PyTorch Made Efficient for the Edge: WASI-NN

Rishit Dagli[1, 2] Shivay Lamba[3]

[1]University of Toronto [2]Vector Institute [3]Meilisearch

PyTorch Conference

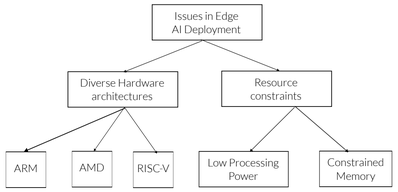
Overview

Our work develops a framework on how Pytorch models can be optimized so that they can be deployed on the edge using WASI-NN which is a standard proposal for performing machine learning inference on WebAssembly, thus making efficient and secure machine learning inference on edge devices using WebAssembly.

Motivation

Challenges in Edge AI Deployment

- Edge devices typically have just 1% compute of regular systems.
- Diverse edge device hardware necessitates unique AI model optimizations, increasing development complexity due to varying instruction sets, and memory architectures.



Enabling Edge AI

- In order to streamline On-Device AI inference: Our core motivation is to leverage the power of WebAssembly (Wasm) and the standardized interface provided by WASI-NN, to optimize Pytorch models to run on a number of edge devices while also being efficient and secure thanks to the advantages of WebAssembly.

Related Work

- One approach for running ML inference on edge device is to convert the ML model to a format that can be run on a variety of devices, such as ONNX.
  - **Conversion to ONNX Format**: Another approach involved converting PyTorch models to the ONNX (Open Neural Network Exchange) format and then executing them using an ONNX runtime or other edge AI execution environments like OpenVINO. ONNX provides a data flow graph that defines the operations. The graph is defined using protobuf which provides platform-agnostic data form thus increasing portability.
  - **PyTorch Mobile**: Provides APIs and a runtime environment to execute state-of-the-art machine learning models on mobile devices while also enabling privacy-preserving features via federated learning techniques.

  There are some limitations to these approaches though. Models with dynamic control flow can be more difficult to handle due to ONNX having a static graph representation. Another consideration is for support for multiple operators in ONNX.
- WASI-NN is a standard proposal for performing machine learning (ML) inference on WebAssembly (WASM). It allows WASM modules to call the low-level bits required for inference, abstracting the module from the underlying system. This allows the host to use any available hardware, so the same module can run ML inference on multiple systems. By using the capability of WASM, there is more efficient and secure ML deployments.
- Compared with Onnx and Edge AI execution environments, the biggest benefit of using WASI-NN is the deployment environment. As an extension, WASI-NN is not tied to any specific platform. This eliminates the need for model conversion, simplifying the deployment process. The WASI-NN PyTorch backend is PyTorch, allowing for seamless integration of PyTorch models. Contrary to PyTorch Mobile, this approach eliminates the need for model conversion, thereby preserving the original PyTorch model format for seamless deployment.

Methods

We are interested in edge scenarios where we are limited by memory and compute. We aim to address the challenge of running PyTorch models in these environments and facilitating portability by running these as Wasm modules.

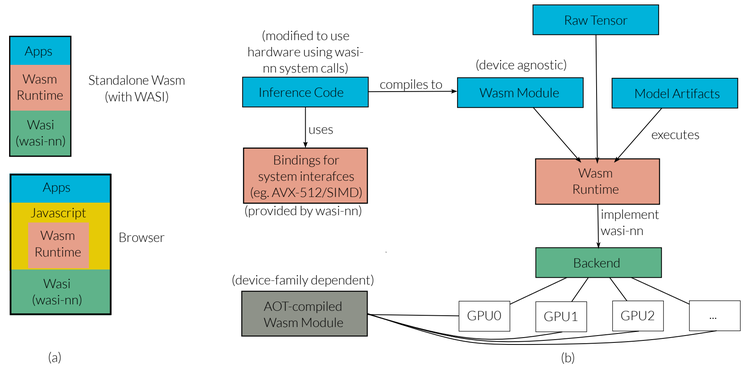


Figure 1. Overview of our proposed plug-and-play framework to run PyTorch models with Wasm on multiple targets. (a) denotes the main settings for which we propose this framework and shows how WASI-NN fits in the model runtime (b) denotes how our framework runs while inference.

- Inference code gets modified while compiling to use WASI bindings to make system calls for accessing the file system or using any hardware.
- Wasm runtimes are equipped to handle these instruction sets and are platform-specific.
- Make Wasm-specific optimizations to inference code by comparing LLVM-IR.
- A fully-portable, near-native, immensely small, and secure (due to sandboxing, but with smaller sizes than a container) Wasm module is ready to be deployed in multiple settings.

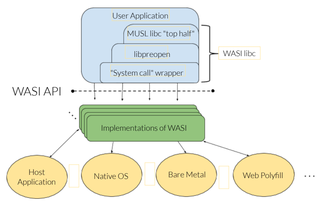


Figure 2. The architecture of how WASI handles systems calls [2].

Theoretical Improvements

Security

Each Wasm module executes within a sandboxed environment separated from the host runtime. The model executes independently, and can't escape the sandbox without going through appropriate APIs.

Full Portability

A Wasm module is just a binary format with no host-specific OS calls and it runs with limited, local, nondeterminism which sometimes might lead to unexpected behavior. Runtimes invent their own APIs.

Small Artifacts

A Wasm module is also size-efficient when contrasted with containers since each module only has the runtime code and shares the runtime with other Wasm modules. The execution environments are still sandboxed from each other and can depend on different backends.

Results Overview

We compare our approach with other edge deployment platforms or near-native approaches: PyTorch Mobile, Taichi, ONNX and OpenVino. However, our approach also comes with another set of benefits apart from the execution time.

- We see ∼40% faster volumetric rendering with MobileNeRFs over Taichi AOT
- We see ∼35% faster volumetric rendering with Instant-NGP over Taichi AOT
- We see ∼10.40% faster inference time with MobileNetV2 over Pytorch Mobile.
- We see ∼8.06% faster inference time with YoloV8 over Pytorch Mobile.

Results

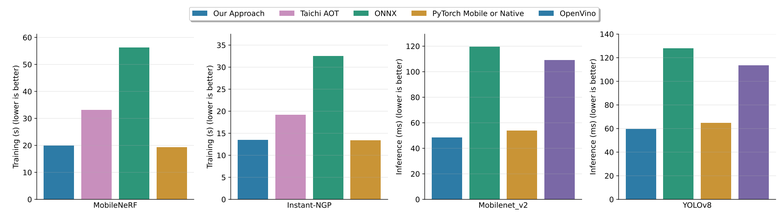


Figure 3. A quantitative comparison of how our approach relates to other edge deployment platforms, we use similarly quantized models to ensure fair comparisons. For MobileNeRF and Instant-NGP we compare our approach to Taichi AOT, ONNX, and a Native runtime. For MobileNet and Yolov8 models we compare our approach to OpenVino, ONNX, and PyTorch Mobile.

Qualitative results with Instant-NGP

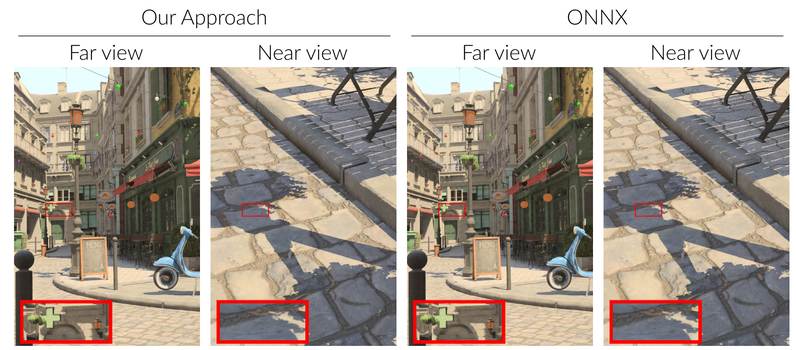


Figure 4. A qualitative comparison between our approach and using ONNX to deploy models. Our goal with this and other comparisons in this section is to set up fair comparisons between our approach and other approaches in the sense that they deploy the same model. The model performance, here Instant-NGP, does not have any impact when the model is deployed using our approach.

Acknowledgement

Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute [a].

References

[1] Wasi-NN Authors. wasi-nn. https://github.com/WebAssembly/wasi-nn.

[2] Wasmtime Authors. Wasmtime. https://github.com/bytecodealliance/wasmtime.

[a] www.vectorinstitute.ai/partnerships/current-partners/