WebLLM: A High-Performance In-Browser LLM Inference Engine

Charlie F. Ruan[1], Yucheng Qin[1], Xun Zhou[1], Ruihang Lai[1], Hongyi Jin[1], Yixin Dong[1], Bohan Hou[1], Meng-Shiun Yu[1], Yiyan Zhai[1], Sudeep Agarwal[1], Hangrui Cao[1], Siyuan Feng[2], Tianqi Chen[1,3]

[1]Carnegie Mellon University, [2]Shanghai Jiao Tong University, [3]NVIDIA

Abstract

Advancements in large language models (LLMs) have unlocked remarkable capabilities. While deploying these models typically requires server-grade GPUs and cloud-based inference, the recent emergence of smaller open-source models and increasingly powerful consumer devices have made on-device deployment practical. The web browser as a platform for on-device deployment is universally accessible, provides a natural agentic environment, and conveniently abstracts out the different backends from diverse device vendors. To address this opportunity, we introduce WebLLM, an open-source JavaScript framework that enables high-performance LLM inference entirely within web browsers. WebLLM provides an OpenAI-style API for seamless integration into web applications, and leverages WebGPU for efficient local GPU acceleration and WebAssembly for performant CPU computation. With machine learning compilers MLC-LLM and Apache TVM, WebLLM leverages optimized WebGPU kernels, overcoming the absence of performant WebGPU kernel libraries. Evaluations show that WebLLM can retain up to 80% native performance on the same device, with room to further close the gap. WebLLM paves the way for universally accessible, privacy-preserving, personalized, and locally powered LLM applications in web browsers. The code is available at: https://github.com/mlc-ai/web-llm.

Keywords: LLM inference, on-device deployment, web browser, WebGPU, open-source

1 Introduction

Recent advancements in large language models (LLMs) have unlocked remarkable capabilities such as question-answering, code generation (Roziere et al. (2023)), and even reasoning (OpenAI (2024), Team (2024)). As the most capable models require server-grade GPUs, the models are typically hosted on the cloud during inference. However, open-source providers recently started releasing smaller models around 1 to 3 billion parameters that achieve competitive performance (Grattafiori et al. (2024), Abdin et al. (2024), Team et al. (2024), Hui et al. (2024)). Meanwhile, consumer devices have grown increasingly powerful: a 4-bit-quantized 3B model decodes 90 tokens per second on an Apple M3 laptop (Table 1). These trends make on-device LLM deployment both promising and practical. On-device deployment preserves privacy, enables personalization with local data, and unlocks new paradigms such as hybrid inference where cloud-based and on-device deployments co-exist (Qualcomm (2023)).

The web browser is an appealing platform for on-device deployment for three reasons. First, the browser is a natural agentic environment (Zhou et al. (2023)) for tasks like managing calendars, responding to emails, and creating documents—activities that could

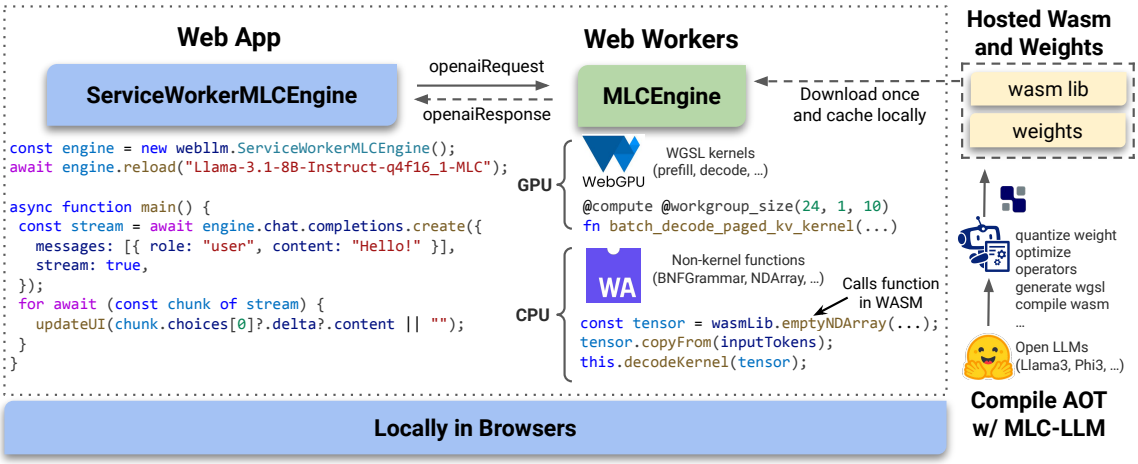

Figure 1: WebLLM System Overview.

potentially be automated by browser-based agents. Second, the web browser is universally accessible: users only need to open a URL without installing any additional software. Third, the browser abstracts away the complexities of different device backends. Although mobile devices come from various vendors, browser-based technologies such as WebGPU are backend-agnostic. For instance, instead of implementing GPU kernels for each backend (e.g. CUDA and Metal) for every new LLM operator, developers only need to provide a single implementation in WebGPU (Kenwright (2022)) (§2.3).

To address these opportunities, we introduce WebLLM, a high-performance in-browser LLM inference engine. This open-source JavaScript framework deploys LLMs locally in the browser, empowering web developers to integrate LLM capabilities into their web applications. WebLLM offers an OpenAI-style API for easy integration, leverages WebGPU and machine learning compilers (Chen et al. (2018), Team (2023)) for efficient local GPU acceleration, uses WebAssembly for performant CPU computation, and supports web workers to separate the backend execution to prevent any disruption to the UI flow. Our evaluation demonstrates that WebLLM can maintain up to 80% native performance, with the potential to further close the gap (§3).

2 System Architecture and Key Components

WebLLM is a JavaScript framework that deploys LLMs locally in the client-side browser, enabling LLM-based features in web applications. Achieving this goal poses three challenges: WebLLM needs (1) a standardized API that web applications can easily incorporate; (2) adaptation to the browser's runtime environment; and (3) efficient GPU acceleration. As shown in Figure 1, WebLLM's architecture addresses these challenges by dividing the system into three parts correspondingly: a user-facing engine `ServiceWorkerMLCEngine` with endpoint-like behavior, an encapsulated `MLCEngine` that resides in the web worker (a background thread in JavaScript), and ahead-of-time compiled efficient WebGPU kernels.

2.1 An LLM Inference Engine

Web developers instantiate a lightweight `ServiceWorkerMLCEngine` in the web application frontend and treats it like an endpoint. The engine loads an LLM when specified, takes in an OpenAI-style request at any time, and streams back the output in an OpenAI-style response, which the web application can use to update the frontend.

This familiar, endpoint-like design brings several benefits. Endpoint-like APIs are JSON-in-JSON-out and thus have well-defined behavior. Besides, OpenAI-style API is widely adopted and makes WebLLM easy to integrate into existing projects. This design also allows WebLLM to extend to advanced features with minimal changes to the API. Advanced features that WebLLM supports with this API include: structured generation with JSON Schema and context-free grammar (Dong et al. (2024)), image input with vision language models (Abdin et al. (2024), Hui et al. (2024)), and loading multiple models in the same engine for applications like retrieval-augmented generation (Lewis et al. (2020)).

2.2 Adapting to the Browser Runtime

Unlike most LLM inference engines that are either C++ or Python-based, WebLLM is implemented in JavaScript. This non-conventional LLM runtime environment requires WebLLM to adapt to the technologies offered in browsers to ensure high performance.

Web workers LLM workloads are computationally heavy and could block the UI if run on the main thread. In JavaScript, web workers are used to separate heavy computation into background threads for a smooth UI. Therefore, WebLLM leverages web workers by having two engines: a lightweight frontend engine `ServiceWorkerMLCEngine` that is exposed to the web application, and a backend engine `MLCEngine` in the worker thread that actually computes the LLM workload (Figure 1). The two engines communicate via message-passing, and the messages are simply OpenAI requests and responses.

WebGPU LLM inference requires GPU acceleration. WebGPU is a JavaScript API that allows web applications to leverage the device's GPU in the browser (Kenwright (2022)). WebGPU is also backend-agnostic: the same WebGPU kernel can run on devices with different GPU vendors, such as Apple laptops with M chips and laptops with NVIDIA GPUs. Therefore, WebLLM leverages WebGPU for any workload in LLM inference that requires GPU. We discuss in detail how such kernels are generated in §2.3.

WebAssembly Having WebGPU is not enough, as LLM inference also requires non-trivial computation on the CPU. WebAssembly (WASM) is a portable low-level bytecode that can be compiled from C++ code and run in JavaScript runtime with near-native performance (Haas et al. (2017)). Therefore, instead of re-implementing CPU workload in JavaScript, WebLLM leverages Emscripten (Zakai (2011)) to compile high-performance subsystems written in C++ into WebAssembly for various CPU workloads in LLM inference, including a grammar engine for structured generation (Dong et al. (2024)), sequence management in the paged KV cache (Team (2023)), and tensor manipulation for launching kernels (Chen et al. (2018)). This enables C++ code reuse for WebLLM without sacrificing performance.

2.3 GPU acceleration with WebGPU via MLC-LLM

GPU acceleration is crucial for high-performance LLM inference. WebGPU provides a standardized API to leverage GPU in JavaScript and abstracts out devices with different GPU vendors. However, unlike native backends such as CUDA, WebGPU does not have accelerated GPU libraries for common kernels. This makes it challenging to write high-performance customized GPU kernels such as PagedAttention and FlashAttention for WebGPU (Kwon et al. (2023), Dao et al. (2022)).

WebLLM resolves this by leveraging machine learning compilation libraries MLC-LLM and Apache TVM to compile performant WebGPU kernels. MLC-LLM takes in any open-source model's implementation in Python, which uses techniques such as the aforementioned PagedAttention and FlashAttention, and compiles the model's computation into the backend of interest (in this case, WebGPU). Besides compiling to the specified target, MLC-LLM also provides both graph-level optimizations (e.g. kernel fusion) and operator-level optimizations (e.g. GEMM tiling) to ensure the kernel performance.

MLC-LLM converts open-source models into two artifacts: converted weights and a WASM library. The WASM library contains both the WebGPU kernels and non-kernel functions in WebAssmebly. As shown in Figure 1, the models that WebLLM loads in are compiled ahead of time and hosted online.

3 Evaluation

Model	**WebLLM** (tok/s)	**MLC-LLM** (tok/s)	**Perf. Retained**
Llama-3.1-8B	41.1	57.7	71.2%
Phi-3.5-mini (3.8B)	71.1	89.3	79.6%

Table 1: Performance comparison with decoding speed between WebLLM (v0.2.75) and MLC-LLM (at commit d23d6f5). WebLLM runs on Chrome Canary 133.0.6870.0 (arm64).

We evaluate the performance of WebLLM by comparing its performance with MLC-LLM on the same Apple Macbook Pro M3 Max. The former leverages WebGPU kernels and a combination of JavaScript and WebAssembly runtime. The latter leverages Metal kernels and a combination of Python and C++ runtime. Our result shows that WebLLM preserves up to 80% of the decoding speed of MLC-LLM. There are various opportunities to further close the gap, including leveraging recent WebGPU features such as shuffling and a more careful optimization of WebLLM's runtime in general.

4 Conclusion

WebLLM demonstrates that high-performance, on-device LLM inference is feasible directly within the browser. As an open-source JavaScript framework, it empowers developers to integrate advanced LLM capabilities into web applications without sacrificing privacy or requiring server-level resources. WebLLM thus paves the way for accessible, personalized, and private LLM-driven experiences in everyday web usage.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, and Angela Fan et al. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 185–200, 2017.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Benjamin Kenwright. Introduction to the webgpu api. In *Acm siggraph 2022 courses*, pages 1–184. 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

OpenAI. Learning to reason with llms, 2024. URL `https://openai.com/index/learning-to-reason-with-llms/`.

Qualcomm. The future of ai is hybrid. Technical report, May 2023.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

MLC Team. MLC-LLM, 2023. URL `https://github.com/mlc-ai/mlc-llm`.

Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, 2024. URL `https://qwenlm.github.io/blog/qwq-32b-preview/`.

Alon Zakai. Emscripten: an llvm-to-javascript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312, 2011.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.