WEBASSEMBLY (WASM): BRAUZERDA YUQORI UNUMDORLIKKA ERISHISH VA NATIVE TEZLIKDA ISHLAYOTGAN VEB-ILOVALARNI QURISH IMKONIYATLARI

Islomov Diyorbek Zafar o'g'li
Iqtisodiyot va pedagogika universiteti o'qituvchisi
Rasulova Xulkar, Fayziyev Muxriddin, Hamidova Ma'rufat,
Samandarova Dilso'z, Bobomurodov Siroj
Iqtisodiyot va pedagogika universiteti talabalari

Annotatsiya: *Ushbu maqolada WebAssembly (WASM) texnologiyasining veb-brauzerlarda native tezlikka yaqin unumdorlikni ta'minlash mexanizmlari, JavaScript bilan taqqoslash natijalari va amaliy qo'llanilishi tadqiq etiladi. C/C++, Rust va Python tillarida yozilgan algoritmlar Emscripten va wasm-pack kompilyatorlari yordamida WASM moduliga o'girildi hamda brauzer muhitida benchmark testlari o'tkazildi. Natijalar shuni ko'rsatdiki, WASM hisoblash-intensiv vazifalarda JavaScriptdan o'rtacha 4.7 marta tez ishlaydi, C/C++ native koddan esa atigi 15–25% sekinroq. Maqolada WASM ning raqamli signallar qayta ishlash, kriptografiya, kompyuter ko'rish va o'yin dvigatellari sohasidagi amaliy imkoniyatlari ham ko'rib chiqiladi.*

Kalit so'zlar: *WebAssembly, WASM, Emscripten, wasm-pack, brauzer unumdorligi, JavaScript, Rust, bytecode, sandboxing, veb-ilova*

1. KIRISH (INTRODUCTION)

Zamonaviy veb-ilovalar tobora murakkablashib bormoqda: 3D vizualizatsiya, video montaj, sun'iy intellekt inferensiyasi, kriptografik operatsiyalar – bularning barchasi an'anaviy ravishda faqat mahalliy (native) ilovalarga xos deb hisoblanardi. JavaScript (JS) dinamik interpretatsiya qilinadigan til sifatida ushbu vazifalar uchun jiddiy unumdorlik to'sig'iga duch keladi, hatto JIT (Just-In-Time) kompilyatsiya qo'llanilganda ham.

2019-yilda W3C tomonidan rasmiy veb standarti sifatida tasdiqlangan WebAssembly (WASM) – bu muammoga tub yechim sifatida paydo bo'ldi. WASM – past darajali binary instruction format bo'lib, u turli yuqori darajali tillar (C, C++, Rust, Go, Python) dan kompilyatsiya qilinadi va brauzerda virtual mashina (VM) ichida stack-based arxitekturada ijro etiladi. Muhim jihati shundaki, WASM modullar JavaScript bilan bir xil sandboxing xavfsizlik modelida ishlaydi, shu bilan birga native tezlikka yaqin ko'rsatkich namoyish etadi.

Ushbu tadqiqot quyidagi savollarni hal qilishni maqsad qilgan: (1) WASM hisoblash-intensiv vazifalarda JavaScriptdan qanchalik tez? (2) Kompilyatsiya tili (C/C++ vs Rust vs Python) WASM unumdorligiga qanday ta'sir qiladi? (3) WASM ning qaysi amaliy sohalarda qo'llanilishi eng samarali? Maqolaning dolzarbligi shundan iboratki, O'zbekiston veb-ishlab chiquvchi hamjamiyatida WASM hali keng tarqalmagan va bu texnologiyaning imkoniyatlari ko'pchilik uchun noma'lum bo'lib qolmoqda.

2. MATERIALLAR VA METODLAR (MATERIALS & METHODS)

2.1. Test muhiti va vositalar

Barcha benchmark testlari quyidagi muhitda o'tkazildi: Windows 11 Pro (22H2), Intel Core i5-12400F (6 yadro / 12 ip, 2.5–4.4 GHz), 16 GB DDR4-3200 RAM, Google Chrome 124.0 (V8 engine), Mozilla Firefox 125.0 (SpiderMonkey engine) va Safari 17.4 (JavaScriptCore). Kompilyatsiya zanjirlari: Emscripten 3.1.50 (C/C++ → WASM), wasm-pack 0.12.1 + Rust 1.78 (Rust → WASM), Pyodide 0.25 (CPython → WASM).

2.2. Benchmark vazifalari

Vazifa 1 – Matritsa ko'paytirish: 1024×1024 o'lchamli float64 matritsalar ko'paytirish (2.1 mlrd arifmetik operatsiya). Xotira o'lchami: 8 MB.

Vazifa 2 – FFT (Fast Fourier Transform): 2^{24} = 16,777,216 nuqtali signal bo'yicha Cooley-Tukey algoritmi. Raqamli signal qayta ishlash scenariysi.

Vazifa 3 – SHA-256 hashing: 100 MB ma'lumot ustida kriptografik hash funksiyasi (kriptografiya va blockchain scenariysi).

Vazifa 4 – Image Blur (Gaussian): 4K (3840×2160) rasm ustida 15×15 Gaussian filtr qo'llash (kompyuter ko'rish va media qayta ishlash).

2.3. O'lchov metodikasi

Har bir test 50 marta takrorlandi va o'rtacha, median, standart og'ish hamda 95-persentil qiymatlari hisoblab chiqildi. performance.now() Web API yordamida mikrosekund aniqligida vaqt o'lchandi. "Isitish" (warm-up) sikllari JIT kompilyatsiyasi ta'sirini bartaraf etish uchun dastlabki 5 iteratsiya hisobga olinmadi. WASM modullar wasm-opt O3 optimizatsiya darajasida kompilyatsiya qilindi.

1-jadval. Tadqiqot parametrlari va asbob-uskunalar

Parametr	**Qiymat / Versiya**	**Izoh**
Kompilyator (C/C++)	Emscripten 3.1.50	clang-17 backend
Kompilyator (Rust)	wasm-pack 0.12.1	wasm32-unknown-unknown
Optimizatsiya	wasm-opt -O3	Binaryen toolkit
O'lchov API	performance.now()	Mikrosekund aniqlik
Takrorlash	50 iteratsiya	5 warm-up chiqarildi

3. NATIJALAR (RESULTS)

3.1. Unumdorlik taqqoslash natijalari

2-jadvalda to'rtta benchmark vazifasi bo'yicha JavaScript (Chrome V8), WASM (C/C++ Emscripten), WASM (Rust/wasm-pack) va Native C++ (tizim kompilyatori, taqqoslash uchun) natijalar keltirilgan. Barcha qiymatlar millisekundda (ms), kichik qiymat = tezroq.

2-jadval. Benchmark natijalari: ijro vaqti (ms), n=45 ta o'lchov o'rtachasi

Vazifa	**JS (V8)**	**WASM (C++)**	**WASM (Rust)**	**Native C++**
Matritsa 1024×1024	3,847	892	878	731

FFT 2^{24} nuqta	2,213	498	511	401
SHA-256 (100 MB)	1,654	324	318	267
Gaussian Blur 4K	5,128	1,043	1,067	856
O'rtacha tezlanish (JS/WASM)	**—**	**4.72×**	**4.68×**	**— (ref)**

** Native C++ natijalar taqqoslash uchun ko'rsatilgan; u brauzerda emas, OS darajasida ishlaydi*

3.2. Brauzerlar bo'yicha WASM unumdorligi

Chrome V8 brauzeri WASM ijrosida eng yuqori natijani ko'rsatdi. Firefox SpiderMonkey umumiy o'rtacha jihatdan Chrome dan 6–9% sekinroq ishladi, biroq FFT testida ikkisi deyarli teng natijalarga erishdi. Safari JavaScriptCore WASM ilovalarida Chrome dan 12–18% pastroq ko'rsatkich namoyish etdi. Shunday bo'lsa-da, barcha uch brauzerda ham WASM JS ga nisbatan 4–5 marta tezroq ishladi – bu engine implementatsiyasidan qat'i nazar WASM ning strukturaviy ustunligini tasdiqlaydi.

3.3. Modul o'lchami va yuklanish vaqti

WASM modullarining binary hajmi optimize qilinmagan holatda C/C++ manbasidan 30–45%, Rust dan esa 40–60% kattalikda hosil bo'ldi. wasm-opt O3 + brotli siqish kombinatsiyasi modul hajmini o'rtacha 68% ga kamaytirdi. Yuklanish (parse + compile) vaqti jihatidan Chrome'ning streaming compilation mexanizmi (WebAssembly.compileStreaming) modulni tarmoqdan yuklab olish bilan parallel kompilyatsiya qiladi, bu esa birinchi yuklanish kechikishini 2.3 marta qisqartiradi.

3-jadval. WASM modul o'lchami va yuklanish optimizatsiyasi

Modul	**Ham (KB)**	**wasm-opt (KB)**	**+brotli (KB)**	**Tejash (%)**
Matritsa (C++)	284	198	61	78.5%
FFT (Rust)	312	201	58	81.4%

SHA-256 (C++)	156	98	34	78.2%
Image Blur (Rust)	428	287	79	81.5%

4. MUHOKAMA (DISCUSSION)

4.1. Unumdorlik farqining sabablari

JavaScript interpretatsiya jarayonida dinamik tip tekshiruvi, garbage collection pauzalari va JIT optimizatsiyasining noaniqliklari (deoptimization) sekinlashuvga olib keladi. WASM esa oldindan statik tiplangan binary format sifatida bevosita mashina kodiga yaqin ko'rsatmalar ketma-ketligiga aylantiriladi. Linearly-addressed xotira modeli va strukturali boshqaruv oqimi (structured control flow) WASM ni prediktor-do‘stona qiladi, ya'ni protsessor branch prediction dan to'liq foydalanishi mumkin.

Native C++ dan 15–25% sekin bo'lish sababi ikki omilga bog'liq: birinchisi – WASM sandbox xavfsizlik chegaralari (memory bounds checking, indirect call validation); ikkinchisi – SIMD (Single Instruction Multiple Data) ko'rsatmalaridan cheklangan foydalanish. Ammo WASM SIMD proposal (128-bit vektorizatsiya) Chrome 91 dan boshlab qo'llab-quvvatlanmoqda va bu farqni yana kamaytiradi.

4.2. Amaliy qo'llanilish sohalari

Figma va grafik muharrirlari: Figma o'zining render dvigatelini C++ da yozib, WASM orqali brauzerda ishlaydi – natijada 60 FPS real-time vektorial rendering. Bu WASM ning real ishlab chiqarish muhitidagi eng mashhur muvaffaqiyat hisoblanadi.

AutoCAD Web: Autodesk 2023-yilda AutoCAD'ning 35 yillik C++ kod bazasini WASM ga ko'chirdi va brauzerda to'liq CAD funksionalligini ta'minladi.

Machine Learning inferensiyasi: TensorFlow.js WASM backend yordamida MobileNet modelini brauzerda JS backenddan 3.8 marta tezroq ijro etadi.

Blockchain va kriptografiya: MetaMask va boshqa Web3 ilovalari WASM da SHA-256, keccak256 va EC kriptografik operatsiyalarni JS dan 4–6 marta tezroq bajaradi.

4.3. Cheklovlar va muammolar

WASM ning hozirgi cheklovlari orasida asosiy muammo – DOM ga to'g'ridan-to'g'ri kirish yo'qligi. WASM modul DOM elementlarini o'zgartirish uchun JavaScript orqali "ko'prik" (JS bindings) orqali murojaat qilishi kerak, bu esa latency qo'shadi. Component Model (WASI) standarti bu muammoni hal qilish ustida ish olib bormoqda.

Debugging jarayoni ham qiyinroq: WASM binary formatni o'qish va xatolarni izlash an'anaviy JS da debuggingdan murakkabroq, garchi DWARF debug symbols va source map qo'llab-quvvatlash yaxshilanib bormoqda. Bundan tashqari, garbage collection til (Java, C#, Python) lardan kompilyatsiya qilingan WASM modullar hali ham o'z GC runtime larini birga olib kelishi modul hajmini oshiradi.

4.4. O'zbekiston veb-ishlab chiquvchilari uchun tavsiyalar

O'zbekiston IT bozorida WASM dan foydalanish hali boshlang'ich bosqichda. Amaliy nuqtai nazardan: ta'lim platformalari (online kod muharrirlari, in-browser IDE), fintech (kriptografik operatsiyalar), va media ilovalari (brauzerda video/audio kodek) – bularning barchasi WASM dan sezilarli foyda ko'rishi mumkin. Rust tilini o'rganish WASM ishlab chiqish uchun eng samarali yo'l hisoblanadi: Rust WASM arxitekturasiga nisbatan birinchi darajali qo'llab-quvvatlash taqdim etadi.

5. XULOSA (CONCLUSION)

Ushbu tadqiqot WebAssembly texnologiyasining veb-brauzerlarda hisoblash-intensiv vazifalarda JavaScriptdan 4.7 marta tezroq ishlashini eksperimental ravishda isbotladi. C/C++ Emscripten va Rust wasm-pack kompilyatsiya zanjirlari deyarli bir xil unumdorlikni ta'minladi, biroq Rust xotira xavfsizligi kafolatlari tufayli ishlab chiqish uchun ustunlik qiladi. Native C++ ga nisbatan 15–25% sekinlik esa sandboxing xavfsizlik modeli narxidir va bu aksariyat veb-ilovalar uchun maqbul hisoblanadi.

Modul optimizatsiyasi (wasm-opt + brotli) tomonidan binary hajm 78–81% ga kamaytirilishi tarmoq sarfini keskin qisqartiradi va past tarmoq tezligiga ega mintaqalar (shu jumladan O'zbekistonning qator tumanlari) uchun ham WASM ni amaliy qiladi. Streaming compilation mexanizmi esa birinchi yuklanish muammosini hal qiladi.

Amaliy tavsiyalar: (1) hisoblash-intensiv veb-ilovalar uchun WASM ni standart tanlov sifatida ko'rib chiqish; (2) Rust + wasm-pack + wasm-opt texnologik zanjiri bilan boshlash; (3) ta'lim muassasalarida WASM dasturlash kurslarini joriy etish; (4) O'zbekiston fintech va edtech startaplari uchun brauzer-tabiatli kriptografiya va real-time vizualizatsiya modullarini WASM da qurishni rag'batlantirish. WASM Component Model va WASI standartlari tugallangan sayin, WASM ning qo'llanilish doirasi server tarafga ham kengayadi.

ADABIYOTLAR RO'YXATI

[1] W3C WebAssembly Working Group (2019). WebAssembly Core Specification. W3C Recommendation. https://www.w3.org/TR/wasm-core-1/

[2] Haas, A. et al. (2017). Bringing the Web up to Speed with WebAssembly. Proceedings of the 38th ACM SIGPLAN PLDI, 185–200.

[3] Lehmann, D., Kinder, J., & Pradel, M. (2020). Everything Old is New Again: Binary Security of WebAssembly. USENIX Security Symposium, 217–234.

[4] Jangda, A., Powers, B., Berger, E.D., & Guha, A. (2019). Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code. USENIX ATC, 107–120.

[5] Emscripten Project (2024). Emscripten 3.1 Documentation. LLVM/Clang-based WebAssembly compiler toolchain. https://emscripten.org/docs/

[6] Mozilla Foundation (2024). WebAssembly Concepts. MDN Web Docs. https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts

[7] Rust and WebAssembly Working Group (2023). Rust and WebAssembly Book. wasm-pack documentation. https://rustwasm.github.io/docs/book/

[8] Figma Engineering (2020). Building a professional design tool on the web. Figma Blog. Engineering case study, C++ to WASM migration.

[9] Google Chrome Team (2023). WebAssembly SIMD proposal: 128-bit packed SIMD. V8 Engine Blog. https://v8.dev/features/simd

[10] Бобомуродов, Б. С., & Исломов, Д. З. (2026). ИНФОРМАЦИОННАЯ СИСТЕМА СБОРА И ВИЗУАЛИЗАЦИИ ДАННЫХ С ГЕОРАСПРЕДЕЛЁННЫХ ДАТЧИКОВ. Modern education and development, 45(2), 263-273.

[11] Бобомуродов, Б. С., & Исломов, Д. З. (2026). СИСТЕМА МОНИТОРИНГА КАЧЕСТВА ВОЗДУХА НА ОСНОВЕ РАСПРЕДЕЛЁННОЙ СЕТИ IoT-ДАТЧИКОВ AIR QUALITY MONITORING SYSTEM BASED ON A DISTRIBUTED IoT SENSOR NETWORK. Modern education and development, 45(2), 251-262.

[12] Islomov, D. Z. (2024). ARGO UML DASTURI YORDAMIDA AXBOROT TIZIMINI LOYIHALASHTIRISH USULLARI. Экономика и социум, (12-2 (127)), 367-372.