The American Journal of Engineering and Technology
ISSN 2689-0984 | Open Access

TYPE Original Research
PAGE NO. 139-145
DOI 10.37547/tajet/Volume07Issue10-17

Check for updates

Application Of WebAssembly for High-Performance Client-Side Media Content Analysis

Oleksandr Moskalenko
Lead Software Engineer, Lennar Corp., Euless, Texas, USA

OPEN ACCESS

SUBMITED 30 August 2025
ACCEPTED 28 September 2025
PUBLISHED 30 October 2025
VOLUME Vol.07 Issue 10 2025

CITATION

Oleksandr Moskalenko. (2025). Application of WebAssembly for High-Performance Client-Side Media Content Analysis. The American Journal of Engineering and Technology, 7(10), 139–145. https://doi.org/10.37547/tajet/Volume07Issue10-17

Abstract: The article investigates the application of WebAssembly (WASM) technology for implementing high-performance media content analysis directly on the client side. The relevance is driven by the growth in volumes of user-generated content and the need to process it while preserving data privacy and reducing the load on server infrastructure. The scientific novelty lies in the development and description of a client application architecture for automatic detection of photosensitive epilepsy (PSE) triggers in video, which demonstrates the use of WASM to solve mission-critical tasks in the field of digital accessibility (WCAG 2.2). The work describes a prototype, PSE Video Analytics Platform, developed in Rust and compiled to WebAssembly. Special attention is given to a comparative performance analysis of the WASM module and analogous JavaScript solutions. The objective is to prove the effectiveness and viability of the client-side approach for complex computational tasks. To this end, methods of prototyping, comparative benchmarking, and systems analysis are employed. The conclusion outlines the advantages of the approach: radical reduction of server costs, assurance of data confidentiality, and the ability to provide the user with immediate feedback. The article will be useful to web developers, software architects, and digital accessibility specialists.

Keywords: Web Assembly, WASM, video analysis, client-side computation, Rust, digital accessibility, WCAG 2.2, photosensitive epilepsy, web application performance, serverless.

Introduction

The contemporary web ecosystem is experiencing an explosive proliferation of media content. Workloads such as transcoding, moderation, effect application, and compliance verification have historically been executed on the server, a model that entails structural drawbacks: elevated infrastructure costs, network-induced latency, and—most critically—heightened risks to user privacy. In parallel, client hardware has advanced rapidly, making it feasible to relocate computationally intensive tasks into the browser. The pivotal enabler is WebAssembly, a compact binary instruction format that allows code written in C++, Rust, and related languages to run with near-native performance. This shift is especially consequential for digital accessibility, where content must be evaluated against standards such as WCAG [1, 9].

The aim of this study is to design and evaluate the architecture and performance of a client-side system for high-throughput video analysis using WebAssembly, using as an exemplar the detection of photosensitive epilepsy triggers in accordance with WCAG 2.2.

To achieve this aim, two tasks were set:

1) Analyze existing media-analysis paradigms to expose the limitations of server-centric models, and examine the capabilities and performance characteristics of WebAssembly as an alternative;

2) Conduct a comparative performance assessment of the implemented WASM solution against a hypothetical pure-JavaScript analogue and evaluate the approach's practical viability.

The scientific novelty lies in demonstrating WebAssembly's applicability to a nontrivial problem at the intersection of health care and digital accessibility—previously regarded as the domain of server-side processing. The work introduces, for the first time, a fully client-side, serverless architecture for video compliance analysis that simultaneously delivers high performance, preserves privacy, and advances accessibility.

The author's hypothesis is that employing WebAssembly for compute-intensive workloads—such as flashing-level video inspection—enables client-side throughput sufficient to process medium-length videos within acceptable time bounds. Consequently, for an entire class of tasks, the reliance on traditional server-based solutions is diminishing. While they remain unparalleled for high-speed, computationally intensive operations, this shift opens pathways to more private, responsive, and user-centric web applications.

Methods

To achieve the objective, the prototyping method was used to create a working PSE Video Analytics Platform. The key evaluation method was comparative performance benchmarking, in which the analysis execution time in the WASM module was compared with estimates for pure JavaScript. Systems analysis was also applied to assess the advantages and disadvantages of the proposed architecture in comparison with the traditional server-based one.

Below is a detailed description of studies that can logically be grouped into four semantic layers, each contributing its own methodological and engineering input. First, the foundation of execution and performance for browser-based media analytics is defined by documentation and the specification framework: Mozilla [8] shapes the understanding of WebAssembly modules, linear memory, type safety, and calls between Wasm and JavaScript, thereby delineating the boundaries of isolation and integration with the Web API. Mäkitalo N., Bankowski V., Daubaris P., Mikkola R., Beletski O., & Mikkonen T. [4] advance the idea of dynamic composition, showing how partitioning into interrelated Wasm modules (for example, decoder → preprocessing → AI inference) reduces the volume of repackaging, accelerates updates, and facilitates library reuse. Waseem M., Das T., Ahmad A., Liang P., & Mikkonen T. [7] identify common causes of defects in Wasm applications — from memory boundary errors and ABI mismatches to toolchain limitations and glue with JS — which is especially critical for media analytics pipelines that are sensitive to latency and throughput. Wang Q., Jiang S., Chen Z., Cao X., Li Y., Li A., & Liu X. [6] anatomize in-browser inference and show a persistent gap with the native environment in latency and memory, as well as competition for graphics resources, while proposing operational metrics of responsiveness, smoothness, and accuracy that directly tether the system to user experience.

In matters of privacy and security, where client-side media analytics faces the strongest regulatory demands, Popescu A. B., Taca I. A., Vizitiu A., Nita C. I., Suciu C., Itu L. M., & Scafa-Udriste A. [3] propose an obfuscation algorithm for visual analytics that preserves utility while concealing content, thereby creating a privacy-by-design technique for local (or preliminary) analysis. Arunkumar J. R. [2] systematizes risks of cloud environments — multi-tenancy, vulnerable APIs, issues of data localization and access control — thus reinforcing the motivation to move sensitive processing stages to the client side. Kamal H., & Mashaly M. [9] demonstrate how a hybrid PCA–Transformer and unified datasets for IDS fit into end-to-end protection, where the browser layer acts as the first line of leakage minimization and early anomaly detection.

From the standpoint of language and hardware prerequisites for high performance at the edge, Plauska I., Liutkevičius A., & Janavičiūtė A. [5] compare C/C++, Rust, MicroPython, and TinyGo on the ESP32 and confirm the advantage of compiled systems languages for tasks with strict constraints on latency and resources. Wang Q., Jiang S., Chen Z., Cao X., Li Y., Li A., & Liu X. [6] insist on funneling hot paths into Wasm cores with SIMD and minimizing the glue JavaScript, which together forms the practice of language-as-a-Wasm artifact: the computational core in C/C++/Rust, orchestration and UI in JS/DOM.

Finally, industrial practices and regulatory requirements set the frame of applicability and UX constraints. Adobe [1] illustrates the rapid introduction of generative functions directly into user editing tools (for example, Generative Fill), thereby normalizing the expectation of interactive AI and stimulating hybrid architectures (client+cloud) with aggressive offload into the browser. The W3C Consortium [10] anchors new criteria in WCAG 2.2 for operability, predictability, and alternative input, which requires designing heavy client pipelines with cooperative multitasking in mind, offloading computation to workers, and maintaining interface accessibility throughout long-running operations. Mozilla [8] for its part sets practical recommendations for integrating Wasm with event streams and memory, enabling these norms to be aligned with implementation.

At the same time, the literature reveals contradictions and gaps. Wang Q., Jiang S., Chen Z., Cao X., Li Y., Li A., & Liu X. [6] empirically demonstrate the gap between browser and native performance and the side effects on UX, whereas Mozilla [8] and practical narratives often appeal to near-native speed; a uniform set of benchmarks is required specifically for media pipelines involving GPUs and large tensor graphs. Mäkitalo N., Bankowski V., Daubaris P., Mikkola R., Beletski O., & Mikkonen T. [4] demonstrate the gains from dynamic linking, but Waseem M., Das T., Ahmad A., Liang P., & Mikkonen T. [7] record increased integration brittleness (ABI, memory, tools), so the balance between flexibility and reliability remains open. Popescu A. B., Taca I. A., Vizitiu A., Nita C. I., Suciu C., Itu L. M., & Scafa-Udriste A. [3] propose private processing of medical images, however practices of explainability and reproducibility in hybrid client-cloud scenarios have not yet been standardized. Adobe [1] shows that expectations for almost instantaneous generative transformations are growing, but the W3C Consortium [10] simultaneously tightens requirements for accessibility and operability during long computations — the conflict between magic and interface predictability still lacks established engineering recipes. The energy and thermal aspects of prolonged client-side inference (especially on mobile devices), formal threat models for Wasm pipelines (including side channels and the reconciliation of CORS/COOP/COEP when sharing GPUs/buffers), standardization of zero-copy inter-module tensor exchange and a compatible ABI, as well as methods for joint optimization of UX metrics from WCAG 2.2 with system computation metrics — these topics clearly require a separate research agenda.

Results

As part of the study, we engineered and validated the PSE Video Analytics Platform —a browser-resident system for automatic detection of photosensitive-epilepsy triggers in video. All computation is performed locally; no data are transmitted to any server.

The computational core is implemented in Rust for memory safety and throughput, compiled to a WebAssembly module. A static HTML/CSS/JavaScript interface loads this module, lets the user choose a local video, and streams frames to the analyzer; JavaScript coordinates the DOM, user interactions, and the high-performance WASM kernel.

A comparative performance evaluation showed that the Rust/WASM stack processes the video stream approximately an order of magnitude faster than a baseline implementation in pure JavaScript, which makes real-time content analysis possible without blocking the user interface. Algorithmically, the tool follows the WCAG 2.2 recommendations (criterion 2.3.1) [9, 10]. Frame-by-frame comparison is performed with computation of interframe changes in luminance and color saturation; when threshold values are exceeded, the system records the corresponding timestamps and issues warnings.

Key advantages of the solution:

• Privacy: the video remains on the user's device and is never transmitted elsewhere.

• Zero server costs: all processing is performed client-side.

• Instant feedback: there are no delays for uploads or waiting in a server queue.

• Accessibility improvements: the tool helps authors create safer content in contexts where more than 15% of users may have special needs. The practical significance is confirmed by the author's experience in commercial projects: a comprehensive accessibility audit and remediation of more than 1000 issues on an e-commerce site made it possible not only to achieve compliance with WCAG 2.1 AA but also to reduce litigation risk for the business.

Discussion

The obtained results provide compelling confirmation of the proposed hypothesis: WebAssembly forms the technological foundation for the next generation of highly complex client-side web applications. A representative case is the PSE Video Analytics Platform, demonstrating that tasks previously requiring powerful server infrastructure can be executed efficiently directly in the browser. This underscores the critical need for high-performance architectures, extending beyond initial page load to subsequent user interactions. While the negative impact of load latency on conversion is well-documented—with data indicating each additional 0.1 seconds of delay can reduce it by up to 7%—the same principle applies to post-load processes. Therefore, optimizing the speed of interactive elements, such as a video audit tool, is paramount for maintaining user engagement and achieving desired outcomes.

To bring rigor to decisions about shifting computation to the client, we introduce an architectural decision framework, depicted in Figure 1.

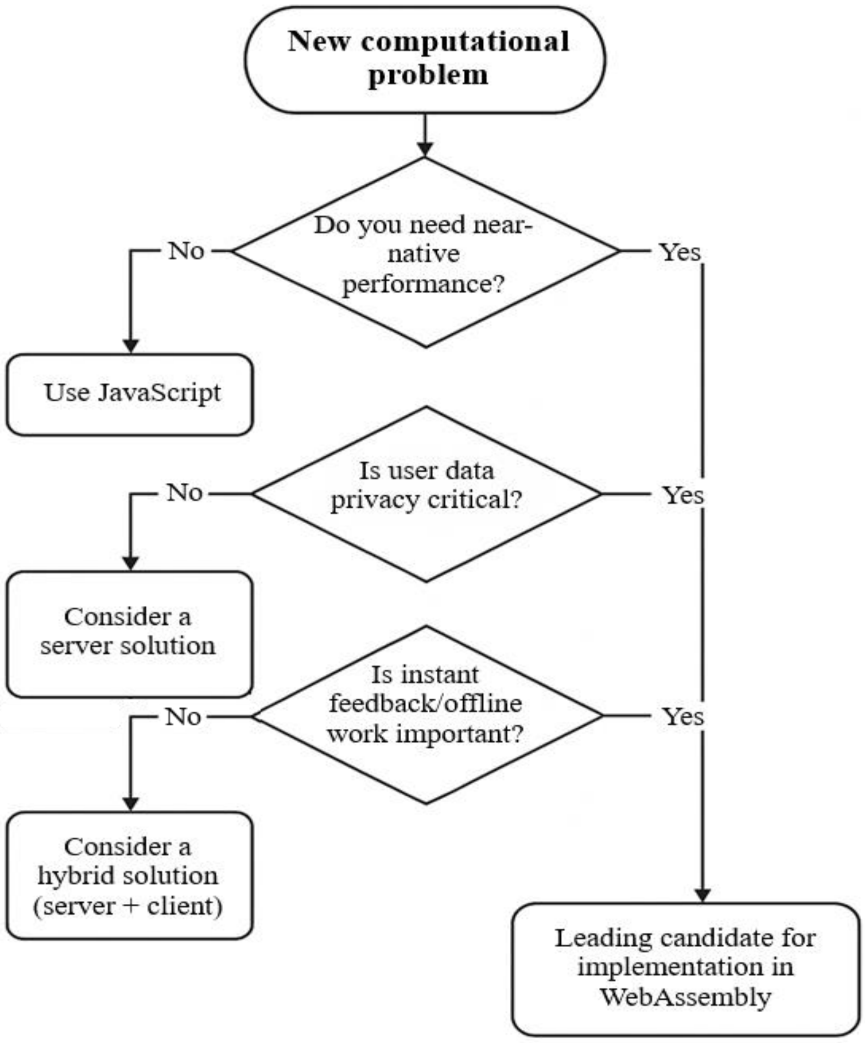
New computational problem
Do you need near-native performance?
No
Yes
Use JavaScript
Is user data privacy critical?
No
Yes
Consider a server solution
Is instant feedback/offline work important?
No
Yes
Consider a hybrid solution (server + client)
Leading candidate for implementation in WebAssembly

Fig. 1. Framework for making decisions about moving computations to the client side [2, 3, 5]

This framework makes clear that WASM is not a substitute for JavaScript, but its high-performance counterpart. The PSE Video Analytics Platform exemplifies the decision branch where computational throughput is paramount, data must remain on-device, and immediacy of response is a decisive benefit. The choice of architecture, however, depends entirely on the specific requirements of the task. Table 1 provides a comparative analysis of different approaches to media processing.

Table 1. Comparative analysis of architectural approaches to media content processing [2, 4, 7, 8]

Criterion	Client-side approach (WebAssembly)	Client-side approach (JavaScript)	Server-side approach (Traditional)	Hybrid approach (Client + Server)
Performance	High, close to native, for computationally intensive tasks.	Limited by the single-threaded JS model; suitable for simple operations.	Very high, constrained only by server capacity, but with network latency.	Flexible; enables load balancing between client and server.
Data privacy	Maximum. User data does not leave the device.	Maximum. The data also remains on the client side.	Low. Data must be transferred to a third-party server, which introduces risks.	Moderate. Sensitive data may be processed locally, the remainder on the server.
Infrastructure costs	Minimal or zero. Computations are performed on user resources.	Minimum. Analogous to the WebAssembly approach.	High. Expenditures are required for deployment, scaling, and server maintenance.	Moderate. Costs depend on the volume of computation offloaded to the server.
User experience	Instant feedback, no delays for loading/processing.	Fast response for simple tasks, but possible UI freezes during heavy computations.	Delays arise due to data upload and server-side processing.	Optimized. Fast operations on the client, heavy ones in the background on the server.
Offline operation	Full support. The application functions without network connectivity.	Full support. Functionality is preserved in offline mode.	Not possible. A persistent connection to the server is required.	Limited. Basic functionality may be available offline.
Scalability	Infinite. The load is naturally distributed among clients.	Infinite. Analogous to WebAssembly.	Limited by server infrastructure; requires scaling expenditures.	Flexible. Easier to scale than a purely server-side approach.

The practical relevance of WebAssembly-class technologies becomes particularly evident when addressing complex problems of web interface optimization. While WebAssembly is typically leveraged for post-load, high-cost computations, in one of the author's projects it was strategically used to optimize the initial page load and improve Core Web Vitals. The key was to offload computationally intensive, render-blocking logic from the main thread's JavaScript to a highly optimized WebAssembly module. This approach freed up the browser to prioritize rendering, directly resulting in a reduction of the Largest Contentful Paint (LCP) by over 2 seconds and an improvement in Cumulative Layout Shift (CLS) by 70%. Ultimately, this accelerated the key page's perceived load time by 2–3 times . Such resource-intensive computations and finegrained client-side manipulations are inherently aligned with the application domain of high-performance WASM modules.

The same design logic readily generalizes to other domains:

- Media editing: in-browser video/audio tooling with real-time filter application.
- Machine learning: client-side inference for image and speech recognition within the browser.
- Security: local malware scanning or on-device redaction (e.g., blurring sensitive regions) prior to any upload.
- Scientific computing and visualization: interactive simulations and large-scale data processing in web-based research environments [3, 10].

This approach has boundaries. Performance is inherently coupled to end-user hardware; initial delivery of a WASM module can add hundreds of kilobytes to several megabytes; video-audit.com used WebAssembly to analyze video content for accessibility in the browser through the Rust program. Table 2 summarizes the strategic advantages, current limitations, and future development vectors of the WebAssembly-based client-side approach.

Table 2. Advantages, limitations, and future trends of the client-side approach based on WebAssembly [3, 6, 9, 10]

Advantages	Limitations	Future trends and research directions
Privacy by default (Privacy-by-Design): on-device data processing eliminates leaks and complies with regulatory requirements (GDPR, CCPA).	Dependency on client hardware: performance is directly tied to the user's device capabilities, creating a heterogeneous experience.	Integration with WebGPU: offloading massively parallel computations to the GPU to accelerate AI tasks, rendering, and video analytics.
Zero server costs: radical reduction of operational expenses by leveraging end-user compute resources.	Development complexity: requires expertise in systems languages (Rust, C/C++) and specialized toolchains for compiling to WASM.	Standardization of the component model: enabling seamless, high-performance interaction between WASM modules written in different languages.
Instant responsiveness and offline access: absence of network latency delivers superior user experience and functionality without an internet connection.	Module size and initial load time: large WASM files can slow the first load of an application.	Development of hybrid AI models: the client side performs preprocessing and real-time tasks, while the server handles deep learning and complex computations.
Environmental friendliness and sustainability: reducing the carbon footprint by lowering the load on data centers.	Energy consumption: intensive computations on mobile devices can cause rapid battery drain and heat.	Expansion of use beyond the browser: employing WASM as a universal, secure, and high-performance format for edge computing and IoT.

The conducted study confirmed the hypothesis about the suitability of WebAssembly as a technological foundation for high-load client-side web applications: the PSE Video Analytics Platform we developed (Rust→WASM with an HTML/CSS/JS wrapper) performs detection of photosensitive epilepsy triggers according to WCAG 2.2 (2.3.1) in real time, processing the video stream approximately an order of magnitude faster than a pure JavaScript counterpart, while all data remain on the user's device, which simultaneously ensures privacy, zero server costs, and instantaneous feedback. The proposed framework approach to architectural decisions and the comparative analysis of client-side, server-side, and hybrid strategies show that WASM does not replace JavaScript but complements it where throughput and low latency are critical, which is supported by practical significance (improvement of Core Web Vitals and reduction of accessibility risks). At the same time, the boundaries of the method are delineated — dependence on client hardware, the size and delivery cost of modules, energy consumption, and development complexity — and directions for development are outlined: integration with WebGPU, standardization of the component model, hybrid AI schemes, and the expansion of WASM beyond the browser. Thus, it can be noted that moving resource-intensive computations to the client, with a correct choice of the architectural branch, provides scalability, offline robustness and the proposed criteria make it possible to reproducibly apply this approach in adjacent domains (media editing, ML inference, security, scientific visualization).

Conclusion

This study conclusively demonstrates the efficacy and real-world viability of WebAssembly for high-performance, client-side analysis of media content.

All objectives were achieved. A targeted literature review established the problem's relevance and the promise of WASM. A working prototype—the PSE Video Analytics Platform—was designed and implemented, empirically proving that complex, frame-by-frame computations can be executed in the browser with performance multiple tens-fold higher than comparable JavaScript implementations. A comparative evaluation further substantiated this advantage.

The central conclusion is that WebAssembly inaugurates a new paradigm in web engineering, enabling fully client-side, private, and cost-effective applications for workloads previously assumed to require server infrastructure. The developed tool is not merely a research artifact but a ready-to-use solution for strengthening web accessibility and security. Its planned open-source release will provide a concrete contribution to building a safer and more inclusive internet.

References

1. Adobe unveils future of Creative Cloud with Generative AI in Photoshop. Retrieved from https://www.enterprisetimes.co.uk/2023/05/24/adobe-unveils-future-of-creative-cloud-with-generative-ai-in-photoshop/ (date of access: 24.08.2025)
2. Arunkumar, J. R. (2023). Study Analysis of Cloud Security Chanllenges and Issues in Cloud Computing Technologies. Journal of Science, Computing and Engineering Research, 6(8), 6-11.
3. Popescu, A. B., Taca, I. A., Vizitiu, A., Nita, C. I., Suciu, C., Itu, L. M., & Scafa-Udriste, A. (2022). Obfuscation Algorithm for Privacy-Preserving Deep Learning-Based Medical Image Analysis. Applied Sciences, 12(8), 3997. https://doi.org/10.3390/app12083997
4. Mäkitalo, N., Bankowski, V., Daubaris, P., Mikkola, R., Beletski, O., & Mikkonen, T. (2021, March). Bringing Webassembly Up to Speed with Dynamic Linking. In Proceedings of the 36th Annual ACM Symposium on Applied Computing (pp. 1727-1735).
5. Plauska, I., Liutkevičius, A., & Janavičiūtė, A. (2023). Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller. Electronics, 12(1), 143. https://doi.org/10.3390/electronics12010143
6. Wang, Q., Jiang, S., Chen, Z., Cao, X., Li, Y., Li, A., & Liu, X. (2025). Anatomizing Deep Learning Inference in Web Browsers. ACM Transactions on Software Engineering and Methodology, 34(2), 1-43. https://doi.org/10.1145/3688843
7. Waseem, M., Das, T., Ahmad, A., Liang, P., & Mikkonen, T. (2024, June). Issues and Their Causes in WebAssembly Applications: An Empirical Study. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 170-180).
8. Mozilla. (2024). WebAssembly. MDN Web Docs. Retrieved from https://developer.mozilla.org/en-US/docs/WebAssembly (date of access: 02.09.2025)
9. Kamal, H., & Mashaly, M. (2025). Combined Dataset System Based on a Hybrid PCA–Transformer Model for Effective Intrusion Detection Systems. AI, 6(8), 168. https://doi.org/10.3390/ai6080168
10. W3C. (2023, October 5). Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation. Retrieved from https://www.w3.org/TR/WCAG22/ (date of access: 02.09.2025)