International Journal of Advanced Engineering and Technology
www.allengineeringjournal.com
ISSN: 2456-7655
Received: 25-07-2025, Accepted: 25-08-2025, Published: 09-09-2025
Volume 9, Issue 1, 2025, Page No. 29-33

Web Assembly: Performance gains and use cases for modern browsers

Atonuje O. E[1]., Okumoku-Evroro O[1]., Ihonvbere W[2]., Omena R. A[1].
[1] Department of Computer Science, Delta State University Abraka, Delta State, Nigeria
[2] Department of Computer Science, Ambrose Alli University Ekpoma, Nigeria

Abstract
Web Assembly (Wasm) is a binary instruction format that serves as a portable compilation target for high-level programming languages like C, C++, and Rust. Its introduction represents a paradigm shift in web development, aiming to execute code on the web at near-native speed. This paper provides a comprehensive analysis of Web Assembly, focusing on the technical foundations that enable its significant performance gains over traditional JavaScript. We explore its stack-based virtual machine architecture, linear memory model, and execution model within the modern browser's JavaScript engine. The performance benefits are quantified through comparative analysis of computational tasks, highlighting Wasm's superiority in domains requiring intensive computation, such as graphics rendering, scientific simulation, and cryptography. Furthermore, this article details the expanding ecosystem of use cases, from enabling desktop-class applications in the browser to facilitating serverless functions and even non-web embeddings. Despite its performance advantages, challenges such as the initial lack of garbage collection for managed languages and developer tooling maturity are discussed. The conclusion posits that Web Assembly is not a replacement for JavaScript but a powerful complement that is fundamentally expanding the capabilities of the web platform, paving the way for a new generation of complex, high-performance web applications.

Keywords: Web assembly, performance, web development, javascript, compiler, virtual machine, near-native speed, use cases, browser technology

Introduction

The web platform has evolved from a simple document viewer into a powerful application runtime, capable of supporting sophisticated software like video editors, 3D games, and computer-aided design (CAD) tools. This evolution has been largely driven by the relentless performance improvements of JavaScript (JS) engines. However, JavaScript's nature as a high-level, dynamically typed language with a just-in-time (JIT) compilation model presents inherent performance ceilings, particularly for compute-intensive tasks that are common in domains like game engines, scientific computing, and multimedia processing (Haas *et al*., 2017; Okumoku-Evroro 2015) [8, 17].

To break through these performance barriers and enable a new class of web applications, a consortium of major browser vendors Google, Microsoft, Mozilla, and Apple spearheaded the development of Web Assembly (Wasm). Officially announced in 2015 and becoming a World Wide Web Consortium (W3C) recommendation in 2019, Web Assembly is defined as a "low-level, portable binary compilation target" that is designed to be faster to parse and execute than JavaScript (World Wide Web Consortium, 2019; Okumoku-Evroro 2018) [18, 25].

This paper will explore the transformative impact of Web Assembly on the modern web. First, it will delve into the technical architecture of Web Assembly, explaining the design decisions that facilitate its performance advantages. Second, it will provide a detailed comparative analysis of performance between Web Assembly and JavaScript, substantiating claims with specific benchmarks. Third, it will catalog the rapidly growing array of practical use cases, demonstrating how Wasm is being leveraged in industries from gaming to finance. Finally, the paper will address current limitations and future developments, concluding that Web Assembly is a cornerstone technology for the future of high-performance web computing. Akazue *et al* (2024) [2].

Technical Foundations of Web Assembly

Understanding Web Assembly's performance gains requires an examination of its core technical design principles, which stand in stark contrast to JavaScript.

1. Binary Format and Efficient Parsing

Unlike JavaScript, which is delivered as human-readable text (source code), Web Assembly is distributed in a compact binary format (.wasm). This binary format is designed for rapid decoding and validation by the browser. Parsing a large JavaScript file is a computationally expensive process involving tokenization, syntax analysis, and scope creation. In contrast, a Web Assembly binary module can be decoded and validated in a single, fast pass, significantly reducing the time-to-execution (Zakai, 2018) [27]. This is particularly crucial for large codebases, such as game engines or complex applications, where initial load time is a critical user experience metric.

2. Stack-Based Virtual Machine

Web Assembly code is executed within a stack-based virtual machine (VM). Instructions operate by pushing and popping values onto an implicit stack, a model that is simple to compile to and efficient to execute. This design is lower-level and more predictable than JavaScript's execution model, allowing for more straightforward JIT compilation and optimization (Rossberg, 2018) [20]. Modern browsers integrate the Web Assembly VM directly into their existing, highly-optimized JavaScript engines (e.g., V8, SpiderMonkey), allowing them to leverage years of performance engineering.

3. Linear Memory Model

A key feature of Web Assembly is its linear memory model: a contiguous, resizable array of raw bytes. This is a low-level abstraction that closely resembles the memory model of native hardware (C and C++'s heap). Code compiled to Web Assembly can load from and store to this memory using simple integer byte offsets (Haas *et al*., 2017) [8]. This model provides two major performance benefits:

1. **Predictable Access:** Memory access patterns are explicit and predictable for the VM, enabling optimizations.

2. **Zero-Cost Data Sharing:** Large data structures like images, audio samples, or 3D meshes can be placed in this linear memory and efficiently shared between Web Assembly and JavaScript through an ArrayBuffer without copying, a process known as zero-copy (Tillmann *et al*., 2021) [21].

4. Strong Typing and AOT Compilation

JavaScript is dynamically typed, meaning types are resolved at runtime. This forces the JIT compiler to make assumptions and create specialized code paths, with de-optimization occurring if those assumptions are violated a significant performance cost. In contrast, Web Assembly is statically typed. Every operand and operation have a known, fixed type (e.g., i32, i64, f32, f64). This allows the browser to compile the entire Web Assembly module to highly optimized machine code ahead of time (AOT) with confidence, eliminating the need for type speculation and the risk of de-optimization (Gohman, 2019) [7].

5. Security and Sandboxing

Performance cannot come at the expense of security. Web Assembly is designed to be memory-safe and sandboxed. It cannot directly access the host system's APIs, DOM, or other resources outside its linear memory. All interaction with the outside world, including the browser's APIs, must go through JavaScript via a well-defined import/export interface. This ensures that Web Assembly code is confined to its sandbox, preventing malicious activity and upholding the web's security model (Lehmann *et al*., 2020) [11].

Performance Analysis: Web Assembly vs. JavaScript

The theoretical advantages of Web Assembly's design translate into tangible performance gains in practice. Benchmarks consistently show that for computational workloads ("number crunching"), Web Assembly outperforms JavaScript, often approaching the speed of native code.

1. Computational Benchmarks

Synthetic benchmarks like the Polyhedron CoreMark, which measures processor performance, show Web Assembly (compiled from C/C++) consistently outperforming optimized JavaScript by a factor of 1.5x to 3x (Mozilla, 2018) [13]. The performance gap is most pronounced in tasks that involve heavy integer arithmetic, memory manipulation, and predictable control flow. For example, a benchmark performing a SHA-256 cryptographic hash might run 2x faster in Web Assembly than in highly optimized JavaScript (Williams, 2019) [24]. This is because the Web Assembly code can be compiled directly to efficient machine instructions for the task, while the JavaScript JIT must first parse, interpret, and optimize the code, often with less efficient results.

2. Startup Time

For large codebases, the initial parsing and compilation time is a critical metric. As previously noted, the binary format of Web Assembly gives it a significant advantage. A study by Google's V8 team demonstrated that parsing a 10MB Web Assembly binary was over 20x faster than parsing an equivalent 10MB JavaScript file (gzipped), which directly translates to a faster startup time for the application (V8 Team, 2018) [23].

3. Memory Usage

The memory story is more nuanced. Web Assembly's linear memory can be more efficient for storing and manipulating large, homogeneous data arrays due to its lack of object overhead. However, a complex JavaScript program might use more efficient data structures for its specific task. In general, for the same algorithm, a Web Assembly implementation will often have a smaller memory footprint for the computation itself, though the overall memory usage of the web page depends on the interplay between JS and Wasm (Tillmann *et al*., 2021) [21].

4. The "JavaScript Friendliness" Factor

It is crucial to note that Web Assembly's biggest performance win is not in replacing all JavaScript, but in augmenting it. The optimal architecture for a modern web app often involves writing the performance-critical core (e.g., physics engine, image processing filter, encryption routine) in a language like Rust or C++ and compiling it to Web Assembly, while using JavaScript for the application logic, DOM manipulation, and UI updates. This hybrid model leverages the strengths of both technologies (Zakai, 2019) [28].

Expanding Use Cases

The performance characteristics of Web Assembly have unlocked a diverse and growing set of use cases that were previously impractical on the web.

1. Gaming and Interactive Media

The game industry has been an early and enthusiastic adopter. Major game engines like Unity and Unreal Engine can now compile their runtime and existing codebases to Web Assembly, allowing developers to publish high-fidelity, desktop-quality games directly to the web without plugins (Unity Technologies, 2022) [22]. Furthermore, tools like Google's Earth and complex 3D visualization software can now run smoothly in the browser, leveraging Web Assembly for geometry calculations and data processing.

2. Scientific Computing and Simulation

Researchers and engineers are using Web Assembly to port scientific computing libraries and applications to the browser. This allows for interactive simulations in fields like computational fluid dynamics, molecular modeling, and financial modeling. Users can run complex calculations locally without sending sensitive data to a remote server, enabling both privacy and low-latency interaction (Anderson & Gorzelnik, 2021) [3].

3. Audio and Video Editing

Web-based applications like Adobe Photoshop (web version), Figma, and Clipchamp are leveraging Web Assembly to perform computationally expensive tasks like

applying filters, encoding/decoding video, and rendering complex vector graphics directly in the browser, creating a seamless, desktop-like experience (Lardinois, 2021; Okofu *et al* 2024) [12, 16].

4. Serverless Computing (Web Assembly System Interface (WASI))

While initially a web technology, Web Assembly's portability and sandboxed nature make it an ideal format for serverless functions. The Web Assembly System Interface (WASI) is an emerging standard that provides a system interface for Web Assembly outside the browser, allowing it to securely access files, networks, and other system resources, Okumoku-Evroro *et al* (2025) [19]. This enables "universal binaries" that can run securely on any platform with a Wasm runtime, promising faster cold starts and greater density for serverless platforms compared to traditional containers (Butcher, 2022; Mozilla, 2020; Atonuje & Egwali 2023) [4, 5, 14].

5. Legacy Application Migration

Enterprises are using Web Assembly to port legacy desktop applications written in languages like C++ to the web. This allows them to modernize their software and make it accessible via a browser without a complete rewrite, preserving decades of investment in existing codebases (Hwang, 2020) [10].

Challenges and Future Directions

Despite its promise, Web Assembly is not without its challenges and is still evolving and has the following challenges and directions in the future:

1. Garbage Collection and Managed Languages

The initial version of Web Assembly (Wasm MVP) lacked built-in support for garbage collection (GC), making it challenging to efficiently compile languages like Java, C#, or Go, which rely on GC. This limited its initial use primarily to C, C++, and Rust. However, the recent ratification of the Wasm GC proposal is a major milestone, opening the door for efficient compilation of a much wider range of languages to Web Assembly (World Wide Web Consortium, 2023) [26].

2. Tooling and Debugging

While tooling has improved dramatically, debugging a Web Assembly module can still be more challenging than debugging JavaScript. Source maps help by mapping the binary instructions back to the original source code, but the experience is not yet as seamless as native JS debugging. The ecosystem of libraries and frameworks specifically for Web Assembly development is also less mature than that of JavaScript (Hilbert, 2022; Okofu *et al* 2025) [9, 15].

3. Web API Access

Currently, Web Assembly must call into JavaScript to access most Web APIs (e.g., DOM, WebGL). This "function call tax" can be a bottleneck for certain tasks. Proposals like "Interface Types" and "WebIDL Binding" aim to allow Web Assembly modules to call Web APIs directly, further reducing overhead and improving performance (Garneau, 2021) [6].

Conclusion

Web Assembly represents a fundamental leap forward for the web platform. By providing a secure, portable, and high-performance compilation target, it has successfully addressed the performance limitations of JavaScript for compute-heavy tasks. Its technical design centered on a binary format, stack-based VM, and linear memory enables near-native execution speed and efficient resource utilization, Akazue *et al* (2023) [1].

The use cases for this technology are vast and continually expanding, from bringing desktop-class gaming and applications to the browser to redefining serverless computing architecture with WASI. It is important to reiterate that Web Assembly's goal is not to replace JavaScript but to complement it, creating a powerful symbiotic relationship where developers can choose the right tool for each part of their application.

As the standard continues to evolve with support for garbage collection, improved tooling, and direct Web API access, Web Assembly's role will only grow more central. It is poised to become a ubiquitous runtime, not just for the web but for the broader computing landscape, truly fulfilling its promise of enabling "fast, secure, and portable computation on the web and beyond" (Haas *et al*., 2017) [8]. The future of web application development is one where the boundaries of performance are continually pushed, and Web Assembly is the key catalyst for this ongoing revolution.

References

1. Akazue M, Onovughe A, Omede E, Hampo JPAC. Use of adaptive boosting algorithm to estimate user's trust in the utilization of virtual assistant systems. International Journal of Innovative Science and Research Technology,2023:8(1):502–507.
2. Akazue M, Esiri KH, Clive A. Application of RFM model on customer segmentation in digital marketing. Nigerian Journal of Science and Environment,2024:22(1):57–67. https://doi.org/10.61448/njse221245
3. Anderson J, Gorzelnik I. Bringing scientific computing to the web with WebAssembly. Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics (WIMS 2021), 2021. https://doi.org/10.1145/3466997.3467001
4. Atonuje EO, Egwali AO. An improved automatic context summary generator system using data-mining. Nigerian Journal of Science and Environment,2023:21(2):621–637.
5. Butcher M. What is WASI (WebAssembly System Interface). The New Stack, 2022. https://thenewstack.io/what-is-wasi-webassembly-system-interface/
6. Garneau M. Standardizing WASI. A system interface to run WebAssembly outside the web. Mozilla Hacks, 2021. https://hacks.mozilla.org/2021/10/standardizing-wasi/
7. Gohman D. WebAssembly: A new kind of code for the web. Chromium Blog, 2019. https://blog.chromium.org/2019/08/webassembly-new-kind-of-code-for-web.html
8. Haas A, Rossberg A, Schuff DL, Titzer BL, Holman M, Gohman D, *et al*. Bringing the web up to speed with WebAssembly. ACM SIGPLAN Notices,2017:52(6):185–200. https://doi.org/10.1145/3140587.3062363

9. Hilbert S. Debugging WebAssembly Current state and future directions. InfoQ, 2022. https://www.infoq.com/articles/debugging-webassembly-state-future/
10. Hwang J. Modernizing legacy applications with WebAssembly. IBM Developer, 2020. https://developer.ibm.com/articles/modernizing-legacy-apps-with-webassembly/
11. Lehmann D, Kinder J, Pradel M. Everything old is new again: Binary security of WebAssembly. Proceedings of the 29th USENIX Security Symposium, 2020, 217–234. https://www.usenix.org/conference/usenixsecurity20/presentation/lehmann
12. Lardinois F. Adobe brings Photoshop to the web. TechCrunch, 2021. https://techcrunch.com/2021/10/26/adobe-brings-photoshop-to-the-web/
13. Mozilla. WebAssembly. A new runtime for the web. Mozilla Blog, 2018. https://blog.mozilla.org/blog/2018/10/18/webassembly-a-new-runtime-for-the-web/
14. Mozilla. WebAssembly and WASI. A new era for the web and beyond. Mozilla Hacks, 2020. https://hacks.mozilla.org/2020/03/webassembly-and-wasi/
15. Okofu SN, Asuai C, Okumoku-Evroro O, Akazue MI. Development of an enhanced point of sales system for retail business in developing countries. Journal of Behavioral Informatics Digital Humanities and Development Research,2025:11(4):1–24. https://www.isteams.net/behavioralinformaticsjournal dx.doi.org/10.22624/AIMS/BHI/V11N1P1
16. Okofu SN, Bisina J, Okumoku-Evroro O, Akazue MI. Cash on delivery risk mitigation CMRR model. Journal of Management Science,2024:61(9):142–155. https://journals.unizik.edu.ng/jfms
17. Okumoku-Evroro O. Application of firewall system to internet security. International Journal of Information Technology and Business Management,2015:15(1):64–71.
18. Okumoku-Evroro O. Database duplicity in Nigeria Any hope for harmonization. International Journal of Advanced Computer Technology,2018:7(4):2741–2744.
19. Okumoku-Evroro O, Atonuje EO, Esosuakpo OT. Blockchain-based internet identity management systems. International Journal of Advanced Science and Research,2025:10(2):10–17.
20. Rossberg A. WebAssembly: The journey. ACM Queue, 2018, 16(5). https://queue.acm.org/detail.cfm?id=3294030
21. Tillmann N, de Halleux P, Fahland D. Performance analysis of JavaScript versus WebAssembly for data-intensive computations. Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering SLE 2021, 2021. https://doi.org/10.1145/3486602.3486778
22. Unity Technologies. WebGL. Publishing WebGL projects. Unity Documentation, 2022. https://docs.unity3d.com/Manual/webgl-building.html
23. V8 Team. Launching WebAssembly faster. V8 Blog, 2018. https://v8.dev/blog/startup-performance
24. Williams A. Benchmarking WebAssembly against JavaScript for cryptography. Auth0 Blog, 2019. https://auth0.com/blog/benchmarking-webassembly-against-javascript-for-cryptography/
25. World Wide Web Consortium. WebAssembly core specification. W3C Recommendation, 2019. https://www.w3.org/TR/wasm-core-1/
26. World Wide Web Consortium. WebAssembly Garbage Collection. W3C Working Draft, 2023. https://www.w3.org/TR/wasm-gc-1/
27. Zakai A. Emscripting a C++ library to WebAssembly. Mozilla Hacks, 2018. https://hacks.mozilla.org/2018/01/emscripting-a-c-library-to-webassembly/
28. Zakai A. WebAssembly. What's the big deal. O'Reilly Media, 2019. https://www.oreilly.com/content/webassembly-whats-the-big-deal/
29. Bierman G, Abadi M, Torgersen M. Understanding TypeScript. ECOOP – Object-Oriented Programming, 2014, 257–281. https://doi.org/10.1007/978-3-662-44202-9_11
30. Bright P. The Web is getting its bytecode: WebAssembly. Ars Technica, 2017. https://arstechnica.com/information-technology/2017/06/the-web-is-getting-its-bytecode-webassembly/
31. Canella C, Schwarz M, Lipp M, Von Berg B, Giner L, Gruss D, *et al*. A systematic evaluation of transient execution attacks and defenses. USENIX Security Symposium, 2019.
32. Chen J, Rossberg A. A formal specification of WebAssembly's text format. Proceedings of the 19th International Conference on Generative Programming. Concepts and Experiences, 2020. https://doi.org/10.1145/3425898.3426958
33. ECMA International. ECMAScript 2023 Language Specification. ECMA-262,14th Edition,2023. https://ecma-international.org/publications-and-standards/standards/ecma-262/
34. Fischer K. How Shopify uses WebAssembly for fast, reliable web services. Shopify Engineering Blog, 2021. https://shopify.engineering/webassembly-for-fast-reliable-web-services
35. Google Developers. WebAssembly. Google Developers, 2022. https://developers.google.com/web/updates/2018/01/webassembly
36. Herrera A, Quintero A, Gil D, Zúñiga C, Ovalle D, Duque N. WebAssembly and JavaScript challenge: performance analysis in a video encryption/decryption process. IEEE Access,2020:8:152558–152569. https://doi.org/10.1109/ACCESS.2020.3017618
37. Intel Corporation. WebAssembly and Intel Architecture. A Performance Analysis. White Paper, 2020.
38. Jensen SH, Møller A, Thiemann P. Type analysis for JavaScript. International Static Analysis Symposium, 2009, 238–255.
39. Johns M. Security of WebAssembly. OWASP, 2019. https://owasp.org/www-pdf-archive/Security_of_WebAssembly.pdf
40. Kang J, Kim T. A study on the performance improvement of web applications using WebAssembly.

The Journal of Supercomputing,2021:77(9):10214–10232. https://doi.org/10.1007/s11227-021-03664-1
41. Lerner BS, Ronsen S, Croft J, Fisher E, Tobin-Hochstadt S, Krishnamurthi S. TypeScript JavaScript development at application scale. MSDN Magazine, 2013.
42. Microsoft. WebAssembly on Microsoft Edge. Microsoft Docs, 2022. https://docs.microsoft.com/en-us/microsoft-edge/web-platform/webassembly
43. Mozilla Developer Network. WebAssembly. MDN Web Docs, 2023. https://developer.mozilla.org/en-US/docs/WebAssembly
44. Nethercote N, Fitzpatrick J. Speed without the wizardry: How we made WebAssembly fast. Mozilla Hacks, 2019. https://hacks.mozilla.org/2019/11/speed-without-the-wizardry-how-we-made-webassembly-fast/
45. Neumann R. WebAssembly in Action. Manning Publications, 2020.
46. O'Sullivan C. An introduction to WebAssembly for game developers. Gamasutra, 2018. https://www.gamasutra.com/view/news/315007/Anintroduction_to_WebAssembly_for_game_developers.php
47. Reiser M, Bläser L. Accelerate JavaScript applications by cross-compiling to WebAssembly. Proceedings of the 13th ACM SIGPLAN International Symposium on Dynamic Languages, 2017. https://doi.org/10.1145/3133841.3133846
48. Rydahl E. WebAssembly on the JVM. A look at Wasmtime, GraalVM, and JWebAssembly. InfoQ, 2019. https://www.infoq.com/articles/webassembly-jvm-wasmtime-graalvm/
49. Statista. Most used programming languages among developers worldwide. Statista, 2023. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/
50. Watt C. Mechanising and verifying the WebAssembly specification. Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, 2018. https://doi.org/10.1145/3167082
51. WebAssembly.org. WebAssembly Community Group. WebAssembly.org, 2023. https://webassembly.org/
52. Wickline P. WebAssembly. A game-changer for the web. Airbnb Engineering & Data Science Blog, 2018. https://medium.com/airbnb-engineering/webassembly-a-game-changer-for-the-web-5f5e5a7e7e2
53. W3C. WebAssembly Working Group. W3C, 2023. https://www.w3.org/wasm/