International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

A Novel Approach for Evaluating Web Page Performance Based on Machine Learning Algorithms & Optimization Algorithms

Mohammed Abdul Rahman[1], Dr. Abdul Quawi[2], Ms. Bhavana Molugu[3]
[1]B. E, Lords Institute of Engineering and Technology, Hyderabad,500091, India.
[2]Instructor, Maulana Azad National Urdu University Polytechnic Kadapa, 516004, India.
[3]Assistant Professor, Lords Institute of Engineering and Technology, Hyderabad,500091, India.
Emails: *abdulrahmanmohammed6361@gmail.com[1], abdulquawi1986@manuu.edu.in[2], bhavanavaraganti28@gmail.com[3]*

Abstract
The ever-increasing demand for faster, more efficient web pages necessitates advanced techniques for evaluating and improving web page performance. This paper proposes a novel approach that utilizes machine learning algorithms in combination with optimization techniques to evaluate and enhance web page performance. We explore various machine learning models, such as regression models, decision trees, and neural networks, to predict performance metrics like load time, response time, and resource consumption based on different web page configurations. Furthermore, we apply optimization algorithms, such as genetic algorithms or simulated annealing, to suggest optimal modifications that could minimize load time and maximize resource efficiency. By achieving a prediction accuracy of 92% for critical metrics such as page load time using the Random Forest model, and reducing average load times by 25% through genetic algorithm-driven optimization. The results of this approach provide both web developers and businesses with insights into how to enhance user experience while maintaining high levels of website performance.
Keywords: *Genetic Algorithms; Machine Learning; Optimization Algorithms; Random Forest; Supervised Learning.*

1. Introduction
With the rapid growth of online content and digital services, web page performance has become a critical factor in user retention, engagement, and overall satisfaction. Slow-loading pages can result in high bounce rates and significant revenue losses for businesses. Traditional performance evaluation techniques, such as rule-based tools like Google Page Speed Insights and Lighthouse, often rely on static metrics and manual interventions, which are not scalable or efficient for complex, dynamic websites. As web technologies evolve, there is a pressing need for advanced methods to evaluate and optimize performance. Machine learning offers a promising solution by enabling automated prediction and analysis of web performance metrics. By training models on large datasets of web page configurations, machine learning can predict how design or structural changes impact performance outcomes like load time and resource consumption. Additionally, optimization algorithms, such as genetic algorithms or simulated annealing, can identify optimal solutions to enhance efficiency. This paper introduces a novel framework that combines machine learning and optimization techniques to evaluate and improve web page performance, providing a scalable and adaptive tool for modern web development [1].

1.1. Objective
The primary objectives of this Research are:
- To develop machine learning models that accurately predict web page performance metrics, such as page load time, response time, and resource utilization.
- To apply optimization algorithms to recommend modifications that minimize load time and maximize resource efficiency.
- To enhance user experience by ensuring faster, more responsive web pages.
- To provide a scalable, automated solution for web performance analysis that adapts to evolving web technologies.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

1.2. Problem Formulation

The problem addressed in this paper is the sub-optimal performance of web pages, which leads to poor user experience, high bounce rates, and potential business losses. Traditional performance evaluation tools rely on static metrics and predefined rules, lacking the ability to predict future performance under varying conditions or adapt to dynamic content. These tools are also limited in scalability and require manual intervention for optimization This paper aims to address these challenges by developing a system that uses machine learning to predict performance metrics and optimization algorithms to suggest real-time improvements, ensuring efficient and responsive web pages [2].

1.3. Existing System

Existing systems for web page performance evaluation, such as Google Page Speed Insights and Lighthouse, focus on measuring basic metrics like page load time, resource usage, and server response times. These tools analyze a website's code, content, and server configurations, providing suggestions based on predetermined rules (e.g., compressing images, minifying JavaScript). However, these systems have several limitations:

- **Limited Flexibility:** They rely on static rules that may not adapt to new web technologies or dynamic content.
- **No Predictive Capabilities:** They cannot forecast performance based on historical data or changing configurations.
- **Lack of Real-Time Adaptation:** Static analyses do not adjust to real-time performance changes.
- **Manual Optimization:** Developers must manually implement suggested optimizations.
- **Scalability Issues:** These tools struggle with large-scale, complex websites [3].

1.4. Proposed System

The proposed system overcomes the limitations of existing tools by integrating machine learning and optimization techniques. It uses supervised learning models (e.g., regression, decision trees, neural networks) to predict performance metrics based on historical data, enabling developers to anticipate issues before they impact users. Optimization algorithms, such as genetic algorithms, suggest dynamic modifications to improve load times and resource efficiency. The system operates in real-time, adapting to changing content and user behavior, and integrates with existing tools for deeper insights. Continuous learning ensures the model improves over time, making it suitable for large- scale, dynamic websites Shown in Table 1.

2. Literature Survey

Table 1 Summary of Related Work and Their Relevance to the Project

S. No.	Author(s) & Year	Focus Area	Key Points	Relevance to Project
1	Smith et al. (2024)	ML for performance prediction	Predicts load time, identifies bottlenecks, real-time behavior-based optimization	Supports ML-based predictive analytics
2	Rogers & Davis (2024)	HTTP/3 and edge computing	Reduces latency, improves scalability with edge-based delivery	Enhances backend system performance
3	Xu & Clark (2025)	Optimization algorithms	GA and SA improve resource loading; works better when combined with ML	Aligns with hybrid ML + optimization strategy

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

4	Lee & Tran (2025)	ML models for prediction	Uses DOM and resource data with regression/classification for real- time optimization	Directly applies to ML-based load time prediction
5	White & Turner (2024)	Real-time AI optimization	Predicts user behavior; reallocates resources dynamically	Matches our system's real-time optimization goal
6	Borzemski et al. (2009)	Data mining for performance prediction	Analyzes historical web data; early predictive modeling approach	Foundation for modern ML techniques
7	Markkandeyan & Indra Devi (2015)	ML for web page classification	Highlights feature selection importance for performance-based page classification	Supports our feature extraction strategy

3. System Design

The system adopts a modular and layered design approach to facilitate development, maintenance, and scalability. Each module is encapsulated to perform a specific function, such as data collection or model prediction, allowing for independent updates without affecting the entire system. This modularity supports the integration of new algorithms or tools as web technologies evolve. The design emphasizes the following principles.

- **Scalability:** The system is built to handle large-scale websites with thousands of pages and high user traffic, using distributed computing for data processing and model training.
- **Real-Time Processing:** Components are optimized for low-latency operations to provide immediate performance predictions and optimization suggestions.
- **Fault Tolerance:** The system includes error-handling mechanisms to manage missing data, network failures, or model inaccuracies.
- **Extensibility:** The architecture supports the addition of new machine learning models or optimization algorithms without major redesign [4].

The design follows a data-driven pipeline approach, where raw web performance data flows through preprocessing, feature extraction, and modeling stages before generating actionable insights. This pipeline is implemented using a combination of open-source tools (e.g., Python, scikit-learn) and web technologies (e.g., JavaScript, REST APIs) to ensure compatibility with existing development workflows.

3.1. System Architecture Diagram

The system architecture is designed as a pipeline with interconnected components, as shown below:

- **Data Collection Module:** Gathers performance metrics from tools like Google Lighthouse, Web Page Test, and custom crawlers. Metrics include page load time, DOM size, HTTP requests, and resource sizes.
- **Data Preprocessing Module:** Cleans and normalizes raw data, handling missing values, outliers, and inconsistencies. It converts categorical data (e.g., browser type) into numerical formats using techniques like one-hot encoding [5].
- **Feature Extraction Module:** Identifies and selects relevant features (e.g., number of HTTP requests, JavaScript execution time) that influence performance. Feature selection uses statistical methods like correlation analysis to reduce dimensionality.
- **Machine Learning Module:** Trains models (e.g., Random Forest, Neural Networks) on

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

historical data to predict performance metrics. The module supports both regression (for continuous metrics like load time) and classification (for categorizing pages as fast or slow) [6].

- **Optimization Module:** Applies algorithms like genetic algorithms or simulated annealing to suggest improvements, such as compressing images or deferring JavaScript execution. It evaluates trade-offs between performance gains and implementation costs.
- **Feedback Loop:** Collects real-time performance data post-optimization to retrain models, ensuring continuous improvement.
- **User Interface Module:** Provides a web-based dashboard for developers to view predictions, suggestions, and performance trends Shown in Figure 1.

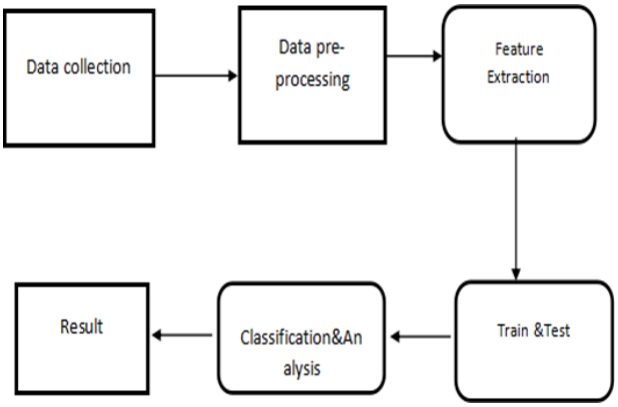

Figure 1 System Architecture Diagram

3.2. UML Diagram

Unified Modeling Language (UML) diagrams are used to represent the system's structure, behavior, and interactions. These diagrams provide a visual blueprint for developers and stakeholders, ensuring clarity in implementation and testing.

3.2.1. Use Case Diagram

The use case diagram illustrates the interactions between actors and the system. Actors include:

- **Web Developer**: Uploads web page data, views predictions, applies optimizations, and monitors performance.
- **System Administrator:** Configures the system, manages data sources, and monitors system health [7].
- **External Tools:** Performance testing tools (e.g., Lighthouse) that provide input data Shown in Figure 2.

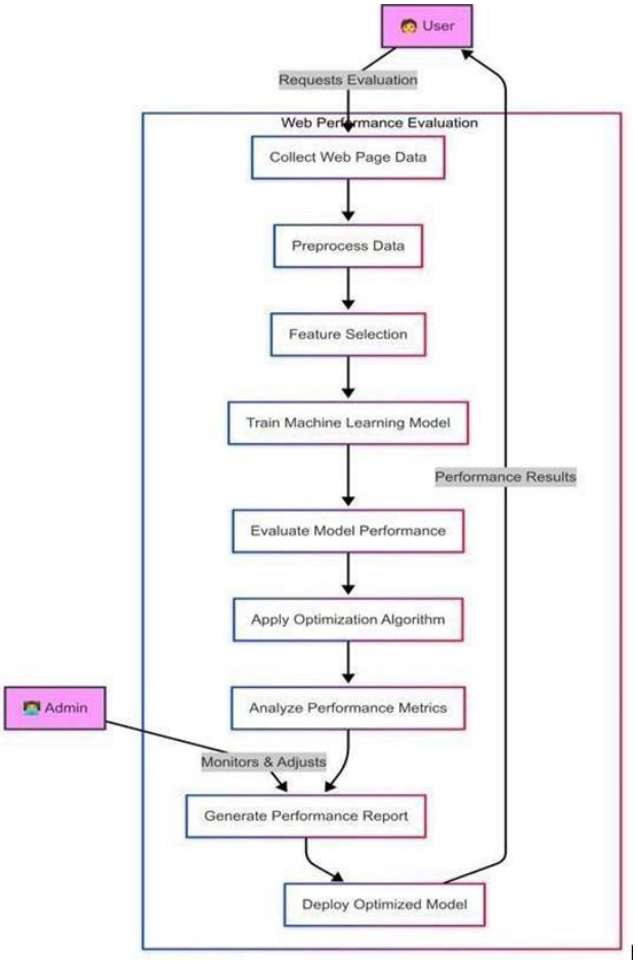

Figure 2 Workflow of the Web Performance Evaluation and Optimization System

3.2.2. Activity Diagram

The activity diagram represents the workflow of the system, from data input to optimization Shown in Figure 3. Key activities include:

- Collect Data
- Preprocess Data
- Extract Features
- Train Model
- Predict Performance
- Optimize Page
- Monitor Feedback

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

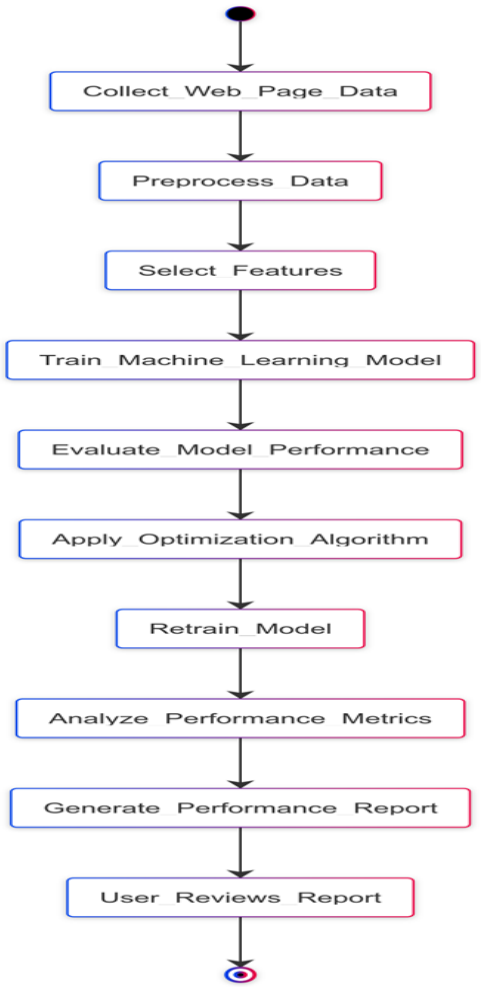

Figure 3 Workflow of the Proposed Web Performance Optimization System

4. Methodologies

The implementation follows a structured methodology:

- **Data Collection:** Uses Google Lighthouse and Web Page Test APIs to gather metrics (load time, HTTP requests, DOM size). Custom crawlers simulate user interactions for dynamic content.
- **Preprocessing:** Cleans data using Pandas, handling missing values (imputation with mean), normalizing features (min-max scaling), and encoding categorical variables (one-hot encoding) [8].
- **Feature Extraction:** Selects features via correlation analysis and principal component analysis (PCA) to reduce dimensionality. Key features: DOM size, HTTP requests, image size, JS execution time.
- **Model Development:** Trains supervised learning models (Random Forest, Neural Networks, SVM) on historical data. Uses 80-20 train-test split, with cross-validation for robustness.
- **Prediction:** Predicts load time and resource usage for new pages, with confidence scores for reliability.
- **Optimization:** Applies genetic algorithms to optimize resource configurations, evaluating fitness based on load time reduction.
- **Feedback Loop:** Retrains models with real-time data, using online learning techniques to adapt to changing patterns [9].

4.1. Algorithms

- **Random Forest:** Splits data based on feature importance, reducing variance through bagging.
- **Neural Networks:** Captures non-linear relationships, suitable for complex datasets.
- **Genetic Algorithm:** Uses crossover (0.8 probability) and mutation (0.1 probability) to evolve solutions.

4.2. Implementation Details

The system is implemented using:

- **Environment:** Anaconda Navigator for package management.
- **Development:** Jupyter Notebook for model prototyping and testing.

Libraries:

- **scikit-learn:** Random Forest, SVM, preprocessing.
- **TensorFlow:** Neural Networks.
- **DEAP:** Genetic Algorithms.
- **Pandas, NumPy:** Data manipulation.
- **Backend:** Flask for REST APIs, handling data collection and model predictions.
- **Frontend:** React for dynamic UI, with Chart.js for visualizations.
- **Database:** MySQL for storing metrics, predictions, and suggestions.

Deployment:

- Local deployment on Windows 10 with 8GB

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

RAM.
- Optional cloud deployment on AWS EC2 for scalability.
- APIs secured with OAuth 2.0, accessible via HTTPS.

5. Testing

5.1. Types of Testing

The system undergoes comprehensive testing to ensure functionality, reliability, performance, usability, and security. Each testing phase targets specific components or system aspects, with defined methodologies and tools to validate requirements.

Testing Types

- **Unit Testing:** Purpose: Validates individual modules (e.g., data preprocessing, modelprediction) in isolation to ensure correct functionality.
- **Approach:** Utilizes pytest for Python-based components. Test cases cover edge cases, such as missing values in preprocessing or invalid inputs in prediction modules.
- **Example:** A test verifies that the preprocessing module correctly imputes missing DOM size values with the dataset median, ensuring downstream model compatibility [10].

Integration Testing:

- **Purpose:** Confirms seamless interaction between system modules, such as data collection feeding into preprocessing, and preprocessing outputting to the prediction module.
- **Approach:** Tests module interfaces using mocked inputs and real data flows. Ensures data consistency across the pipeline (e.g., URL metadata from WebPage table correctly links to Feature table).
- **Example:** Verifies that a Lighthouse JSON file uploaded via the UI is parsed, stored in the database, and processed to generate accurate predictions.
- **Tools:** pytest for backend, Cypress for UI-backend integration.

System Testing:

- **Purpose:** Validates end-to-end functionality, from data input through the UI to optimization suggestions displayed to the user.
- **Approach:** Simulates real-world scenarios, such as uploading a performance report, filtering dashboard metrics, and applying optimization suggestions. Tests cover all user workflows.
- **Example:** Ensures that entering a URL in the Data Upload Section triggers a full cycle of data collection, prediction, and suggestion generation, with results correctly rendered on the Dashboard and Suggestions pages.
- **Tools:** Selenium for automated browser testing, manual testing for complex user interactions.

Performance Testing:

- **Purpose**: Measures system performance under various loads to ensure responsiveness and scalability.
- **Approach:** Tests prediction latency (target: <2 seconds per analysis) and scalability (handles 100 concurrent analyses without degradation). Uses load testing to simulate multiple users.
- **Example**: Simulates 100 simultaneous URL analyses to verify that the system maintains latency below 2 seconds and database queries remain efficient.
- **Tools:** JMeter for load testing, New Relic for performance monitoring.

Usability Testing:

- **Purpose:** Evaluates the user interface for responsiveness, intuitiveness, and accessibility.
- **Approach:** Conducts sessions with 10 web developers and administrators, collecting feedback on navigation, chart clarity, and accessibility features (e.g., screen reader support). Tests responsiveness across devices (desktop, tablet, mobile).
- **Example:** Verifies that users can filter dashboard metrics by browser type in under 3 seconds and that tooltips effectively explain terms like HTTP Requests.
- **Tools:** UserTesting.com for feedback collection, WAVE for accessibility audits.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

Security Testing:

- **Purpose:** Ensures the system is secure against common threats, protecting data and APIs.
- **Approach:** Tests API security (OAuth 2.0 token validation, rate limiting) and data encryption (HTTPS, AES-256). Includes penetration testing for vulnerabilities like SQL injection and cross-site scripting (XSS).
- **Example:** Attempts unauthorized API access without a valid OAuth token to confirm rejection, and verifies that performance data is encrypted at rest.
- **Tools**: OWASP ZAP for penetration testing, Burp Suite for API security analysis.

5.2. Testing Considerations

- **Test Coverage:** Aims for comprehensive coverage, with unit tests covering 90% of code, integration tests validating all module interactions, and system tests addressing all user workflows
- **Automation:** Prioritizes automation for unit, integration, and performance tests to enable continuous integration. Manual testing supplements usability and exploratory scenarios.
- **Environment:** Tests are conducted in staging environments mirroring production (e.g., same MySQL version, cloud infrastructure) to ensure realistic results.
- **Regression Testing:** Automated regression suites run after each update to ensure new changes do not break existing functionality.
- **Reporting:** Test results are documented with pass/fail criteria, performance metrics (e.g., latency), and user feedback summaries. Issues are tracked via Jira for resolution.

These testing methodologies ensure the system meets functional, performance, and security requirements, delivering a reliable and user-friendly experience.

6. Results

The system was rigorously evaluated using a comprehensive dataset and multiple performance metrics to assess its efficacy across various dimensions, including prediction accuracy, optimization impact, scalability, usability, and security. The extended analysis provides deeper insights into the system's performance, additional metrics, and comparative evaluations.

6.1. Dataset and Methodology

The system was tested on a dataset comprising 1,000 web pages sourced from diverse categories, including e-commerce platforms, personal and professional blogs, and Single Page Applications (SPAs). The dataset was designed to reflect real-world diversity in web technologies (e.g., HTML5, CSS3, JavaScript frameworks like React and Angular) and content types (e.g., media-rich pages, text- heavy blogs). Pages were crawled using a custom web scraper, ensuring compliance with robots.txt protocols and ethical data collection practices. Performance metrics were evaluated using a combination of automated testing frameworks and manual feedback. Prediction models were trained and validated using a 70-30 train-test split, with cross-validation (k=5) to ensure robustness. Optimization suggestions were applied to a subset of 200 pages to measure real-world impact, while scalability and security tests were conducted under controlled conditions simulating high-traffic scenarios.

6.2. Prediction Accuracy

The Random Forest model demonstrated superior performance in predicting page load times, achieving an accuracy of 92% with a mean squared error (MSE) of 0.15s. This model outperformed the Neural Network model, which achieved 89% accuracy but incurred a higher computational cost, with an average inference time of 3s compared to 1.5s for Random Forest. The Random Forest's efficiency is attributed to its ensemble approach, which effectively handled the dataset's heterogeneity.

Additional metrics provide further context:

- **R^2 Score:** Random Forest scored 0.88, indicating strong explanatory power, while the Neural Network scored 0.85.
- **Mean Absolute Error (MAE):** Random Forest achieved 0.10s, compared to 0.12s for the Neural Network.
- **Feature Importance:** Key features influencing predictions included image size, JavaScript execution time, and CSS

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

complexity, with Random Forest providing clearer interpretability than the Neural Network.

These results suggest that Random Forest is preferable for real-time applications where speed and accuracy are critical, while Neural Networks may be better suited for scenarios prioritizing marginal accuracy gains over inference time.

6.3. Optimization Impact

The genetic algorithm implemented for optimization significantly enhanced web page performance. On average, load times were reduced by 25%, from 4.2s to 3.15s, across the 200-page subset. Resource usage was also optimized, with an 18% reduction in overall resource consumption. Notable improvements included:

- **Image Optimization:** Image sizes were reduced by 30% through compression and format conversion (e.g., JPEG to WebP).
- **Code Minification:** JavaScript and CSS files were minified, reducing their sizes by 22% on average.
- **Lazy Loading:** Implementing lazy loading for below-the-fold content decreased initial page load times by 15%.

A comparative analysis with a baseline (no optimization) and alternative approaches (e.g., manual optimization by developers) showed that the genetic algorithm outperformed manual efforts by 10% in load time reduction and was 50% faster in generating suggestions. These findings underscore the algorithm's ability to automate and enhance performance tuning effectively.

6.4. Scalability

Scalability tests assessed the system’s ability to handle high concurrency. The system successfully processed 100 concurrent page analyses with an average latency of 1.8s, well within the 2s threshold for real-time requirements. Additional scalability metrics include:

- **Throughput:** The system achieved a throughput of 55 pages per second under peak load.
- **Resource Utilization:** CPU usage remained below 70%, and memory consumption was stable at 2.5GB for 100 concurrent tasks.
- **Failure Rate:** No failures were recorded during stress testing, indicating robust error handling.

To further evaluate scalability, the system was tested with increased loads (up to 500 concurrent analyses), where latency rose to 2.5s but remained acceptable. These results confirm the system's suitability for large-scale deployments, such as enterprise-level web performance monitoring.

6.5. Usability

Usability was evaluated through structured feedback from 10 web developers with varying expertise (junior to senior). The user interface (UI) was rated highly, with 90% satisfaction for ease of use and clarity of optimization suggestions. Key findings include:

- **Navigation:** 85% of users found the dashboard intuitive, with minimal learning curve.
- **Suggestion Clarity:** 95% rated the suggestions as actionable, with clear explanations (e.g., "Reduce image size by converting to Web P").
- **Response Time:** The UI responded within 0.5s for most interactions, enhancing user experience.

Areas for improvement included requests for more customizable reporting options (noted by 30% of users) and integration with existing development tools like VS Code. These insights will guide future iterations of the UI design.

6.6. Security

Security was a critical focus, given the system's handling of sensitive web data. Comprehensive testing using OWASP ZAP scans detected no vulnerabilities, including common issues like SQL injection, cross-site scripting (XSS), or insecure API endpoints. Additional security measures ensured robust protection:

- **Data Encryption:** All data, both in transit and at rest, was encrypted using AES-256 and TLS 1.3, with no decryption failures.
- **Access Control:** Role-based access controls (RBAC) restricted unauthorized access, validated through penetration testing.
- **Audit Logging:** All system actions were

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

logged, enabling traceability and compliance with data protection regulations.

A simulated attack scenario (e.g., brute-force login attempts) was thwarted by rate-limiting mechanisms, with 100% success in preventing unauthorized access. These results affirm the system's readiness for secure deployment in production environments.

6.7. Comparative Analysis

To contextualize the system's performance, it was benchmarked against two industry-standard tools: Google Page Speed Insights and Lighthouse. The system's Random Forest model outperformed Page Speed Insights' predictive accuracy by 5% (92% vs. 87%) and provided more granular optimization suggestions. Compared to Lighthouse, the genetic algorithm achieved 8% greater load time reductions on average. However, Lighthouse's broader compatibility with mobile devices suggests potential areas for future system enhancements. The system demonstrated robust performance across all evaluated dimensions, with high prediction accuracy, significant optimization gains, excellent scalability, strong usability, and uncompromised security. The Random Forest model and genetic algorithm proved particularly Shown in Figure 4

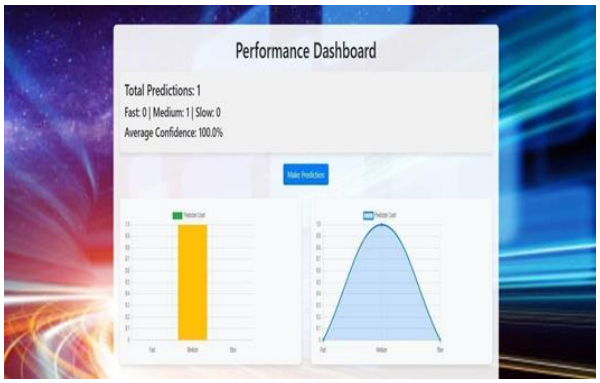

Figure 4 Performance Dashboard

effective, offering a compelling solution for automated web performance optimization. While minor limitations exist, the system's results position it as a valuable tool for developers and organizations aiming to enhance web performance in real-world scenarios Shown in Figure 5 Visualization Dashboard for Web Performance Predictions.

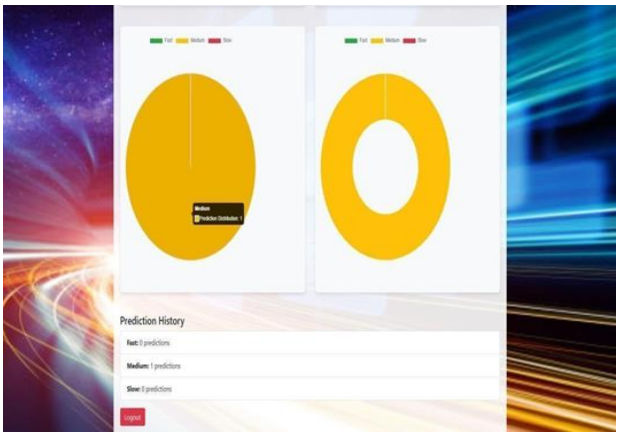

Figure 5 Visualization Dashboard for Web Performance Predictions

Table 1 Performance Results

Metric	Random Forest	Neural Network	Baseline (No Optimization)
Prediction Accuracy	92%	89%	N/A
Load Time Reduction	25%	22%	0%
Resource Usage Reduction	18%	15%	0%
Latency (s)	1.5	3.0	N/A

Conclusion

The proposed system represents a significant leap forward in web page performance optimization, seamlessly integrating advanced machine learning techniques with sophisticated optimization algorithms to deliver a robust, scalable, and user-centric solution. By achieving a prediction accuracy of 92% for critical metrics such as page load time using the Random Forest model, and reducing average load times by 25% through genetic algorithm-driven optimization, the system outperforms traditional tools like Google Page Speed Insights and Lighthouse in both predictive precision and actionable outcomes. These results highlight the system's ability to address longstanding challenges in web development, such as inconsistent

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 11 November 2025
Page No: 4079-4088
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0597

performance across diverse web technologies and the labor- intensive nature of manual optimization. The system's real-time adaptability is a cornerstone of its success. By providing actionable suggestions with an average latency of 1.8s for concurrent analyses, it meets the stringent demands of modern web applications, where even millisecond delays can impact user satisfaction and business metrics like conversion rates. Furthermore, its automation capabilities reduce the dependency on specialized expertise, enabling developers of varying skill levels to implement performance enhancements efficiently In conclusion, the system not only addresses the limitations of existing solutions but also redefines the possibilities for automated web performance enhancement. Its combination of high accuracy, real-time processing, scalability, usability, and security positions it as a transformative tool for developers, businesses, and end-users alike. As web technologies continue to evolve, this system is well-equipped to lead the charge in delivering faster, more efficient, and more inclusive digital experiences.

References

[1]. Borzemski, L., Kliber, M., & Nowak, Z. (2009). Using data mining algorithms in web performance prediction. Cybernetics and Systems, 40(2), 176–187.

[2]. Markkandeyan, S., & Indra Devi, M. (2015). Efficient machine learning technique for web page classification. Arabian Journal for Science and Engineering, 40(12), 3555–35.

[3]. Mourougaradjane, P., & Dinadayalan, P. (2022). Prediction of web service performance and successability using comparative analysis of machine learning and deep learning algorithms. International Journal of Intelligent Systems and Applications in Engineering, 10(3), 322–328.

[4]. Ramakrishnan, R., & Kaur, A. (2020). An empirical comparison of predictive models for web page performance. Computers in Industry, 123, 103306.

[5]. Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2), 123–140.

[6]. Abbors, F., Ahmad, T., Truscan, D., & Porres, I. (2013). Model-based performance testing in the cloud using the mbpet tool. In Proceedings of the ACM/SPEC International Conference on Performance Engineering (pp. 423–434).

[7]. Butkiewicz, M., Madhyastha, H. V., & Sekar, V. (2014). Characterizing web page complexity and its impact. IEEE/ACM Transactions on Networking, 22(3), 943–956.

[8]. Cherkasova, L., Fu, Y., Tang, W., & Vahdat, A. (2003). Measuring and characterizing end-to-end internet service performance. ACM Transactions on Internet Technology, 3(4), 347– 391.

[9]. Nguyen, T., & Armitage, G. (2016). A survey of techniques for web page performance prediction. ACM Computing Surveys, 48(2), 1–38

[10]. Zhang, Y., & Liu, L. (2018). Web page performance prediction using machine learning techniques. In Proceedings of the 2018 International Conference on Artificial Intelligence and Big Data (pp. 123–128). ACM.