IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 13, Issue 7, July 2024

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.423

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

Volume 13, Issue 7, July 2024

|DOI: 10.15680/IJIRSET.2024.1307032|

Harnessing WebAssembly for Enhanced Web Development: A JavaScript Comparison

Swati Chopade, Aakash Jain, Sandeep Chopade.

Professor, Department of Master of Computer Applications, Veermata Jijabai Technological Institute, Matunga, Mumbai, India

P.G. Student, Department of Master of Computer Applications, Veermata Jijabai Technological Institute, Matunga, Mumbai, India

Assistant Professor, Department of Mechanical Engineering, KJSCE, Mumbai, India

ABSTRACT: Performance and efficiency are critical in the evolving web landscape. This study explores the potential of WebAssembly (Wasm) for modern web development. By conducting a series of benchmarks on desktop and mobile platforms across three major browsers—Google Chrome, Mozilla Firefox, and Microsoft Edge, it aims to provide a comprehensive performance analysis of WebAssembly compared to JavaScript. Benchmarking involves JavaScript and WebAssembly algorithms by implementing and comparing various algorithms that are either computationally intensive and/or run with large input sizes. The benchmarks measure execution speed, memory usage, and rendering efficiency. Experimental results indicate that the current WebAssembly standard significantly outperforms JavaScript in raw computational tasks, often delivering 2-5 times faster performance. However, JavaScript retains an edge in DOM manipulation tasks due to its inherent integration with the browser environment. The findings underscore the complementary nature of WebAssembly and JavaScript. While WebAssembly excels in computationally heavy tasks, JavaScript remains faster in DOM manipulation. This study concludes that strategic integration of both technologies can lead to optimised web application performance, paving the way for more advanced and responsive web experiences.

KEYWORDS: WebAssembly, JavaScript, AssemblyScript, Performance Benchmarking, Modern Web Development.

I. INTRODUCTION

In recent years, web development has significantly transformed, driven by the growing demand for high-performance, interactive, and immersive web applications. JavaScript, as the de facto language of the web, along with its numerous frameworks is instrumental in shaping the digital landscape. Its versatility and ubiquity have made it a cornerstone of modern web development, enabling developers to create dynamic and responsive user interfaces. However, despite its widespread adoption and continual evolution, JavaScript faces inherent limitations, particularly in handling computationally intensive tasks and its memory footprint.

WebAssembly (Wasm) having its first release in 2017 and becoming a W3C standard in 2019, is a promising technology designed to address these challenges. As a binary instruction format, WebAssembly allows code written in high-level statically-typed languages like Rust, Go & AssemblyScript to be compiled into a compact and efficient binary format. This format can be executed at near-native speed by modern web browsers, thus promising to deliver substantial performance improvements over JavaScript. By eliminating the need for parsing and interpretation that characterizes JavaScript execution, WebAssembly offers faster load times and enhanced performance, especially for computationally demanding tasks.

This study aims to conduct a comprehensive performance analysis of WebAssembly in comparison to JavaScript. The study focuses on evaluating the execution speed, memory efficiency, and rendering capabilities of both technologies through a series of benchmarks. These benchmarks include matrix multiplication - quite a common algorithm in computer science with a large number of applications, determinant calculation, Fibonacci series generation, mandelbrot set rendering, and image filtering—each selected to test different aspects of computational performance and rendering efficiency.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

Volume 13, Issue 7, July 2024

|DOI: 10.15680/IJIRSET.2024.1307032|

With this research, we try to answer the following questions:
RQ. 1: What is the current state of WebAssembly as to JavaScript?

RQ. 2: What are the cases in which one is faster than another? How much faster or slower is it for those cases?

RQ. 3: For each platform (desktop and mobile), what is the browser/engine that provides the best WebAssembly performance?

II. LITERATURE SURVEY

WebAssembly (Wasm), announced in 2015 and released as a minimum viable product (MVP) in 2017, became a W3C standard in 2019. [5] Despite its relatively recent introduction, WebAssembly has generated considerable interest within the web development community due to its promise of near-native performance in web applications. However, the body of research on WebAssembly remains limited, with many studies presenting contradictory or incomplete findings. [3] This literature review aims to provide a comprehensive overview of the existing research on WebAssembly, highlighting key studies and their contributions to understanding its performance characteristics relative to JavaScript.

One of the seminal studies on WebAssembly is "WebAssembly versus JavaScript: Energy and Runtime Performance" by Joao De Macedo et al. (2022). [1] This study presents a systematic examination of WebAssembly's energy consumption compared to JavaScript, using Intel's Running Average Power Limit (RAPL) for measurements. The authors observed a 30% higher energy efficiency in their WebAssembly implementations of benchmarks against JavaScript, concluding that real-world implementations would likely follow the same pattern. However, the study has several limitations: it focuses extensively on energy consumption at the expense of a thorough runtime performance comparison. Additionally, the benchmarks were written in C and compiled into WebAssembly using Emscripten, which does not reflect industry practices where languages like Rust, AssemblyScript, or Golang are more commonly used.

Another notable study is "Empowering Web Applications with WebAssembly: Are We There Yet?" by Weihang Wang (2021). [2] This paper delves deeper into the performance aspects of WebAssembly versus JavaScript. Using thirty PolyBenchC programs, which are computationally heavy, the study compares the performance of WebAssembly and JavaScript across major browsers. The results indicate that WebAssembly programs are faster for smaller inputs, but as the input size increases, JavaScript becomes faster. The study also notes that WebAssembly consumes significantly more memory than JavaScript, attributing this to the lack of a garbage collector in WebAssembly at the time. However, the addition of a garbage collector in subsequent updates to WebAssembly has likely mitigated this issue. [6] This study also measures the performance of a single web browser which is V8 powered by Google Chrome.

In addition to academic research, various case studies and practical implementations provide valuable insights into WebAssembly's performance. One such example is the collaborative design tool Figma, which utilizes WebAssembly to enhance performance and scalability. Figma's engineering team identified performance-critical areas within their application, such as rendering complex design elements and handling real-time collaboration updates. By rewriting performance-sensitive portions of their codebase in Rust and compiling them to WebAssembly, Figma achieved significant improvements in rendering performance and real-time collaboration efficiency. [10]

The mixed and sometimes contradictory findings in existing research highlight the complexity of evaluating WebAssembly's real-world impact. While some studies emphasize WebAssembly's superior performance in specific benchmarks, others point out its limitations, such as increased memory consumption and slower performance for large inputs. Furthermore, the rapid evolution of WebAssembly suggests that current conclusions may quickly become outdated as the technology continues to develop.

In addition to formal studies, anecdotal evidence and community experiences also contribute to the understanding of WebAssembly's capabilities. For instance, a StackOverflow user reported that their WebAssembly implementation was 300 times slower than the JavaScript counterpart, highlighting the variability in performance outcomes based on implementation details and specific use cases. [11] Similarly, the demise of ZapLib, an open-source library aimed at speeding up web applications using Rust and WebAssembly, underscores the challenges developers face in realizing the full potential of WebAssembly. [9]

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

Volume 13, Issue 7, July 2024

|DOI: 10.15680/IJIRSET.2024.1307032|

Given the evolving nature of WebAssembly and the mixed results in current research, this study aims to provide a more comprehensive and up-to-date performance analysis. By conducting benchmarks across a range of computationally intensive tasks and comparing the performance of WebAssembly and JavaScript in different browsers and on different platforms, this research seeks to offer clearer insights into the practical benefits and limitations of WebAssembly. The findings will contribute to a deeper understanding of how WebAssembly can be leveraged to enhance web application performance, complementing existing JavaScript implementations and paving the way for future innovations in web development.

III. PROPOSED METHODOLOGY AND DISCUSSION

This study conducts a comparative performance analysis of WebAssembly and JavaScript by implementing and benchmarking five computationally intensive tasks: matrix multiplication, determinant calculation, Fibonacci series generation, Mandelbrot set rendering, and image filtering. The benchmarks were chosen to assess various aspects of performance, including raw execution speed, memory efficiency, and rendering capabilities.

- Implementation:

WebAssembly: AssemblyScript 0.27.27 was used to write the source code, which was then compiled into WebAssembly with the flags of the highest level of optimization. The node version used for compilation was Node 18.20.3 (LTS)
JavaScript: The equivalent algorithms were implemented in ES6 syntax.

- Experimental Setup:

Platforms: The benchmarks were run on both a desktop and a mobile device.
Desktop: A 64-bit Windows 10 machine with an AMD Ryzen 4600H processor, 8GB DDR4 RAM, and a 256GB SSD.
Mobile: A device running on a Snapdragon 855+ processor with Android 11 and 6GB of RAM.
Browsers: Google Chrome, Mozilla Firefox, and Microsoft Edge were used for testing on both platforms.

- Benchmarks:

1. Matrix Multiplication: Randomly generated large matrices of dimensions i x j and j x k were multiplied, and execution times were measured for both WebAssembly and JavaScript. This measured raw computational speeds.
2. Determinant Calculation: A matrix of size ‘n x n x n’ with random values was used to compute the determinant recursively. This was a healthy balance of raw computation with memory efficiency.
3. Fibonacci Series Generation: The nth Fibonacci number was calculated using a recursive approach which was executed in exponential runtime to check memory optimization.
4. Mandelbrot Set Rendering: The Mandelbrot set was rendered to an HTML canvas to evaluate computational and rendering performance.
5. Image Filtering: A grayscale filter was applied to images of size 1920 X 1080 and 2140 X 1440 to measure render speeds in the browser.

- Performance Measurement:

Execution times and memory usage were measured using the native 'performance' API in browsers. Results were compared across different browsers and platforms to identify the best-performing scenarios for each technology. For each browser across each platform, the result was taken as an average of ten separate executions.
This methodology provides a thorough evaluation of WebAssembly's performance relative to JavaScript, highlighting the strengths and limitations of both technologies in various computational contexts.

IV. EXPERIMENTAL RESULTS

The benchmarks conducted in this study reveal distinct performance characteristics of WebAssembly and JavaScript across various computational tasks.

1. Matrix Multiplication:

WebAssembly demonstrated a significant performance advantage over JavaScript for the matrix multiplication benchmark. On desktop platforms, WebAssembly was approximately 2.5 times faster on average than JavaScript. This

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

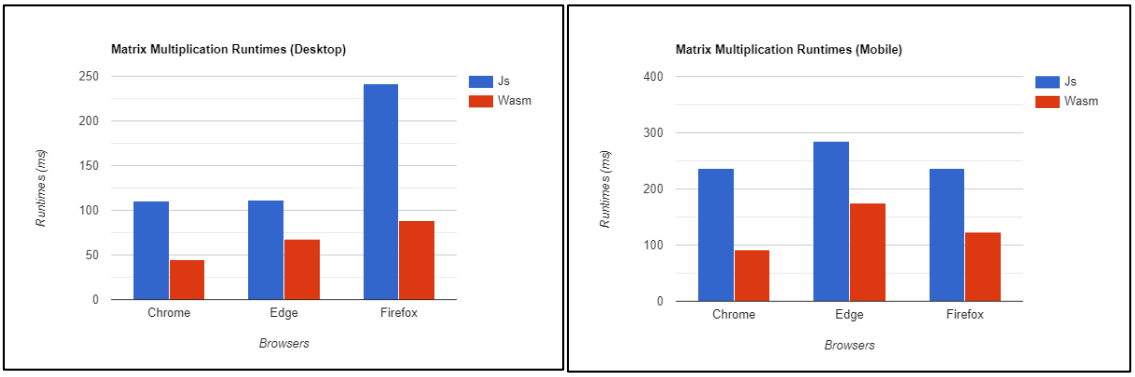
Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

Volume 13, Issue 7, July 2024

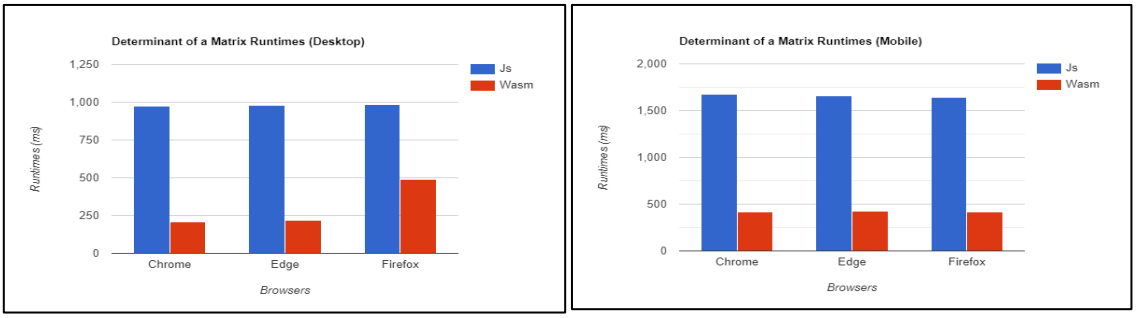
|DOI: 10.15680/IJIRSET.2024.1307032|

performance gap was consistent across different browsers, with minor variations. On mobile platforms, the performance improvement was more pronounced in Google Chrome, where WebAssembly was three times faster, while the improvements in Firefox and Edge were about 0.75 and 0.5 times respectively.



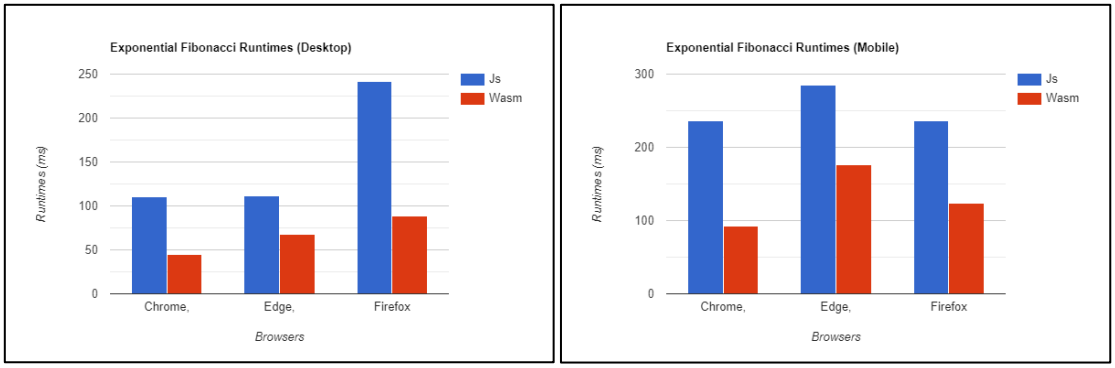
2. Determinant Calculation:

Similar to matrix multiplication, WebAssembly outperformed JavaScript in determinant calculation, showcasing an average speed improvement of 4 times on desktop and 5 times on mobile devices. This benchmark was particularly demanding due to its recursive nature and factorial time complexity, but the addition of a garbage collector in WebAssembly significantly enhanced its efficiency. All browsers exhibited similar performance improvements for WebAssembly.



3. Fibonacci Series Generation:

In the exponential Fibonacci series benchmark, WebAssembly showed a substantial performance boost, being 2.5 times faster than JavaScript on average for inputs ranging from 33 to 37. This translates to handling approximately 1.374 x 10^11 steps efficiently. On mobile devices, WebAssembly was about twice as fast, with Mozilla Firefox surprisingly performing as efficiently as other browsers due to heavy optimizations in its Gecko engine.



International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

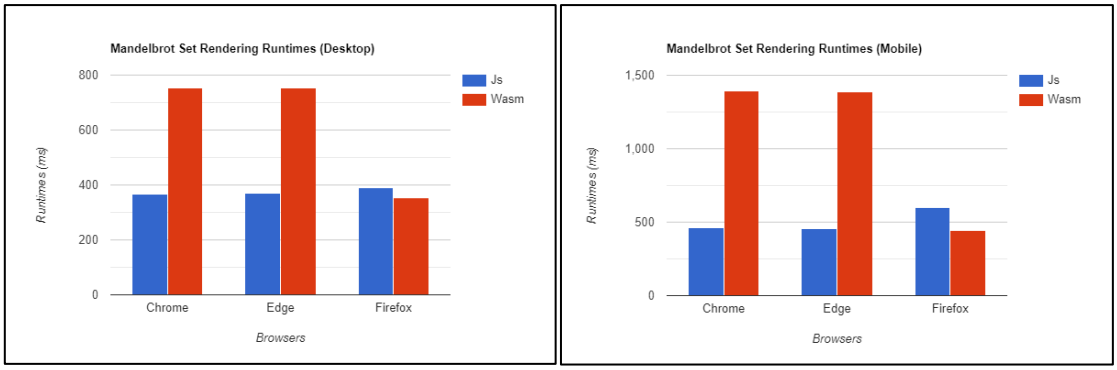
Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

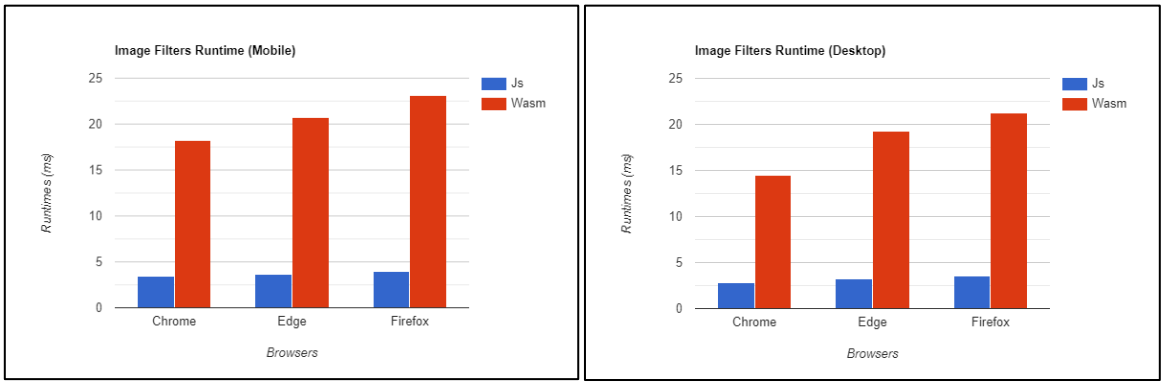
Volume 13, Issue 7, July 2024

|DOI: 10.15680/IJIRSET.2024.1307032|

4. Mandelbrot Set Rendering:
The Mandelbrot set benchmark, which combines computational and rendering tasks, highlighted JavaScript's advantage in DOM manipulation. JavaScript was almost twice as efficient as WebAssembly in rendering the Mandelbrot set on Chromium-based browsers (Chrome and Edge). Interestingly, Firefox's Gecko engine optimized WebAssembly code better, making it slightly faster than JavaScript in this browser. These results suggest that while WebAssembly excels in computation, JavaScript's integration with the DOM remains a critical factor in rendering tasks as the data copy time is a significant factor countering WebAssembly's efficient computations.



5. Image Filtering:
The image filtering benchmark underscored JavaScript's superiority in rendering speed. JavaScript implementations were 10 to 20 times faster than WebAssembly across various image sizes and orientations. For example, a grayscale filter application in WebAssembly took around 23ms, which is significantly slower compared to JavaScript's 2.4ms. This disparity is primarily due to WebAssembly's current limitation in accessing the DOM, requiring data to be copied to JavaScript, which diminishes its computational speed advantage.



Following is the summary of the results:

Platform	Browser	Matrix Multiplication (ms)			Determinant (ms)			Exp. Fibonacci (ms)		
		JS	Wasm	Diff	JS	Wasm	Diff	JS	Wasm	Diff
Desktop	Google Chrome	110.8	44.9	2.47x	974.7	210	4.64x	110.2	45.3	2.43x
	Microsoft Edge	111.1	67.5	1.65x	984.8	217	4.54x	114.3	69.4	1.64x
	Mozilla Firefox	241.7	88.3	2.73x	977.6	458	2.13x	243.6	92.7	2.63x
Mobile	Google Chrome	236.5	92.5	2.55x	1676.7	415	4.04x	236.2	93.2	2.53x
	Microsoft Edge	285	175.67	1.62x	1658.6	424.3	3.9x	285.3	174.3	1.64x

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.423

|www.ijirset.com |A Monthly, Peer Reviewed & Referred Journal| e-ISSN: 2319-8753; p-ISSN: 2347-6710|

Volume 13, Issue 7, July 2024

|DOI: 10.15680/IJIRSET.2024.1307032|

	Mozilla Firefox	236	123.3	1.91x	1640	1046.7	1.57x	239.8	122.7	1.95x

Platform	Browser	Mandelbrot Set (ms)			Filters (ms)		
		JS	Wasm	Diff	JS	Wasm	Diff
Desktop	Google Chrome	370.4	740.7	0.50x	2.83	14.5	0.20x
	Microsoft Edge	372.3	741.6	0.50x	3.25	19.3	0.17x
	Mozilla Firefox	393.3	360.1	1.09x	3.51	21.3	0.16x
Mobile	Google Chrome	460.4	1390.4	0.33x	3.44	18.2	0.19x
	Microsoft Edge	461.3	1388.2	0.33x	3.63	20.7	0.17x
	Mozilla Firefox	585.6	448.3	1.3x	3.99	23.1	0.17x

All the values in JavaScript (JS) and WebAssembly (Wasm) columns indicate average execution time in milliseconds.

V. CONCLUSION

This study evaluated the performance of the current state of WebAssembly (Wasm) versus JavaScript across various computational and rendering tasks. The benchmarks revealed that WebAssembly significantly outperforms JavaScript in raw computational tasks, demonstrating execution speeds 2 to 5 times faster. The addition of a garbage collector in WebAssembly has further improved its efficiency in handling memory-intensive operations.

JavaScript, however, remains superior in tasks involving extensive DOM manipulation. Despite WebAssembly's computational speed, its slower rendering times highlight its current limitations in direct DOM access. This is the 'boundary-crossing' problem where the time required to copy the input from DOM and return the results is more than the performance benefit currently provided by WebAssembly.

We conclude that WebAssembly and JavaScript have complementary strengths. WebAssembly excels in computational tasks, while JavaScript is essential for efficient DOM manipulation. A hybrid approach leveraging both technologies can optimize web application performance, offering a roadmap for developers to enhance user experiences and achieve greater efficiency in web development.

REFERENCES

1. J. De Macedo, R. Abreu, R. Pereira and J. Saraiva, "WebAssembly versus JavaScript: Energy and Runtime Performance," 2022 International Conference on ICT for Sustainability (ICT4S), Plovdiv, Bulgaria, 2022, pp. 24-34, doi: 10.1109/ICT4S55073.2022.00014.
2. W. Wang, "Empowering Web Applications with WebAssembly: Are We There Yet?," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1301-1305, doi: 10.1109/ASE51524.2021.9678831.
3. Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. 2019. Not so fast: analyzing the performance of webassembly vs. native code. In Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19). USENIX Association, USA, 107–120
4. Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the web up to speed with WebAssembly. SIGPLAN Not. 52, 6 (June 2017), 185–200.
5. W3C Press Release, https://www.w3.org/press-releases/2019/wasm/
6. Addition of Garbage Collector in WebAssembly, https://developer.chrome.com/blog/wasmgc
7. WebAssembly developers guide, https://webassembly.org/getting-started/developers-guide/
8. Mozilla Developers Network WebAssembly Guide, https://developer.mozilla.org/en-US/docs/WebAssembly
9. Zaplib Post-mortem, https://zaplib.com/docs/blog_post_mortem.html
10. WebAssembly Cuts Figma Load Time by 3x, https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x
11. StackOverflow User Reporting WebAssembly Function to 300x Slower https://stackoverflow.com/questions/48173979/why-is-webassembly-function-almost-300-time-slower-than-same-js-function

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details