wasi-nn: phase 3 plan

Andrew Brown

August 2024

History

- wasi-nn initially proposed in 2020
- initial implementation in Wasmtime; subsequent implementations in WasmEdge and WAMR
- interest from VMWare, Microsoft, Fastly, Cosmonic, Fermyon, Midokura, Amazon, Sonos, etc.
- collected feedback in the ML Working Group
- no major changes until:
  - preview1-ABI to preview2-ABI change in late 2023
  - discussions about LLM inferencing

Implementations

- Wasmtime
- WasmEdge
- WAMR
  - no component model support yet, could target `wasm32-wasip2-module`

Phase 2 Completed

- Entry requirements:
  - ✅ The portability criteria are documented in the proposal.
  - ✅ Precise and complete overview document is available in a proposal repo around which a reasonably high level of consensus exists.
  - ✅ A WIT description of the API exists.
  - ✅ All dependencies of the WIT description must have reached phase 2.
- During this phase:
  - ✅ One or more implementations proceed on prototyping the API.
  - ⏳ A plan is developed for how the portability criteria will be met.

The Bytecode Alliance ML Working Group has been meeting since October 2023

Intent

- wasi-nn intends to ask for phase 3 acceptance
  - when: October, November?
- phase 3 entry requirements:
  - ✅ all dependencies of the WIT descriptions must have reached phase 3
  - ⏳ portability criteria must be met (or present a plan)

- previous portability concerns:
  - *tricky test suite*: testing internal model logic feels out of scope—too broad
  - *opaque model blobs*: wasi-nn currently has seven graph encodings (`openvino`, `onnx`, `tensorflow`, `ggml`, etc.); likely more to come?

Plan

1. Implement a test suite based on an “echo” model
2. Replace `load` with `load-by-name`
3. Resolve last few changes (`prompt` interface, `error` shape, etc.)
4. Update implementations (“mostly complete”) in 2 engines

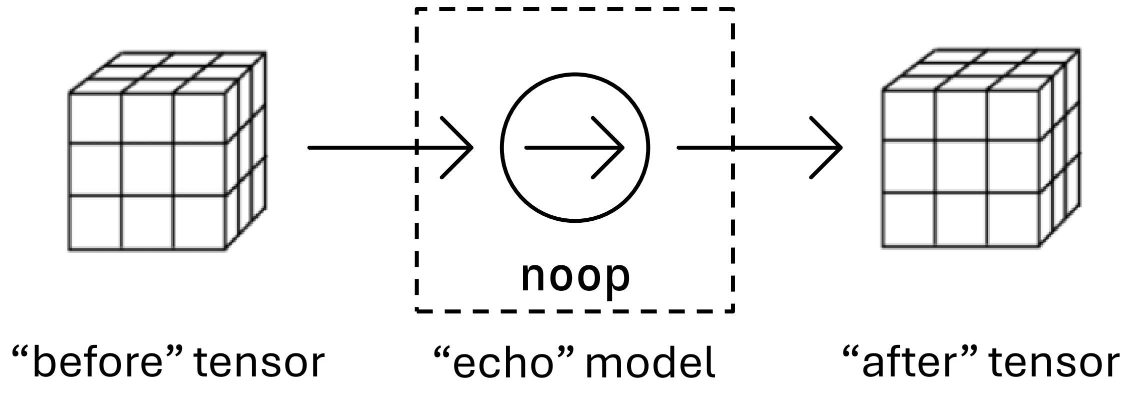
$\downarrow$

Several WIT snapshots implemented in Wasmtime

- Vote on phase 3 in Q4 of 2024

Plan: Test Suite

- wasi-nn testing is fraught:
  - small floating-point differences between ML frameworks, parameters
  - randomness is inherent in some ML models
  - many frameworks, many models, many values = difficult to check
- Conclusion: **test the wasi-nn API, not the ML model logic** (frameworks already test this!)
- How? An “echo” model…



```
let before = Tensor::new(...)?;
let graph = nn::load("echo")?;
let context = inference::init(graph)?;
match context.compute([before])? {
  [after] => assert_eq!(before, after),
}
```

Plan: `load-by-name`

- A previous concern: “load by bytes”
  - graph encoding list never done
  - encoded bytes are opaque
  - retrieving bytes is too slow
  - models are too big!
- Solution: **replace `load` with `load-by-name`**
- ĵ KĂķvỷĂ śźĝº źźśvůŕ8Ă¾-\

Before:

```
load: func(builder: list<list<u8>>,
  encoding: graph-encoding,
  target: execution-target) ->
    result<graph, error>;
load-by-name: func(name: string) ->
    result<graph, error>;
```

After:

```
load: func(name: string) ->
    result<graph, error>;
```

Plan: resolve last changes

- See wasi-nn PRs:
  - error shape: **`resource`** vs **`record`** vs **`string`**Ă6Ĵ Kź7
  - tensor residence (#69, #70)
  - coalesce tensor parameters (#77)
  - add graph parameters (#80)
  - add prompt interface (#79)

Coalesced **`compute`** signature:

```
compute: func(inputs: list<named-tensor>) ->
  result<list<named-tensor>, error>;
```

New **`prompt`** interface:

```
interface prompt {
  use graph.{graph};

  init: func(graph: graph) ->
    result<context, string>;

  resource context {
    compute: func(prompt: string) ->
      result<string, string>;
  }
}
```

Summary

- Let’s move wasi-nn to phase 3!
- A year’s worth of design feedback—a little more needed
- Help appreciated on test suite, implementations

Questions