Secure Systems Design and Technology Development. Veliko Tarnovo, Bulgaria
Proceedings of the International Scientific Conference, 2026. Volume III, 27-32

An Optimized Model for CPU Load Evaluation in the Execution of Code Structures and Script-Based Algorithms

Valentin Atanasov
Faculty of Technical Sciences
Konstantin Preslavsky University of Shumen
Shumen, Bulgaria
v.atanasov@shu.bg

Daniel Denev
Faculty of Technical Sciences
Konstantin Preslavsky University of Shumen
Shumen, Bulgaria
d.denev@shu.bg

***Abstract*. The introduction of models for studying the impact of script-based algorithms in web applications on overall CPU load should occupy an important place in the modern communication environment. However, there is a limited availability of software tools in this field. By exploring this subject, a team of researchers in the area of web applications developed a model with key characteristics representing scalar values of indicators that describe CPU load. This developed model for evaluating scalar time-based metrics has undergone evolution and optimization, expressed in increased accuracy of metrics, improved process stability, enhanced precision of results, and, not least, greater efficiency in user perception. This paper presents an optimized model for evaluating CPU load during the execution of code structures and script algorithms.**

Keywords: Benchmark CPU Algorithm, CPU load, WEB applications, JavaScript, Code Algorithms, DOM.

I. INTRODUCTION

The operation of modern web browsers is based on clearly defined stages of content processing and rendering, which together form the computational pipeline. Instead of simplified models that describe the browser as a collection of separate "machines," contemporary literature adopts a process-oriented approach.

In this context, the following main stages can be distinguished: resource fetching (networking), parsing of HTML and CSS documents, construction of the DOM and CSSOM tree structures, execution of JavaScript code by the JavaScript engine, style calculation, layout (reflow), and the final stage—rendering through painting and compositing.

CPU load during the execution of client-side scripts is directly related to the algorithmic complexity of these stages, particularly with respect to DOM operations, layout recalculation frequency, and the complexity of the executed JavaScript algorithms. In this context, the execution time of individual operations represents a key indicator of web application performance in a client-side environment.

There is a growing trend in the use of the scripting language JavaScript, with a steady increase [1], [2], [4]–[6] in web-based applications. This trend is driven, on the one hand, by the dynamic development of web technologies and the HTML5 suite, and on the other hand, by the total dominance of a key user characteristic—a high degree of interactivity demanded by a highly interactive generation.

With the rapid development of web technologies, JavaScript-based applications, and browser engines, there has been a continuous evolution of models and approaches for analyzing and optimizing computational performance in client-side environments. Each successive generation of benchmarking methodologies builds upon previous approaches by introducing improved mechanisms for measuring execution time, rendering performance, and resource utilization in increasingly complex web systems. One of the most effective directions in this field is the use of structured stress-testing models and deterministic measurement pipelines, which allow more accurate evaluation of CPU load and browser rendering behavior under controlled conditions. In earlier stages of research and practice, performance evaluation of web applications was primarily based on simple timing functions and isolated execution measurements. However, these approaches often lacked sufficient precision, reproducibility, and the ability to capture the full complexity of modern browser pipelines, including the interactions between JavaScript execution, DOM manipulation, CSSOM processing, and rendering stages such as layout, reflow, and repaint. With the increasing complexity of web applications and the growing demand

Online ISSN 3044-7771
https://doi.org/10.68302/std2026.vol3.137

for highly interactive user experiences, the need for more advanced and systematic benchmarking techniques has become evident.

In our scientific research, we review several key contributions in this field. Canay and Kocabıçak [1] explore augmented web usage mining and user experience optimization, emphasizing the importance of enriched analytics data for understanding user interaction patterns in web environments. Xavier [2] presents a quantitative analysis of global web usage, providing insight into large-scale behavioral trends that influence modern web application design and optimization strategies. Atanasov [3] focuses on approaches for evaluating code algorithms in web applications, introducing foundational concepts related to performance measurement and computational analysis in browser-based systems.

Additionally, industry-driven statistical reports [4], [5], [6] highlight the dominant role of JavaScript and related web technologies in contemporary software development, confirming the necessity of efficient performance evaluation tools. Standardization documents such as the DOM Living Standard [7] and CSS Object Model specifications [8] provide the structural basis for understanding how browsers construct and manipulate document representations. Furthermore, Mozilla's documentation on browser internals [9] offers essential insight into the rendering pipeline and the stages influencing runtime performance.

Collectively, these studies demonstrate that modern performance evaluation in web environments requires a combination of empirical measurement, architectural understanding, and statistical analysis. The proposed model extends these foundations by introducing a sandboxed, repeatable, and statistically stabilized benchmarking approach tailored for DOM and JavaScript stress-testing scenarios [11], [14].

II. Materials And Methods

Experimental Setup – Code Component

A university research team analyzed a developed model of a web-based prototype application that examines CPU load during the execution of JavaScript code algorithms for the following set of indicators [3]:

- measurement of performance metrics;
- JavaScript execution time;
- render time;
- layout time;
- reflow / repaint measurements;
- analysis of inefficient DOM manipulations;
- asynchronous operation timing;
- graphical visualization of timing data;
- system logging;
- isolated (sandboxed) execution within an iframe.

The architecture of the developed application was based on two interconnected models—the Document Object Model (DOM) [7] and the Cascading Style Sheets Object Model (CSSOM) [8]. The main set of approaches included:

- execution of user-provided HTML and JavaScript code within a sandboxed iframe;
- isolation of user code;
- measurement of HTML document load time and JavaScript execution time;
- use of performance.now();
- use of setTimeout and setInterval;
- blocking of Promise and fetch;
- real-time updates;
- suitability for micro-benchmarks;
- interception of console.log;
- capability to stop dynamic content updates, clear, and reload the virtual environment.

As functionally complete injected code components, a web-based application featuring a complex DOM structure and a JavaScript algorithm simulating an educational TCP three-way handshake scenario was used.

III. Results And Discussion

Without going into the details of the subsystem modules in the architecture, a summarized set of analytical results from the empirical study of the web-based benchmark prototype is presented:

- The evaluation module of the studied prototype produced observed data in accordance with the designed functionality, along with partial conclusions regarding the use of setInterval instead of requestAnimationFrame, and the resulting logical relationship between Reflow and Repaint;
- During the code evaluation, for $x \in X$ a failure $F(x)$ of the client agent was observed, caused by the following logical chain:

$$\forall x(\exists x(H(x) \wedge E(x, 250) \wedge L(x) \wedge D(x) \wedge C(x)) \rightarrow F(x)$$

where:

$H(x) = x$ heavy engine

$E(x, n) = x$ has n executions

$L(x) = x$ uses setInterval and has memory leaks

$D(x) = x$ the algorithm performs frequent DOM rebuilds

$C(x) = x$ the algorithm performs canvas rendering

$F(x) = x$ leads to crash / high CPU usage / hang

By analyzing the empirical phase, the following conclusions were drawn for the web-based benchmark prototype:

1. **Monolithic Execution**
    - The examined model directly executes JavaScript (new Function(js)), which blocks the UI and makes the benchmark unstable.
    - All executions of user code should be sandboxed, for example in a Web

Worker, in order to isolate and protect the Main Thread and the interface.

2. **Mixing of Responsibilities**
   - The benchmark combines: UI, code execution, statistics, and quality analysis.
   - Code logic, interface, analysis, and statistics should be modular, with clearly separated responsibilities.

1. **Unoptimized Loops and Visualization**
   - In the code, each iteration updates the DOM/graphics, which is CPU-intensive.
   - Batching and throttling in visualization reduce unnecessary load.

2. **Lack of Resource Lifecycle Management**
   - Multiple intervals and DOM elements are created without cleanup, leading to memory leaks.
   - Each resource creation should have a clearly defined lifecycle: initialization → usage → cleanup.

3. **Inefficient Statistics and Reporting**
   - Statistics such as p95, p99, and standard deviation are not collected, which complicates result interpretation.
   - Collecting comprehensive statistical indicators improves performance evaluation.

4. **Risk of Problematic Code**
   - Without static analysis, checks for infinite loops, setInterval without clear, event listener leaks, and DOM manipulation in loops, the benchmark may crash.
   - A static analyzer and heuristics for code smells protect the tool from failure under heavy code.

5. **Approximate CPU Load Estimation**
   - JavaScript does not provide a direct API for CPU usage.
   - Measurement should be based on execution time/wall time, with proper adjustment for frame budget.

Based on the obtained conclusions, an optimization phase was initiated, whose architecture is presented in Fig.1.

The architecture shown in Fig.1 covers the following main tasks:

- construction of a deterministic sandbox environment for execution, which guarantees identical initial conditions and isolation of user code;
- creation of a signal amplification mechanism that makes micro-measurements of JavaScript execution reliable and measurable;
- implementation of a two-layer measurement approach that separates JavaScript execution from render/pipeline latency in the browser;
- development of a statistical pipeline for noise reduction, including aggregation and stabilization of results through metrics such as median, percentile values, and variance, in order to obtain repeatable and interpretable performance results.

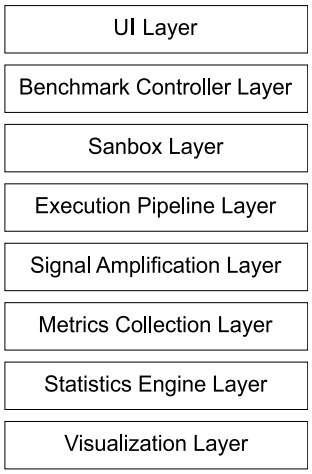

Fig. 1. Architecture of the Optimized Model for a Repeatable DOM & JS Stress Harness with Statistical Output.

As a result of these tasks, the model for a repeatable DOM & JS stress harness with statistical output was further optimized. This optimization phase was followed by code development, which was subjected to a series of tests. One of the main advantages of the optimized code is the high-precision measurement of individual JavaScript operations through the use of performance.now(), which provides millisecond-level accuracy. To support the above with statistical arguments, we will compare two methods, the first, removeOutliers(5%), is applied in the non-optimized model, while the second (10%) is applied in the optimized model. The removeOutliers(5%) method, after sorting, removes one value from each extreme (the lowest and the highest), which may be useful for small datasets but is not always sufficiently valid for larger datasets with many deviations. The second method, trim(10%), removes 10% of the values from each extreme after sorting and provides improved data stability, making it more suitable for realistic and robust measurements in browser testing and performance analysis. Both methods aim to eliminate extreme anomalies that may introduce noise and distort statistical indicators such as the median, standard deviation, and percentile values. The proposed algorithm of the mathematical apparatus used to remove data points that significantly deviate from the main distribution and are not considered representative of the normal behavior of the data is presented in Table 1 [12].

TABLE 1 SCRIPT-BASED ALGORITHM OF THE MATHEMATICAL APPARATUS FOR TRIMMING THE LOWEST AND HIGHEST DATA VALUES

```
// Data trimming method – removes the lowest and highest 10% of values
  function trim(arr, ratio = 0.1) {
// Check if the dataset has fewer than 10 values; in that case, trimming is not applied
  if (arr.length < 10) return arr;
// Sort the data
  const sorted = [...arr].sort((a, b) => a - b);
// Calculate the number of values to remove (10%)
  const cut = Math.floor(arr.length * ratio);
// Return the data without the lowest and highest 10%
  return sorted.slice(cut, sorted.length - cut);
}
```

To ensure the validity of the statistical analysis, an identical population of 300 input values is assumed for both methods. Using statistical processing techniques, the following procedure is applied:

- Generation of 300 random data points from a normal distribution.
- Application of the two data filtering methods: removeOutliers(5%) and trim(10%).
- Calculation of statistical indicators: median, standard deviation, p95, standard error, and confidence interval.
- Conducting a Levene's test to assess differences in variability between the two methods.
- Comparing precision through the standard error (SE) and the width of the confidence interval.

The conducted test generates output data summarized in Table 2. Figure 2, Figure 3, Figure 4, Figure 5 compare the key statistical metrics between the two data processing approaches: removeOutliers (5%) and trim (10%). They visually demonstrate how trimming improves stability, reduces variability, and provides more reliable estimates across all evaluated indicators.

TABLE 2 GENERATED OUTPUT DATA FROM THE STATISTICAL ANALYSIS OF THE METHODS REMOVEOUTLIERS (5%) AND TRIM (10%)

Metric	removeOutliers (5%)	trim (10%)	Difference / Improvement
Median	0.05922	0.05922	0.00% (stable)
Standard deviation	0.75953	0.63363	-16.58% noise
p95 (Upper bound)	1.16111	0.95447	-17.80%
Standard error (SE)	0.04622	0.04090	-11.52% error
95% confidence interval	[-0.109, 0.073]	[-0.103, 0.058]	More focused range
Confidence interval width	0.18201	0.16114	-11.47% narrower

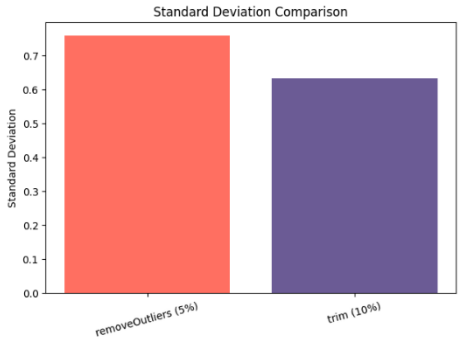

Fig. 2. Standard Deviation Comparison.

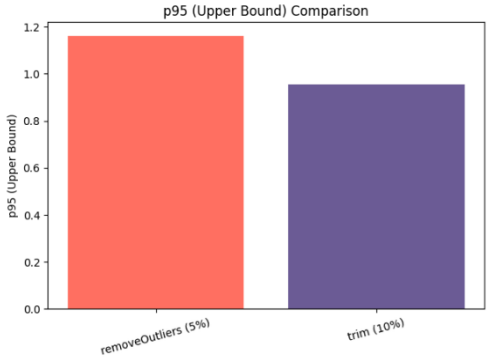

Fig. 3. Standard Deviation Comparison.

Fig. 4. Standard Error (SE) Comparison.

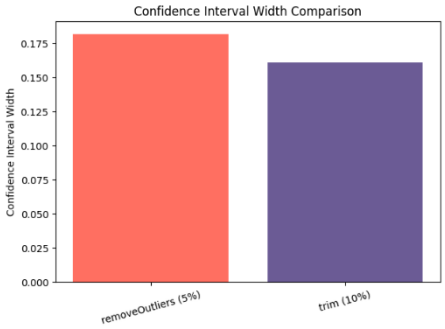

Fig. 5. Confidence Interval Width Comparison.

With the data thus formed and summarized in Table 2, the probable conclusions from the analysis of these data are presented in Table 3.

TABLE 3 ANALYSIS OF THE STATISTICAL RESULTS FROM THE COMPARISON OF THE METHODS REMOVEOUTLIERS (5%) AND TRIM (10%)

Evidence	Value / Result	Conclusions
Levene Test (p-value)	0.00545	The difference is statistically significant ($p < 0.05$). The factor of randomness is excluded.
Stable deviation	16.58% lower	Data under trim(10%) is significantly more homogeneous and cleaned from anomalies.
Precision (SE)	11.52% improvement	The estimate of the mean value is more reliable and robust to new data.
Reliability (CI)	11.47% narrower interval	Higher confidence in predicting results.

Based on the analysis of the processed data, the efficiency index ε is defined as a relative measure. It serves as an indicator of accuracy in terms of reduced noise when transitioning from one filtering method to another. This index is expressed by the following mathematical formulation:

$$\varepsilon = \frac{\sigma_{5\%} - \sigma_{10\%}}{\sigma_{5\%}}$$

where:

$\sigma_{5\%}$ is the standard deviation with 5% outlier removal (0.75953)

$\sigma_{10\%}$ is the standard deviation with 10% trimming (0.63363)

The final conclusion can be summarized as follows:

This index $\varepsilon \approx 16.58\%$ can be interpreted as evidence of the higher precision and stability of the trim(10%) method within the optimized model. There is also a reduction in uncertainty when using the trim(10%) method, which reduces data variability by over 16%. A higher efficiency index ε implies that in subsequent iterations of the experiment with new sets of 300 data points, the results obtained using trim(10%) are less likely to deviate from the original baseline compared to those obtained using the removeOutliers(5%) method. The trim(10%) method achieves a 16.58% improvement in the reliability of the optimized model, demonstrated through statistical variance optimization.

IV. CONCLUSION

Based on the analysis of the prototype, the authors conclude that the examined system provides a deterministic measurement pipeline and a browser sandbox execution environment, in which the main execution flows are separated into execution (Sandbox & Pipeline), measurement (Amplification & Metrics), interpretation (Statistics), and presentation (Visualization). The evolution of the mathematical apparatus for trimming extreme values (outliers), traditionally implemented through statistical software such as R, Python (with libraries like NumPy or Pandas), and MATLAB, has led to its implementation in JavaScript as well. This enables web application developers to perform statistical analyses directly within the browser environment without requiring server-side solutions.

In addition to the conclusions drawn, it can be noted that the proposed model demonstrates the possibility of portability and adaptation to different types of web applications and load scenarios, including both synthetic benchmark tests and real user interactions. This makes the proposed model suitable not only as a research tool, but also as a foundation for future systems for performance monitoring and analysis in client-side environments.

Of additional significance is the fact that separating the core processes into distinct logical layers enables a clearer interpretation of results and easier identification of sources of performance degradation. In this way, the model creates prerequisites for improved diagnosis of complex interactions between JavaScript execution and the browser rendering pipeline.

From the perspective of future development, the model can be extended through the introduction of automated mechanisms for adaptive load generation, where test scenarios change dynamically depending on the observed behavior of the system. This would enable more realistic load simulation and higher sensitivity to edge cases.

Furthermore, the integration of more advanced analytical methods and intelligent algorithms for pattern recognition in the collected metrics could contribute to a deeper understanding of the dependencies between different execution layers. In this context, the model has the potential to evolve into an autonomous system for evaluating and predicting performance in web-based environments [10], [13].

V. ACKNOWLEDGEMENT

In this section of the present study, the authors would like to express their sincere gratitude to all colleagues from the department for their continuous support, constructive discussions, and valuable technical feedback throughout the development and optimization of the proposed web-based benchmarking model. Their expertise and recommendations were essential for refining the methodological approach and improving the overall reliability of the experimental framework. We also extend our deep appreciation to our families for their constant encouragement, understanding, and patience, which provided strong motivation during the research and development process. This publication is funded by the fund "Support for Publishing Publications in Journals with Impact Factor (IF) and Impact Rank (SJR)" of Konstantin Preslavsky University of Shumen, Republic of Bulgaria.

REFERENCES

[1] Ö. Canay and Ü. Kocabıçak, "Augmented web usage mining and user experience optimization with CAWAL's enriched analytics data," *International Journal of Human–Computer Interaction*, vol. 41, no. 11, pp. 7152–7171, 2025, doi: https://doi.org/10.1080/10447318.2025.2495839.

[2] H. S. Xavier, "The web unpacked: a quantitative analysis of global web usage," in *Proc. 20th Int. Conf. Web Information Systems and Technologies (WEBIST)*, 2024, pp. 183–190, doi: https://doi.org/10.5220/0012905900003825.

[3] V. T. Atanasov, "An approach to evaluating code algorithms in web applications," *Annual of Konstantin Preslavsky University of Shumen*, vol. XV E, pp. 25–31, 2025, doi: https://doi.org/10.46687/YT25VTA.

[4] "Comparison of the usage statistics of JavaScript for websites," W3Techs, 2026. [Online]. Available: https://w3techs.com/technologies/comparison/cp-javascript. [Accessed: Apr. 6, 2026].

[5] "Most used programming languages among developers worldwide as of 2025," Statista, 2025. [Online]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. [Accessed: Apr. 6, 2026].

[6] "Most popular technologies," Stack Exchange Inc., 2025. [Online]. Available: https://survey.stackoverflow.co/2024/technology. [Accessed: Apr. 6, 2026].

[7] Web Hypertext Application Technology Working Group, "DOM Living Standard," 2026. [Online]. Available: https://dom.spec.whatwg.org/. [Accessed: Apr. 6, 2026].

[8] Mozilla Corporation, "CSS Object Model (CSSOM)," 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model. [Accessed: Apr. 6, 2026].

[9] Mozilla Corporation, "How browsers work," 2026. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Performance/Guides/How_browsers_work. [Accessed: Apr. 6, 2026].

[10] Ts. Tsankov, L. Staneva, I. Vardeva, and D. Iliev, "Generalized net model of a communication network using the Port Knocking method," in *Proc. Scientific Conf. MATTEH 2022*, vol. 2, Shumen, Bulgaria, 2022, pp. 29–34.

[11] Ts. Tsankov, L. Staneva, I. Vardeva, and D. Iliev, "Application of a generalized net model in the communication of two independent traffic," in *Proc. 15th Int. Scientific Conf. Environment. Technology. Resources*, vol. II, Rezekne, Latvia, 2024, pp. 293–296.

[12] V. Stoyanova, "Research of the characteristics of a steganography algorithm in images when using different alphabet," in *Proc. 15th Int. Scientific and Practical Conf.*, vol. IV, 2024, p. 259.

[13] V. T. Stoyanova, "Steganography system using more LSBs," *ENTRENOVA – Enterprise Research Innovation*, vol. 4, no. 1, pp. 168–174, 2018.

[14] K. Kalchev and Ts. Tsankov, "Use of the decision tree method in analysis of student data from surveys," *Annual of Konstantin Preslavsky University of Shumen*, vol. X E, pp. 72–75, 2020.