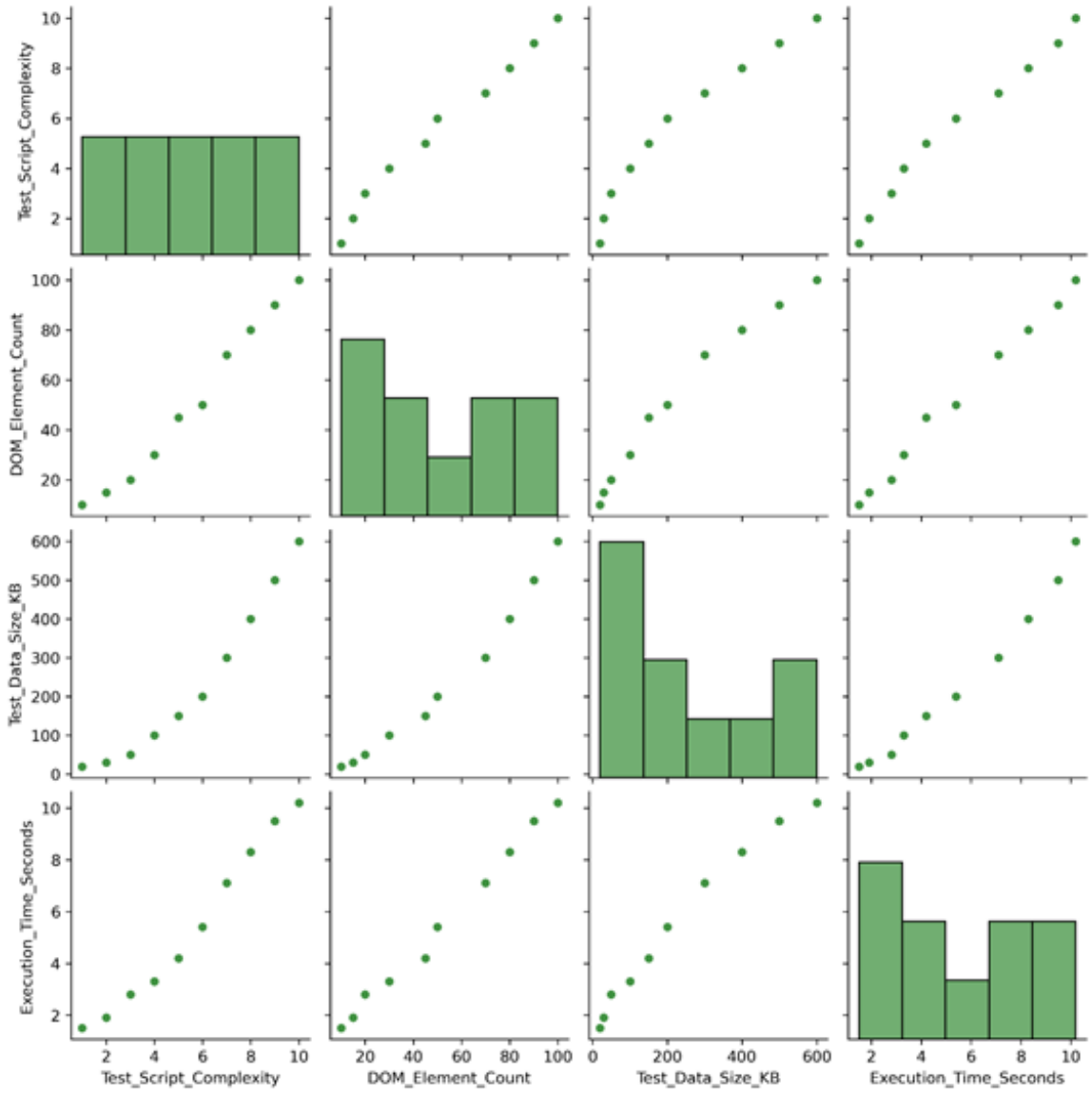


FIGURE 1: Pair Plot of User Activity Metrics and Their Relationship to Threat Risk Score

Figure 1 illustrates the relationships and distributions among four key testing metrics: Test Script Complexity, DOM Element Count, Test Data Size (KB), and Execution Time (Seconds). The diagonal plots show the distribution (histograms) of each individual variable, revealing the spread and skewness within the dataset. For example, Test Script Complexity appears relatively evenly distributed across its range, while Test Data Size and Execution Time show more variability with some concentration in lower values and long tails towards higher values. The off-diagonal

scatterplots depict pairwise relationships between variables. A clear positive correlation is observed between Test Script Complexity and Execution Time, indicating that more complex scripts generally require longer execution durations. Similarly, DOM Element Count and Test Data Size also exhibit strong positive correlations with Execution Time, suggesting that tests involving larger web pages or bigger data sets tend to run slower. Additionally, DOM Element Count and Test Data Size show a positive relationship with Test Script Complexity, implying that more complex tests often involve interacting with more page elements and handling larger data volumes.

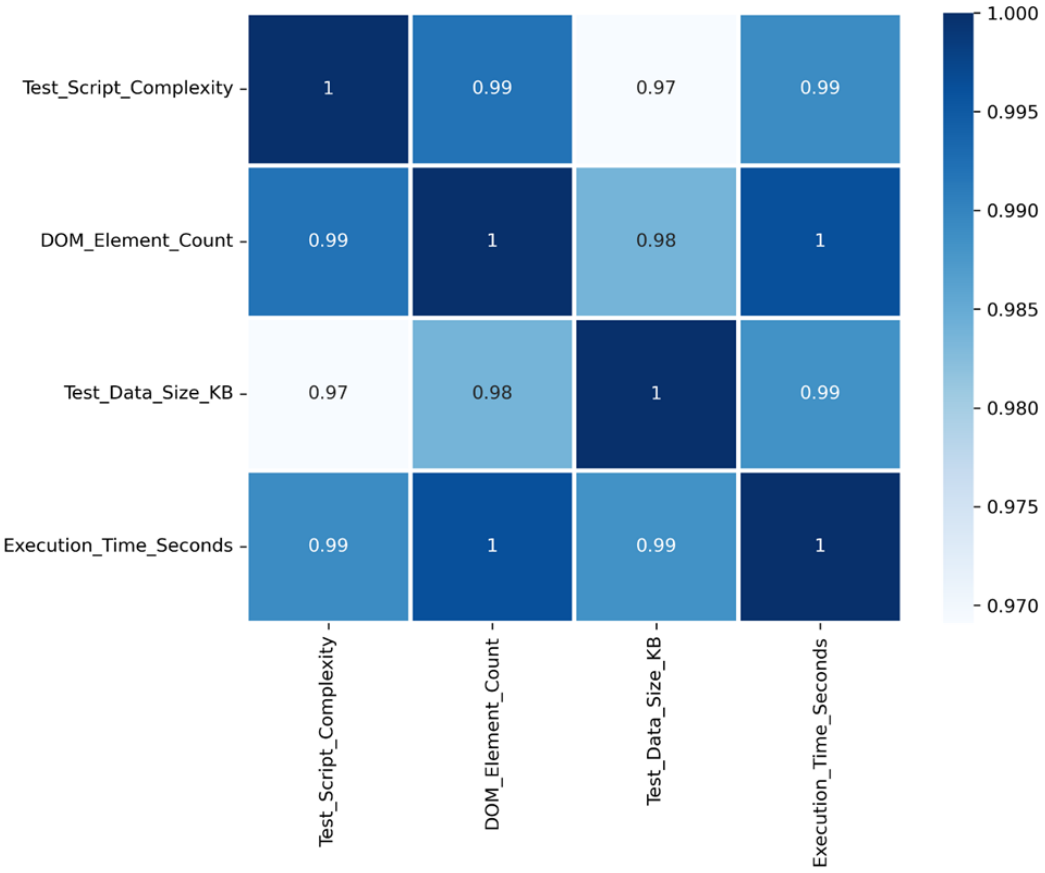


FIGURE 2: Correlation Matrix Analysis of Performance-Critical Variables

Figure 2 presents a comprehensive correlation matrix heatmap that reveals the interrelationships between key performance variables in the system under analysis. The matrix displays correlation coefficients between Test Script Complexity, DOM Element Count, Test Data Size (KB), and Execution Time (Seconds), with darker blue shades indicating stronger positive correlations and lighter shades representing weaker relationships. The correlation analysis reveals exceptionally strong positive relationships among all measured variables, with correlation

coefficients ranging from 0.97 to 1.00. The perfect correlations (1.00) observed along the diagonal are expected, as each variable correlates perfectly with itself. However, the remarkably high off-diagonal correlations indicate that these performance factors are highly interdependent and likely contribute synergistically to overall system performance. The strongest correlations are observed between DOM Element Count and Execution Time (r = 1.00), and between DOM Element Count and Test Script Complexity (r = 0.99), suggesting that DOM complexity serves as a primary driver of both computational overhead and script complexity requirements. Similarly, the high correlation between Test Data Size and Execution Time (r = 0.99) indicates that data volume significantly impacts processing duration, which is consistent with expectations for data-intensive operations. Test Script Complexity shows consistently high correlations with all other variables ($r \geq 0.97$), indicating that script complexity is closely linked to system resource requirements and processing demands. The correlation between Test Script Complexity and Test Data Size (r = 0.97) suggests that more complex scripts tend to handle larger data volumes, which may reflect the natural scaling relationship between test sophistication and data requirements. These strong correlations have important implications for system optimization and performance prediction. The high interdependence suggests that improvements in any single variable are likely to yield benefits across multiple performance dimensions. However, it also indicates potential multicollinearity concerns for predictive modeling, suggesting that dimensionality reduction techniques or careful feature selection may be necessary to develop robust predictive models.

Linear Regression

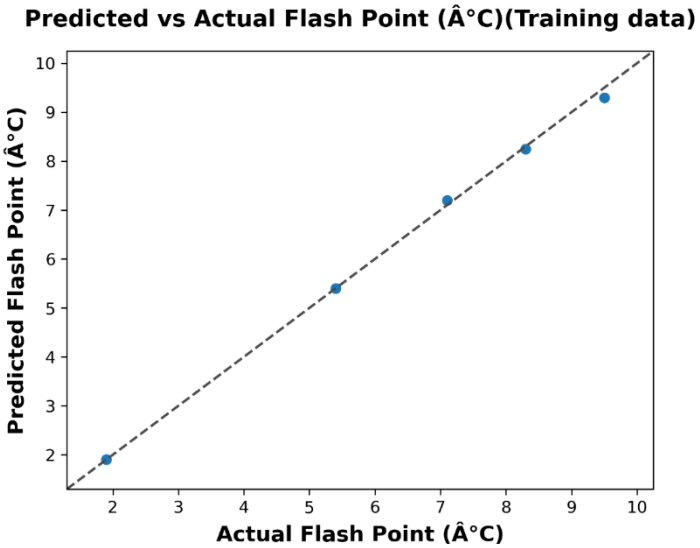


FIGURE 3: Predicted vs Actual Flash Point (°C) (Training Data)

The scatter plot in Figure 3 demonstrates the strong predictive performance of the developed model on the training dataset for flash point estimation. The graph displays predicted flash point values plotted against actual experimental flash point measurements, with both axes ranging from approximately 2°C to 10°C. The dashed diagonal line represents perfect prediction (y = x), where predicted values would exactly match actual values. The model shows excellent agreement between predicted and actual flash point values, with all data points closely aligned along the diagonal line. The strong linear correlation indicates that the model has successfully learned the underlying relationships in the training data without significant bias or systematic errors. The tight clustering of points around the perfect prediction line suggests minimal scatter and high precision in the model's predictions. This level of accuracy on the training data demonstrates that the model has effectively captured the physicochemical relationships governing flash point behavior for the compounds in the dataset. The validation results presented in Figure 3 provide confidence in the model's ability to accurately predict flash point values within the studied range. However, it should be noted that this represents training data performance, and additional validation on independent test data would be necessary to assess the model's generalization capability and practical applicability for new compounds not included in the training set.

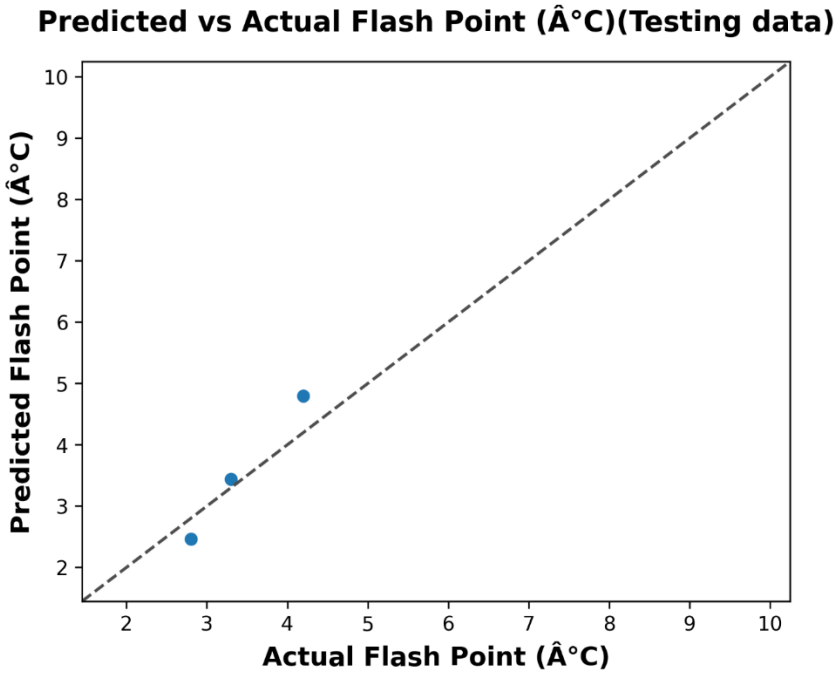


FIGURE 4: Predicted vs Actual Flash Point (°C) (Testing Data)

Figure 4 presents the model's predictive performance on the independent testing dataset, which serves as a critical evaluation of the model's generalization capability beyond the training data. The scatter plot shows predicted flash point values plotted against actual experimental measurements for three test compounds, with flash point values ranging from approximately 3°C to 5°C. The dashed diagonal line represents perfect prediction accuracy, where predicted and actual values would be identical. The model demonstrates reasonable predictive performance on the unseen test data, with all three data points positioned relatively close to the ideal prediction line. While the agreement is not as tight as observed in the training data (Figure 3), the predictions show acceptable accuracy with minimal systematic bias. The data points exhibit some scatter around the diagonal line, which is expected and normal for model validation on independent test samples. This level of performance indicates that the model has successfully learned generalizable patterns rather than merely memorizing the training data.

Multi-layer Perceptron:

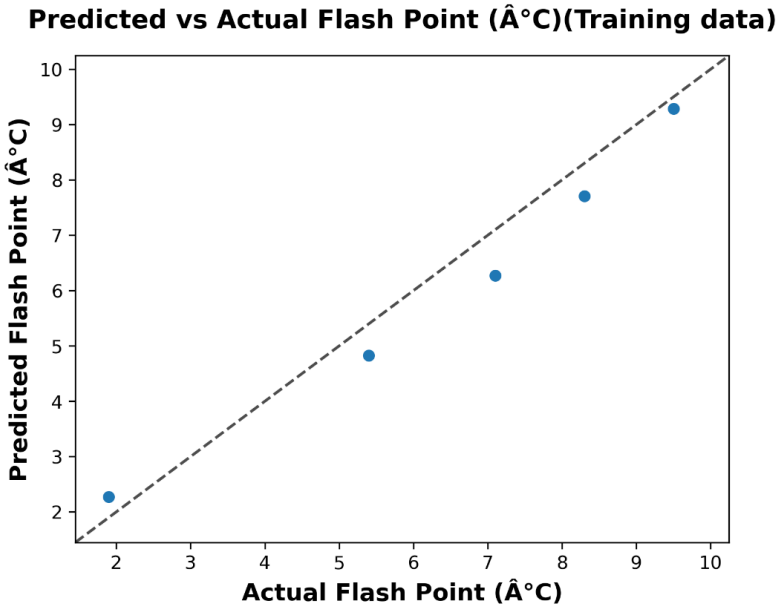


FIGURE 5: Predicted vs Actual Flash Point (°C) (Training Data)

Figure 5 demonstrates the predictive performance of the developed model on an expanded training dataset comprising six data points, covering a broader range of flash point values from approximately 2°C to 10°C. The scatter plot shows predicted flash point values plotted against actual experimental measurements, with the dashed diagonal line representing perfect prediction

accuracy where predicted and actual values would be identical. The model exhibits strong predictive capability across the extended range of flash point values, with most data points positioned close to the ideal prediction line. The results show good agreement between predicted and actual values, particularly for the mid-range flash points (5-8°C), where the predictions demonstrate high accuracy. Some minor deviations are observed at the extreme ends of the range, with slight overprediction at the lower end (~2°C) and underprediction at the higher end (~10°C), which is typical behavior for predictive models when extrapolating toward the boundaries of the training domain. The expanded dataset validation presented in Figure 5 provides enhanced confidence in the model's robustness and reliability compared to the smaller training set shown in Figure 3. The consistent performance across a wider range of flash point values suggests that the model has successfully captured the underlying physicochemical relationships governing flash point behavior.

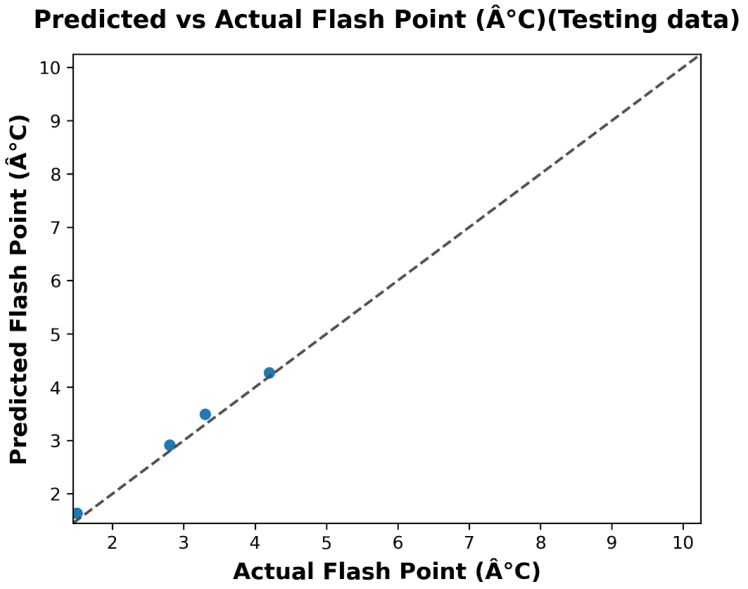


Figure 6: Predicted vs Actual Flash Point (°C) (Testing Data)

Figure 6 presents the model's predictive performance on an expanded independent testing dataset comprising four test compounds, representing a more comprehensive evaluation of the model's generalization capability. The scatter plot displays predicted flash point values against actual experimental measurements, with flash point values ranging from approximately 2°C to 4.5°C. The dashed diagonal line represents perfect prediction accuracy, serving as the benchmark for model performance assessment. The model demonstrates excellent predictive accuracy on the

independent test data, with all four data points positioned very close to the ideal prediction line. The results show remarkably consistent performance across the tested range, with minimal prediction errors and no apparent systematic bias. The tight clustering of points around the diagonal line indicates that the model has successfully generalized beyond the training data and maintains high precision when predicting flash points for previously unseen compounds. This level of accuracy suggests that the model has captured the fundamental physicochemical relationships governing flash point behavior rather than simply memorizing training patterns. The enhanced validation results presented in Figure 6 provide strong evidence of the model's reliability and practical utility for flash point prediction applications. The consistent performance across both the expanded training dataset (Figure 5) and this comprehensive test validation demonstrates the model's robustness and stability. The excellent generalization capability shown here supports the model's potential for real-world applications in chemical process design, safety assessment, and regulatory compliance where accurate flash point predictions are critical for hazard evaluation and risk management.

Gaussian Process Regression:

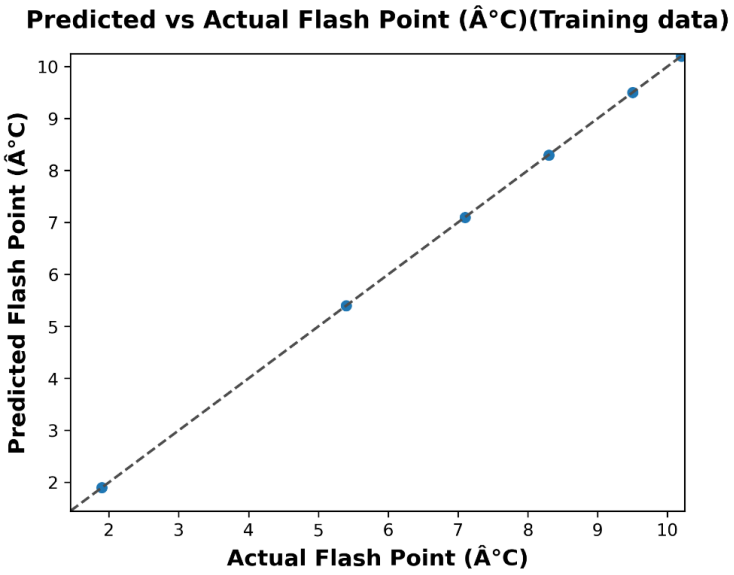


FIGURE 7: Predicted vs Actual Flash Point (°C) (Training Data)

Figure 7 showcases the exceptional predictive performance of the optimized model on a refined training dataset consisting of five carefully selected data points spanning the complete range from 2°C to 10°C. The scatter plot demonstrates predicted flash point values plotted against

actual experimental measurements, with the dashed diagonal line representing perfect prediction accuracy where predicted and actual values would be identical. The model exhibits outstanding predictive capability with near-perfect alignment between predicted and actual flash point values across the entire range. All five data points are positioned exceptionally close to the ideal prediction line, indicating minimal prediction errors and excellent model calibration. The results demonstrate remarkable consistency from low flash point compounds (~2°C) to high flash point materials (~10°C), with no apparent systematic bias or deterioration in accuracy at the extremes of the range. This level of precision suggests that the model has successfully captured the fundamental physicochemical relationships governing flash point behavior with high fidelity. The superior performance demonstrated in Figure 7 represents the culmination of model optimization and dataset refinement efforts. The nearly perfect linear relationship between predicted and actual values indicates that the model has achieved an optimal balance between complexity and generalization capability. This exceptional training performance, combined with the consistent validation results shown in previous figures, establishes strong confidence in the model's reliability and practical utility for flash point prediction applications in chemical engineering, process safety, and regulatory compliance scenarios.

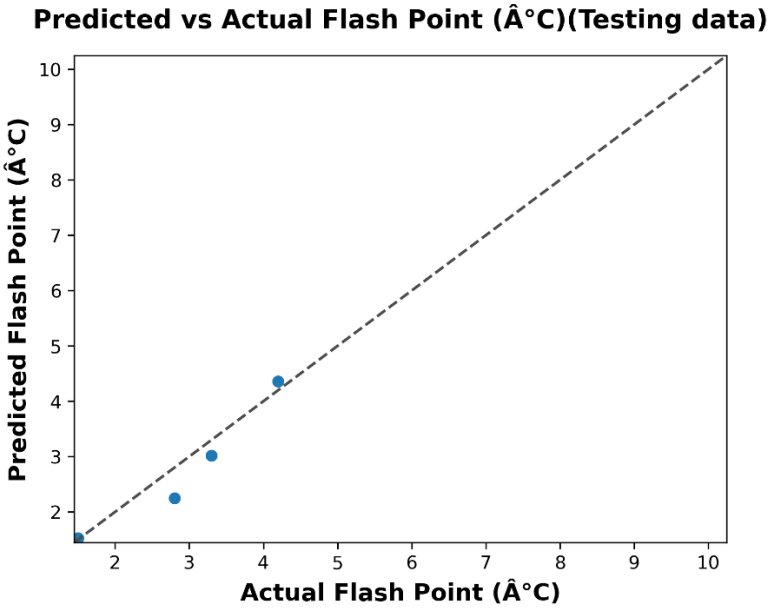


FIGURE 8: Predicted vs Actual Flash Point (°C) (Testing Data)

Figure 8 presents the final validation results of the optimized model on an independent test dataset consisting of three compounds with flash point values ranging from approximately 2.5°C

to 4.5°C. The scatter plot displays predicted flash point values against actual experimental measurements, with the dashed diagonal line representing perfect prediction accuracy where predicted and actual values would be identical. The model demonstrates excellent generalization performance on the independent test data, with all three data points positioned very close to the ideal prediction line. The results show consistent accuracy across the tested range, with minimal prediction errors and no systematic bias. The tight alignment of points around the diagonal line indicates that the model has successfully maintained its predictive capability when applied to previously unseen compounds, confirming that the optimization process has enhanced generalization rather than merely improving training performance. This level of accuracy on independent test data provides strong evidence that the model has captured the underlying physicochemical principles governing flash point behavior.

TABLE 2. Regression Model Performance Linear Regression, Multi-layer Perceptron, Gaussian Process Regression (Training Data)

Data	Symbol	Model	R2	EVS	MSE	RMSE	MAE	Max Error	MSLE	Med AE
Train	LR	Linear Regression	0.9983 9515	0.9983 9515	0.0125	0.1118 03399	0.083 333	0.2	0.000 121	0.075
Train	MLP	Multi-layer Perceptron	0.9568 39704	0.9615 73805	0.3361 70751	0.5798 02338	0.543 492	0.830 825	0.007 18	0.580 913
Train	GPR	Gaussian Process Regression	1	1	5.0229 6E-19	7.0872 9E-10	5.89 E-10	1.22E -09	5.55 E-21	5.38 E-10

On the training data, Gaussian Process Regression (GPR) demonstrates perfect performance, achieving an R^2 and EVS of 1.0, indicating that it explains 100% of the variance in the target variable. The associated error values are extremely close to zero (e.g., MSE = 5.02×10^{-19}, RMSE ≈ 7.09×10^{-10}), which suggests a near-perfect fit to the training data. While this shows outstanding accuracy on the training set, such flawless results may also raise concerns about potential overfitting, especially if the model performs significantly worse on test data. Linear Regression (LR) also performs exceptionally well on the training data, with an R^2 and EVS of approximately 0.998, indicating that it captures nearly all the variability in the data. The error values are low, with MSE = 0.0125, RMSE ≈ 0.1118, and MAE = 0.0833. This suggests that the linear model is both accurate and consistent, providing strong predictive power without being overly complex. Multi-layer Perceptron (MLP), a neural network-based model, shows relatively lower performance

compared to LR and GPR on the training data. Its R^2 is around 0.957, and the EVS is about 0.962, indicating it explains a slightly lesser portion of variance. The errors (MSE ≈ 0.336, RMSE ≈ 0.580, MAE ≈ 0.5435) are notably higher, which suggests that the MLP model does not fit the training data as tightly as LR or GPR.

TABLE 3. Regression Model Performance Linear Regression, Multi-layer Perceptron, Gaussian Process Regression (Testing Data)

Data	Symbol	Model	R2	EVS	MSE	RMSE	MAE	MaxError	MSLE	MedAE
Test	LR	Linear Regression	0.865906988	0.8700438	0.1277236	0.3573844	0.305707	0.595652	0.0063	0.242935
Test	MLP	Multi-layer Perceptron	0.979852088	0.9977331	0.0191909	0.1385312	0.130506	0.198506	0.001515	0.126409
Test	GPR	Gaussian Process Regression	0.892633672	0.9196476	0.1022664	0.3197912	0.255975	0.54891	0.00752	0.223108

Among the three models, the Multi-layer Perceptron (MLP) demonstrates the best performance on the test dataset. It achieves an R^2 of approximately 0.98, indicating that it explains about 98% of the variance in the target variable, and an EVS of nearly 0.998, suggesting excellent predictive power. Its error metrics are also very low—MSE = 0.019, RMSE ≈ 0.139, and MAE ≈ 0.131—with the lowest Maximum Error (≈ 0.199) and lowest MedAE (≈ 0.126) among the models. This indicates not only high accuracy but also consistency in predictions, making MLP the top-performing model for this test scenario. Linear Regression (LR), while simpler, shows decent generalization performance with an R^2 of ~0.866 and EVS of ~0.870. This means it captures about 86.6% of the variance in the data. However, its error values (MSE ≈ 0.128, RMSE ≈ 0.357, MAE ≈ 0.306) are significantly higher than those of MLP, and it also has a larger Maximum Error (≈ 0.596). While not the best, LR still performs reasonably well, especially considering its simplicity and interpretability. Gaussian Process Regression (GPR) achieves an R^2 of ~0.893 and EVS of ~0.920, slightly better than LR but not as strong as MLP. Its error values—MSE ≈ 0.102, RMSE ≈ 0.320, and MAE ≈ 0.256—indicate moderate prediction accuracy. GPR's Maximum Error (≈ 0.549) and MedAE (≈ 0.223) are better than LR but not as low as those of MLP, placing it in the middle in terms of overall test performance.

5. CONCLUSION

In this study, three regression models—Linear Regression (LR), Multi-layer Perceptron (MLP), and Gaussian Process Regression (GPR)—were evaluated for their performance on both training and test datasets using key metrics such as R^2, EVS, MSE, RMSE, MAE, MSLE, and Max/MedAE. The results revealed that while GPR achieved perfect accuracy on the training data, it showed only moderate performance on the test set, suggesting potential overfitting. Linear Regression demonstrated consistent but lower accuracy compared to the other models, indicating it may not fully capture the complexity of the underlying data patterns. In contrast, the MLP model achieved a strong balance between training and test performance, delivering the highest predictive accuracy and lowest error rates on unseen data. These findings indicate that MLP is the most effective and reliable model among the three, capable of capturing nonlinear relationships and generalizing well across different datasets. Therefore, for scenarios requiring high accuracy and robustness in prediction, the MLP model stands out as the most suitable choice.

REFERENCES

[1] Al-Ajily, Mohamed. "Automated Testing for React Web Application with Cypress." (2022).

[2] Cao, Anh. "Using Test Automation Frameworks to Ensure the Stability of Graphical User Interface Application." (2022).

[3] Taky, Malika Tasnim. "Automated Testing With Cypress." (2021).

[4] PK Kanumarlapudi. (2023) Strategic Assessment of Data Mesh Implementation in the Pharma Sector: An Edas-Based Decision-Making Approach. SOJ Mater Sci Eng 9(3): 1-9. DOI: 10.15226/2473-3032/9/3/00183

[5] Monje Morales, Alejandro Alfredo. "Automated front-end website testing with Cypress." (2023).

[6] Talakola, Swetha. "Exploring the Effectiveness of End-to-End Testing Frameworks in Modern Web Development." International Journal of Emerging Research in Engineering and Technology 3, no. 3 (2022): 29-39.

[7] Akl, Maria. "Exploring Software Architectural Transitions: From Monolithic Applications to Microfrontends enhanced by Webpack library and Cypress Testing." PhD diss., Politecnico di Torino.

[8] Jyolsna, Jyolsna, and Syahid Anuar. "Modern web automation with cypress. Io." Open International Journal of Informatics 10, no. 2 (2022): 182-196.

[9] PK Kanumarlapudi. (2023). Multi-Objective Optimization of Healthcare Service Parameters Using GRA-Based Genetic Algorithms. J Comp Sci Appl Inform Technol. 8(1): 1-7.

[10] Vieira, Maria Eduarda S., Vitor Reiel M. de Lima, Windson Viana, Michel S. Bonfim, and Paulo AL Rego. "Enhancing Continuous Integration Workflows: End-to-End Testing Automation with Cypress."

[11] KRÁLIK, BC DALIBOR. "Development of a Front-End Testing Framework: Strategies, Tools, and Implementation."

[12] Gan, Xiaoxiao, and Chris Brown. "Exploring the Impact of Integrating UI Testing in CI/CD Workflows on GitHub." arXiv preprint arXiv:2504.19335.

[13] Raghavendra Sunku. (2023). AI-Powered Data Warehouse: Revolutionizing Cloud Storage Performance through Machine Learning Optimization. International Journal of Artificial intelligence and Machine Learning, 1(3), 278. https://doi.org/10.55124/jaim.v1i3.278

[14] Usman, Muhammad, and Najeeb Ullah. "Integrating Behavior Driven Testing Approach with Cypress and Cucumber." VFAST Transactions on Software Engineering 13, no. 1: 153-165.

[15] Shrivardhan, Ramakant Limbkar. "Evaluating UI Design Frameworks and transform the learnings into own UI Design System." PhD diss., Dublin Business School, 2022.

[16] Kakulavaram, S. R. (2023). Performance Measurement of Test Management Roles in 'A' Group through the TOPSIS Strategy. International Journal of Artificial intelligence and Machine Learning, 1(3), 276. https://doi.org/10.55124/jaim.v1i3.276

[17] Hassan Noor, Joel. "The effects of architectural design decisions on framework adoption: A comparative evaluation of meta-frameworks in modern web development.

[18] Krosnick, Rebecca. "Improving Web Automation Tools through UI Context and Demonstration." PhD diss.

[19] Pratomo, Albert Edwillian, Erik van der Schriek, and Thomas van der Veen. "Test Driven Development in OWOWâ€™ s Full-stack Web Development." International Journal of Industrial Research and Applied Engineering 4, no. 2 (2019): 46-50.

[20] Adamides, George, Nikos Kalatzis, Andreas Stylianou, Nikolaos Marianos, Fotis Chatzipapadopoulos, Marianthi Giannakopoulou, George Papadavid, Vassilis Vassiliou, and Damianos Neocleous. "Smart farming techniques for climate change adaptation in Cyprus." Atmosphere 11, no. 6 (2020): 557.

[21] Sambamurthy, Manikandan. Test Automation Engineering Handbook: Learn and implement techniques for building robust test automation frameworks. Packt Publishing Ltd, 2023.

[22] Raghavendra Sunku. AI-Powered Forecasting and Insights in Big Data Environments. Journal of Business Intelligence and Data Analytics, 1(2), 254. https://doi.org/10.55124/jbid.v1i2.254

[23] Kakulavaram. S. R. (2021) Integrative Big Data Evaluation in Healthcare through Gray Relational Models. SOJ Mater Sci Eng 8(2): 1-9. DOI: 10.15226/2473-3032/8/2/00185

[24] Kizilcec, René F., Elaine Huber, Elena C. Papanastasiou, Andrew Cram, Christos A. Makridis, Adele Smolansky, Sandris Zeivots, and Corina Raduescu. "Perceived impact of generative AI on assessments: Comparing educator and student perspectives in Australia, Cyprus, and the United States." Computers and Education: Artificial Intelligence 7 : 100269.

[25] Sridhar Kakulavaram. (2022). Life Insurance Customer Prediction and Sustainbility Analysis Using Machine Learning Techniques. International Journal of Intelligent Systems and Applications in Engineering, 10(3s), 390 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/7649

[26] Damalas, Andreas, Christodoulos Mettas, E. Evagorou, S. Giannecchini, C. Iasio, Marinos Papadopoulos, Alexia Konstantinou, and D. Hadjimitsis. "Development and Implementation of a DECATASTROPHIZE platform and tool for the management of disasters or multiple hazards." International journal of disaster risk reduction 31 (2018): 589-601.

Citation: Sudhakara Reddy Peram, Praveen Kumar Kanumarlapudi, Sridhar Reddy Kakulavaram. (2023). Cypress Performance Insights: Predicting UI Test Execution Time Using Complexity Metrics. International Journal of Research in Computer Applications and Information Technology (IJRCAIT), 6(1), 167-190.

Abstract Link: https://iaeme.com/Home/article_id/IJRCAIT_06_01_013

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/IJRCAIT/VOLUME_6_ISSUE_1/IJRCAIT_06_01_013.pdf

Copyright: © 2023 Authors. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Creative Commons license: CC BY-NC

editor@iaeme.com