Accelerating AI Inference in the Browser with WebGPU: Benchmarks, Challenges and Prospects

Ignacy Ruszpel, Nikodem Wójcik

Faculty of Electrical Engineering Warsaw University of Technology

Introduction

Browser-based AI is growing rapidly, enabling real-time experiences while keeping user data local. Traditional solutions like WebGL and WebAssembly support basic ML tasks but fall short for deep learning.
WebGPU is a modern browser API offering direct GPU access, compute shaders and efficient parallel processing. This makes it ideal for accelerating operations like matrix multiplications and convolutions—key to AI inference.
This poster highlights:

- Advantages of WebGPU over WebGL for AI workloads
- Use of WebLLM, a WebGPU-accelerated runtime
- Optimization methods like quantization and pruning

WebGPU enables faster, more efficient AI directly in the browser, bringing advanced ML capabilities to the client side.

WebGPU overview

WebGPU is a new graphics and compute API designed to replace or complement older standards like WebGL. It is being developed by major tech companies including Apple, Google, Mozilla, and Microsoft [1]. The goal is to provide a modern, efficient GPU programming interface that closely resembles native APIs such as Metal, Vulkan and DirectX 12.
Compared to WebGL [2], WebGPU brings several important improvements:

- Compute shaders: Enable high-performance operations like matrix multiplication, essential for deep learning.
- Lower overhead: Allows more efficient rendering and compute task execution.
- Better resource management: Gives developers explicit control over GPU buffers, textures and pipeline states.

AI in the Browser

Several modern frameworks now support running AI models directly in the browser [3]:

- TensorFlow.js: Supports multiple backends including WebGL, WebAssembly (WASM) and WebGPU [5].
- ONNX Runtime Web: Loads ONNX models and supports WebAssembly, WebGL and experimental WebGPU.
- Other tools: Projects like transformers.js focus on specific tasks such as natural language processing or image generation.

However, running large models like GPT or StyleGAN in the browser presents several challenges:

- Memory constraints: Browsers limit the memory available to JavaScript and WebGPU.
- Loading and caching: Large model weights increase download time and require smart caching strategies.
- Performance bottlenecks: Browser engines and security sandboxes can slow down execution, even with GPU acceleration.

Conducted experiment

We evaluated AI inference using the WebLLM [4] runtime on a NVIDIA RTX 4060 Mobile GPU. The benchmark used a subset of the MATH dataset (first 30 questions) to balance execution time and result clarity. Accuracy was measured using the official evaluation script from the MATH benchmark repository. Performance was influenced by GPU support for lower-precision formats (FP8/FP16), which WebGPU leverages for faster computation.

TABLE I
MODEL PERFORMANCE AND ACCURACY (MATH: 30 QS)

Model	Quant.	Acc. (%)	Avg. (ms)	Gen. Tok.	Time/Tok.
Llama 3.2 1B	q0f16-MLC	56.7	15 593.7	19 325	23.43
Llama 3.2 1B	q4f16_1-MLC	40.0	12 554.0	18 746	20.00
Gemma 2 2B	q4f16_1-MLC-1k	40.0	12 098.1	9726	37.41
Gemma 2 2B	q4f32_1-MLC	33.3	12 780.1	9050	42.43
Llama 3.2 1B	q4f32_1-MLC	30.0	19 040.5	18 912	33.64
Gemma 2 2B	q4f16_1-MLC	26.7	12 238.6	9793	37.62
TinyLlama 1.1B	q4f16_1-MLC-1k	3.3	3270.3	5412	21.45
TinyLlama 1.1B	q4f16_1-MLC	3.3	3573.7	5878	21.41
TinyLlama 1.1B	q4f32_1-MLC-1k	0.0	3433.6	5482	24.82
TinyLlama 1.1B	q4f32_1-MLC	0.0	4769.9	7374	23.77

TABLE II
APPROXIMATE GPU MEMORY USAGE FOR SELECTED VARIANTS

Model	Variant	Precision	GPU Mem. (MB)
Llama 3.2 1B Instruct	q4f16_1	INT4 (FP16 KV)	879
Llama 3.2 1B Instruct	q4f32_1	INT4 (FP32 KV)	1129
Llama 3.2 1B Instruct	q0f16	FP16	2573
Llama 3.2 1B Instruct	q0f32	FP32	5106
TinyLlama 1.1B Chat	q4f16_1-1k	INT4 (FP16 KV)	675
TinyLlama 1.1B Chat	q4f16_1	INT4 (FP16 KV)	697
TinyLlama 1.1B Chat	q4f32_1-1k	INT4 (FP32 KV)	796
TinyLlama 1.1B Chat	q4f32_1	INT4 (FP32 KV)	840
Gemma 2 2B it	q4f16_1-1k	INT4 (FP16 KV)	1583
Gemma 2 2B it	q4f32_1-1k	INT4 (FP32 KV)	1885
Gemma 2 2B it	q4f16_1	INT4 (FP16 KV)	1895
Gemma 2 2B it	q4f32_1	INT4 (FP32 KV)	2509

Benchmarks conclusions

- Quantization significantly reduces memory usage, cutting it by up to 30%. This is especially useful for in-browser AI inference on devices with limited resources.
- Accuracy loss from quantization varies by model. For example, Llama 3.2 1B handles quantization well, while TinyLlama 1.1B shows notable performance drops, suggesting smaller models may struggle with compressed formats.
- Large models remain memory-heavy, even after quantization. Gemma 2 2B still uses over 2 GB of GPU memory, showing that further optimizations are needed for scalable deployment.
- KV cache format matters for memory efficiency. Switching from FP32 to FP16 caches reduces memory usage without hurting performance significantly.
- WebGPU supports low-precision inference efficiently, especially on modern GPUs like the NVIDIA RTX 4060 Mobile, which are optimized for FP16 and INT4 operations.

Source:
[1] W3C, WebGPU Specification, 2023. Available: https://www.w3.org/TR/webgpu/, [2] Khronos Group, WebGL 2.0 Specification. Available: https://www.khronos.org/registry/webgl/specs/latest/2.0/, [3] H. A. Goh, C. K. Ho and F. A. Abas, Front-end deep learning web apps development and deployment: a review, Springer Nature, 2023. Available: https://link.springer.com/article/10.1007/s10489-022-04278-6, [4] C. F. Ruan, Y. Qin, X. Zhou, R. Lai, H. Jin, Y. Dong, B. Hou, M.-S. Yu, Y. Zhai, S. Agarwal, H. Cao, S. Feng and T. Chen, WebLLM: A High-Performance In-Browser LLM Inference Engine, arXiv preprint arXiv:2412.15803, Dec. 2024. Available: https://arxiv.org/html/2412.15803v1, [5] S. Smilkov, D. Mane, M. Wattenberg and F. Viégas, TensorFlow.js: Machine Learning for the Web and Beyond, Proc. 2nd SysML Conference, Palo Alto, CA, USA, 2019. Available: https://www.academia.edu/69641131/TensorFlow_js_Machine_Learning_for_the_Web_and_Beyond