torch-webgpu: PyTorch compiler and WebGPU runtime

A Preprint

Jędrzej Maczan
jedrzej@maczan.pl

April 14, 2026

1 Summary

PyTorch Paszke et al. [2019] is the dominant framework for machine learning research and deployment, providing tensor operations and automatic differentiation used across virtually all modern deep learning workflows. However, PyTorch's high-performance execution has historically been tied to specific hardware ecosystems — primarily NVIDIA GPUs via CUDA and Apple Silicon via MPS — requiring separate implementations for each target platform.

WebGPU W3C GPU for the Web Working Group [2024] is a modern graphics and compute API that provides a unified abstraction over diverse GPU hardware, with implementations available across desktop, mobile, and browser environments. By targeting WebGPU, computations become portable across hardware that would otherwise require distinct backends.

torch-webgpu bridges these two ecosystems. It implements a PyTorch out-of-tree backend that compiles PyTorch models to WebGPU compute operations, enabling PyTorch tensors and models to run on any WebGPU-capable hardware. Tensor operations are implemented as WGSL GPU for the Web Working Group [2026] compute shaders, and the backend integrates with PyTorch's `torch.compile` infrastructure via the `PrivateUse1` device extension mechanism introduced in PyTorch 2.0.

2 Statement of need

High-performance machine learning research depends on PyTorch, but hardware portability remains a significant challenge. Running PyTorch models on hardware without CUDA support — including integrated GPUs, mobile devices, and non-NVIDIA discrete GPUs — currently requires either CPU fallback with substantial performance loss, or substantial engineering effort to port to a new backend.

WebGPU provides a hardware abstraction layer with broad support across GPU vendors and platforms, including browser-based execution. However, no existing tool provides a direct path from PyTorch models to WebGPU execution. Solutions such as TensorFlow.js Abadi et al. [2016] and ONNX Runtime Web operate on different model representations and require format conversion, breaking the native PyTorch workflow.

torch-webgpu fills this gap by implementing WebGPU as a native PyTorch device. Researchers can run existing PyTorch models on WebGPU hardware using the standard `torch.compile` interface and `device="webgpu"` semantics, without modifying model code or converting to intermediate formats.

3 State of the field

PyTorch 2.0 introduced `torch.compile` with TorchDynamo and TorchInductor as its default compiler stack, targeting CUDA and CPU. The `PrivateUse1` mechanism allows out-of-tree backends to register as first-class PyTorch devices. Existing out-of-tree backends include implementations for Ascend NPUs Huawei Ascend [2024] and other vendor-specific hardware, demonstrating the viability of this approach.

On the WebGPU side, TensorFlow.js provides GPU-accelerated ML in browsers using WebGL and WebGPU, but operates independently of the PyTorch ecosystem. ONNX Runtime Web supports inference from ONNX models in browsers, but again requires format conversion from PyTorch. Neither provides the native PyTorch device experience that torch-webgpu targets.

4 Software design

torch-webgpu is implemented primarily in C++ with a thin Python wrapper distributed as a PyPI package. The backend is structured around four components.

The WebGPU context (`webgpu_context.cpp`) manages adapter and device lifecycle using Dawn Google [2024], Google's native WebGPU implementation. On initialization, the context requests a high-performance adapter and falls back to any available adapter if none is found, then queries adapter limits to configure the device with maximum supported capabilities.

The WebGPU allocator (`webgpu_allocator.cpp`) implements PyTorch's allocator interface, mapping tensor memory to WebGPU buffer objects with `Storage | CopySrc | CopyDst` usage flags. Buffer sizes are aligned to 4-byte boundaries as required by WebGPU's `WriteBuffer` operation.

PyTorch operator bindings (`bindings.cpp`) register the backend under PyTorch's `PrivateUse1` device type using `TORCH_LIBRARY_IMPL`. Operators with native WebGPU implementations are registered directly; unsupported operators fall back to CPU via `at::native::cpu_fallback`.

Compute operations are implemented as WGSL compute shaders. Activation functions including ReLU, GeLU, and SiLU (`activation_functions.cpp`) are dispatched through PyTorch's `TensorIterator` infrastructure to WGSL kernels. Matrix multiplication (`mm.wgsl`) uses a 2D dispatch where each GPU thread computes one output element of C = A x B. The shader accepts a `Params` uniform buffer containing matrix dimensions, per-tensor storage offsets, and stride arrays of up to 8 dimensions, enabling correct indexing of non-contiguous and transposed tensors without requiring data copies. Workgroup size is fixed at 16x16 threads, with dispatch dimensions computed to cover the full M x K output space.

5 Research impact statement

torch-webgpu has been used to conduct empirical benchmarking of PyTorch model inference across heterogeneous hardware via WebGPU, the results of which are reported in Maczan [2026]. That study demonstrates that WebGPU-based execution enables cross-platform ML inference on hardware not supported by existing PyTorch backends.

6 AI usage disclosure

Anthropic's Claude Opus 4.6 was used to generate and optimize some WGSL shader code and to generate test scripts. All design decisions, architecture, and research were made by the author. Claude was also used to proof-read and improve the quality of this paper. All AI-assisted outputs were reviewed and validated by the author.

7 Acknowledgements

The author thanks the PyTorch team for their documentation on out-of-tree backends; Elie Michel for the WebGPU C++ guide Michel [2024]; the W3C WebGPU working group for the WGSL specification W3C GPU for the Web Working Group [2024]; the WebGPU Fundamentals project WebGPU Fundamentals Contributors [2024] for educational resources; and the Ascend PyTorch team Huawei Ascend [2024] for a high-quality reference implementation of a PrivateUse1 backend.

References

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035. Curran Associates, Inc., 2019.

W3C GPU for the Web Working Group. Webgpu specification, 2024. URL https://www.w3.org/TR/webgpu/.

GPU for the Web Working Group. Webgpu shading language, 2026. URL https://www.w3.org/TR/WGSL/.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for Large-Scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283. USENIX Association, 2016. ISBN 978-1-931971-33-1.

Huawei Ascend. Pytorch ascend: Out-of-tree privateuse1 backend for ascend npus, 2024. URL https://github.com/ascend/pytorch.

Google. Dawn: A webgpu implementation, 2024. URL https://dawn.googlesource.com/dawn.

Jędrzej Maczan. Characterizing WebGPU dispatch overhead for LLM inference across four GPU vendors, three backends, and three browsers. arXiv preprint arXiv:2604.02344, 2026. URL https://arxiv.org/abs/2604.02344.

Elie Michel. Learn webgpu: A c++ guide to webgpu for native graphics, 2024. URL https://eliemichel.github.io/LearnWebGPU/index.html.

WebGPU Fundamentals Contributors. Webgpu fundamentals, 2024. URL https://webgpufundamentals.org/.