Introduction to the RustNN Ecosystem

Markus Tavenrath, Principal Engineer Developer Technology

Web Engines Hackfest | 06/2026

RustNN

RustNN is a native implementation of the W3C WebNN standard.

It brings WebNN beyond the browser, enabling a unified web and desktop ecosystem.

To understand RustNN, we first need to understand WebNN.

NVIDIA

What is WebNN

WebNN: One Graph → Any Hardware

- WebNN is a W3C standard under active development for neural network inference
  - Provides an API to describe the computational graph through a MLGraphBuilder interface
- The Browser compiles and dispatches it to the best hardware
  - CPU, GPU, NPU — same graph*
  - Chromium delegates to Windows ML · CoreML · LiteRT
- WebNN reached origin trials phase
  - Wider testing, but still under active development
- One graph builder. The platform handles the rest.
  - But it's browser-only.

executing-an-MLGraph-using-MLTensors.js

```
const descriptor = {dataType: 'float32', shape: [2, 2]};
const context = await navigator.ml.createContext();
const builder = new MLGraphBuilder(context);

// 1. Create a computational graph 'C = 0.2 * A + B'.
const constant = builder.constant(descriptor, new Float32Array(4).fill(0.2));
const A = builder.input('A', descriptor);
const B = builder.input('B', descriptor);
const C = builder.add(builder.mul(A, constant), B);

// 2. Compile the graph.
const graph = await builder.build({'C': C});

// 3. Create reusable input and output tensors.
const [inputTensorA, inputTensorB, outputTensorC] =
    await Promise.all([
      context.createTensor({
        dataType: A.dataType, shape: A.shape, writable: true
      }),
      context.createTensor({
        dataType: B.dataType, shape: B.shape, writable: true
      }),
      context.createTensor({
        dataType: C.dataType, shape: C.shape, readable: true
      })
    ]);

// 4. Initialize the inputs.
context.writeTensor(inputTensorA, new Float32Array(4).fill(1.0));
context.writeTensor(inputTensorB, new Float32Array(4).fill(0.8));

// 5. Execute the graph.
const inputs = {
  'A': inputTensorA,
  'B': inputTensorB
};
const outputs = {
  'C': outputTensorC
};
context.dispatch(graph, inputs, outputs);

// 6. Read back the computed result.
const result = await context.readTensor(outputTensorC);
console.log('Output value:', new Float32Array(result));  // [1, 1, 1, 1]
```

Source: https://webnn.io/en/learn/get-started/quickstart

web machinelearning

Making Machine Learning a first-class web citizen

https://webmachinelearning.github.io/

* Some data types and operators are optional per the spec

NVIDIA

Web Machine Learning Comparison

web neural network

WebGPU

WA

GEMM graph node

Shader using Subgroup matrices

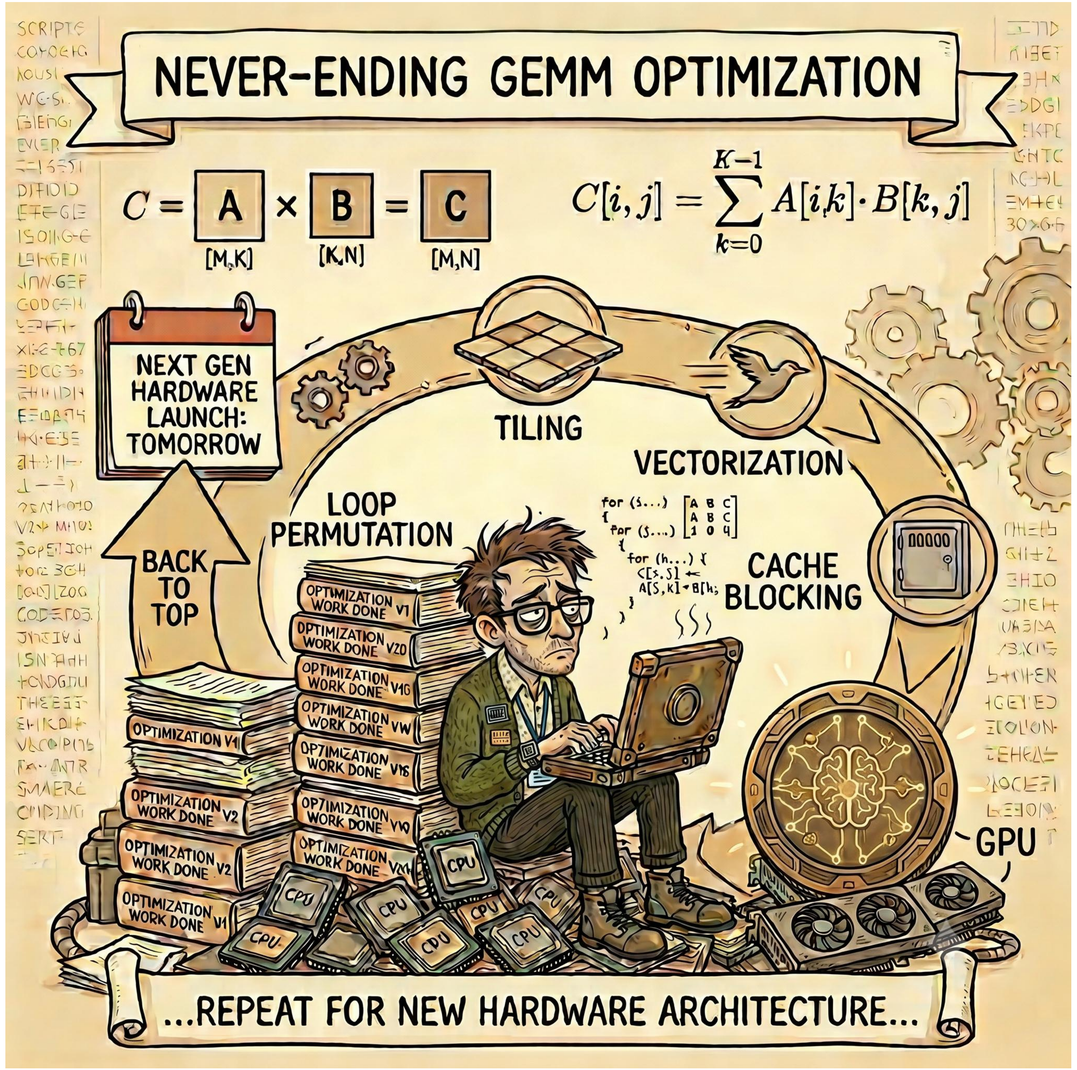
Function using 128-bit packed SIMD Extension

GEMM

Which tensor operation to execute

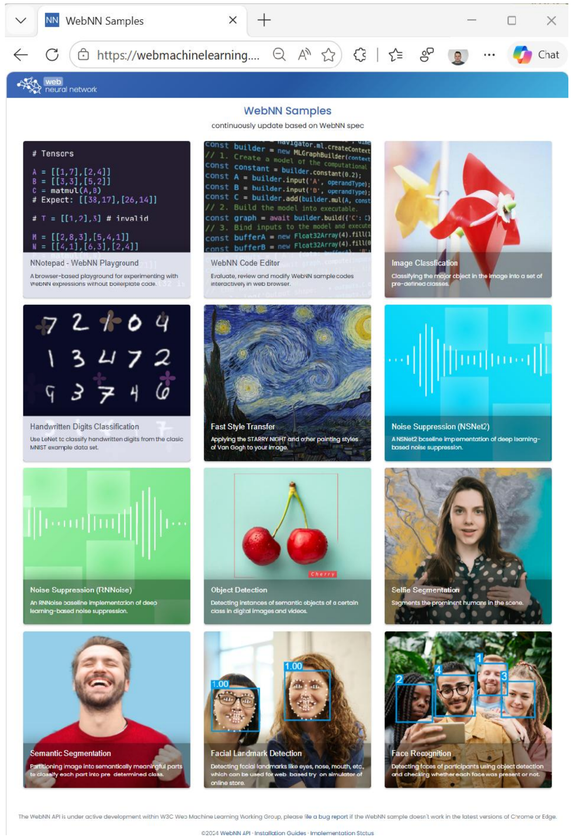
How to execute tensor operation in great detail

NPU Support?



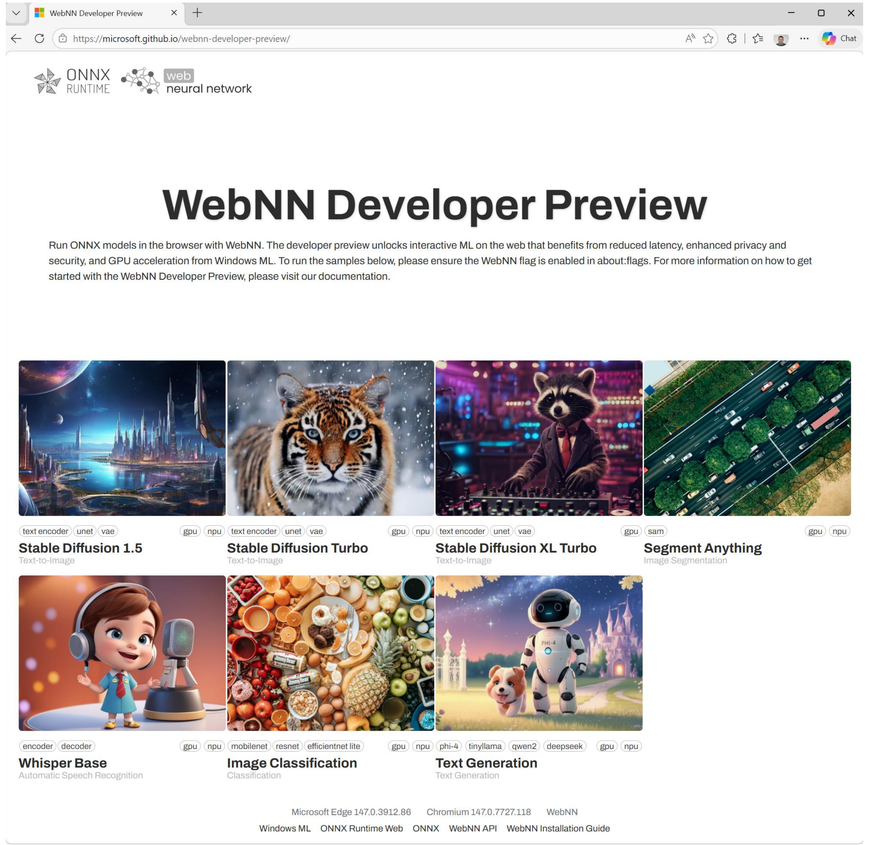
NVIDIA

WebNN Samples – W3C Web Machine Learning Working Group



- Official W3C Working Group samples, focus on data processing
- Introduction samples like matmul
- Image Vision samples
  - OCR
  - Face recognition
  - Facial landmark detection
  - Image Segmentation
  - Style Transfer
- Noise Suppression

WebNN Samples – Developer Preview



- Microsoft's WebNN Developer Preview samples
- More focused on generative AI
  - Speech Recognition
  - Text Generation
  - Image Segmentation
  - Image Generation

WebNN Killer App

Videoconferencing

- (AI) features are being integrated into videoconferencing software
- Background processing
- Noise reduction
- Text-To-Speech / Speech-To-Text
- AI Assistant running in the background
- Eye-Camera adjustments
- Don't let AI drain your battery
- Let the browser run networks on the most (power) efficient device

NVIDIA

Browser as only platform?

If WebNN works great in the browser, can it work well on the desktop as well?

If it works well, can we build a shared ecosystem?

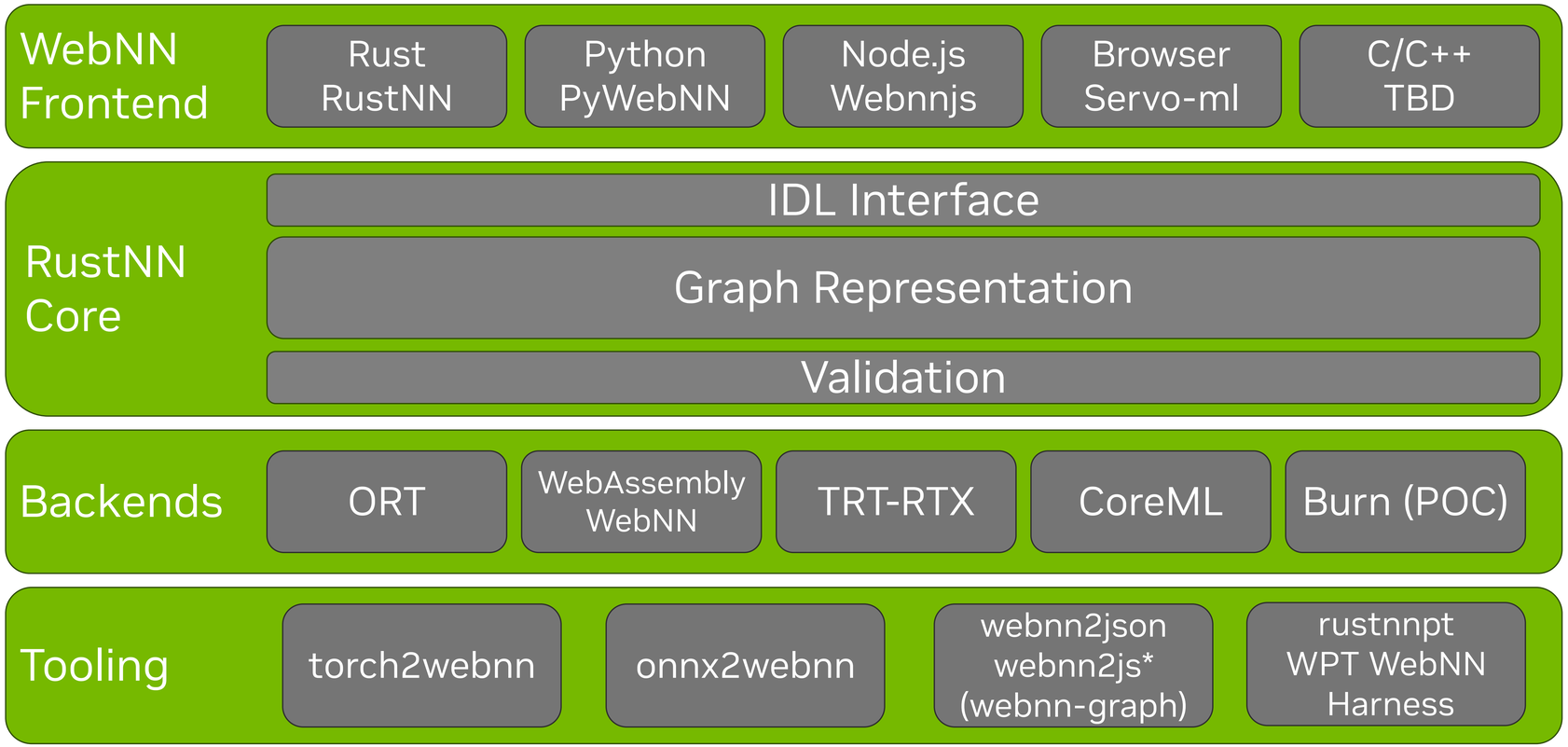
NVIDIA

RustNN – A brief history

- Chromium has a pretty good WebNN implementation on top of ONNX Runtime & LiteRT
  - Unlike WebGPU (Dawn) their implementation is deeply intertwined with Chromium
- What about other browsers or native applications?
  - Tarek Ziadé (Firefox) started rustnn, a Python and Rust Implementation of W3C WebNN aimed at Firefox in December 2025
- I joined in January 2026 after realizing the potential
  - A native implementation would allow for a single deployment path in the web and desktop
- A full ecosystem flow is required to meet the ML developers where they live today: Python

NVIDIA

RustNN Ecosystem



NVIDIA

WebNN everywhere

JavaScript – Rust – Python – and more

JavaScript (Browser, Node.js)

```
const context =
  await navigator.ml.createContext(
    {powerPreference: 'default'});
const builder =
  new MLGraphBuilder(context);
const a = builder.input('a',
  {dataType: 'float32',
   dimensions: [3, 4]});
const b = builder.input('b',
  {dataType: 'float32',
   dimensions: [4, 3]});
const c = builder.matmul(a, b);
const graph =
  await builder.build({c});
```

Rust

```
let mut ctx = MLContext::create(
  &MLContextOptions::new(
  MLPowerPreference::Default, true))?;
let mut builder =
  MLGraphBuilder::new(&mut ctx)?;
let a = builder.input("a",
  &MLOperandDescriptor::new(
    MLOperandDataType::Float32,
    vec![3, 4]))?;
let b = builder.input("b",
  &MLOperandDescriptor::new(
    MLOperandDataType::Float32,
    vec![4, 3]))?;
let c = builder.matmul(a, b)?;
let graph = builder.build(
  &HashMap::from([("c", c)]))?;
```

Python

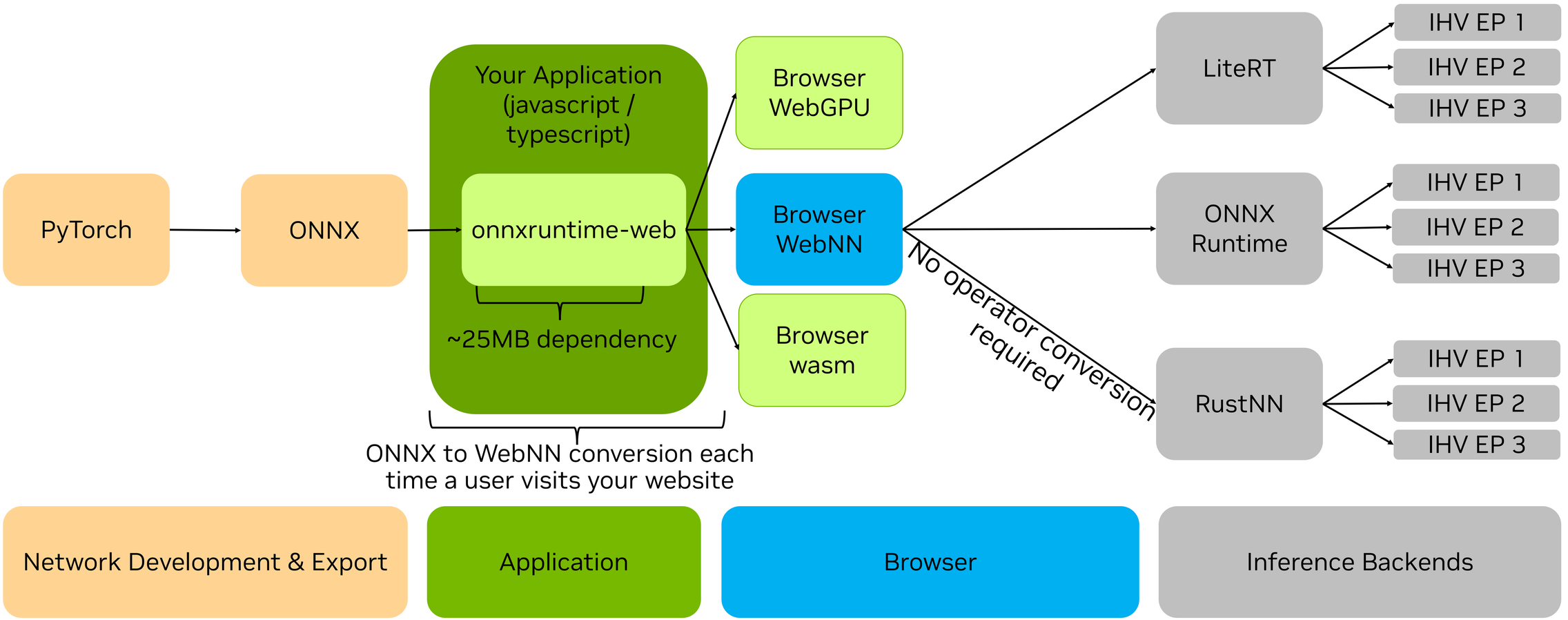
```
ml = webnn.ML()
context = ml.create_context(
  power_preference="default",
  accelerated=True)
builder = context.create_graph_builder()
a = builder.input("a", [3, 4], "float32")
b = builder.input("b", [4, 3], "float32")
c = builder.matmul(a, b)
graph = builder.build({"c": c})
```

WebNN-Graph – a simple serialization of the WebNN graph usable in JavaScript, Rust, and Python

NVIDIA

Ecosystem / Workflow for WebNN Network Deployment 1

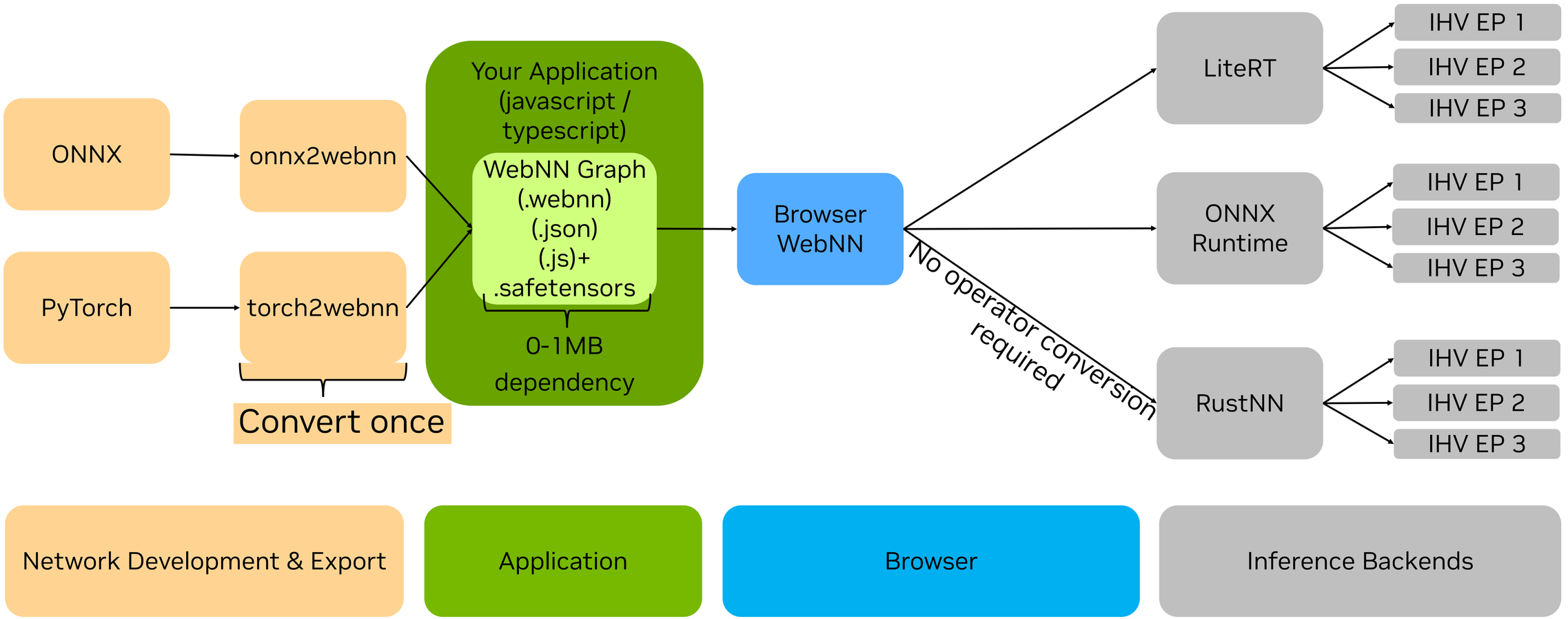
State of the Art



NVIDIA

Ecosystem / Workflow for WebNN Network Deployment 2

Optimized Conversion



NVIDIA

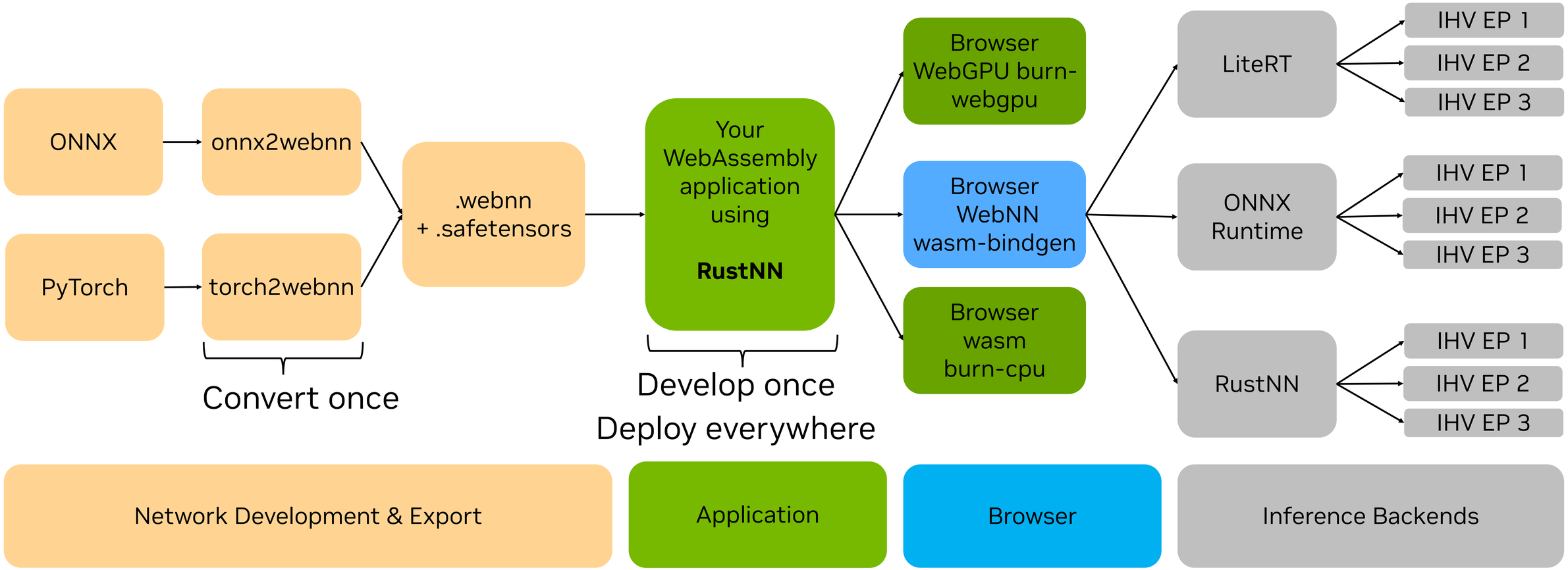
Ecosystem / Workflow for WebNN Applications 1

- Running the network is only half of the game
- AI workloads require pre and postprocessing like
  - Image decoding
  - Color space conversions
  - Normalization
  - Data Type Conversion
  - Tokenization
  - Sampling
  - Decoding
  - …
- You might have code for this in Rust already – so why not use it?

NVIDIA

Ecosystem / Workflow for WebNN Applications 2

Develop once – ship native and as web application



NVIDIA

WebNN Anywhere

Matt Hargett ported RustNN to the Apple Watch

https://github.com/rustnn/rustnn/issues/110

We've spent ~30 days running rustnn end-to-end on **Apple Watch SE 2 (arm64_32 ILP32, watchOS 11.6)** — both the most-restricted Apple runtime in current shipping hardware and a clean reproducer for every `.mlmodelc` -loader gap on the watchOS family. The

NVIDIA

RustNN Conformance

RustNN Core (ONNX Runtime Backend)

- ONNX Runtime backend will allow running across all ORT-supported execution providers
  - Execution provider selection not yet integrated
- WPT tests validate spec conformance, numerical precision, and operator coverage
- 99.36% of identified operator tests pass via rustnnpt, our Rust -> JavaScript WPT test harness
  - Regex reliably identifies operator tests, but misses more general tests
- Full test suite requires a real web environment.
  - webnnjs is a Rust/napi-rs Node.js addon that exposes the WebNN API backed by RustNN
  - Next step is to plumb it into the cargo test pipeline

NVIDIA

What’s next?

- Conformance
  - Ensure we can run 100% of the conformance tests for better test coverage
- Browser integration to get more content for testing and benchmarks
  - Update Firefox integration prototype
- Generic GPU Backend https://github.com/rustnn/rustnn/pull/135
  - WebGPU has experimental support for subgroup matrices
  - Supported by Vulkan, Metal and D3D12 through DX LinAlg Matrix Preview
  - Experiment with Burn WebGPU and ORT-WebGPU! Burn is more flexible supporting (CPU, Vulkan, DX, CUDA, Metal, …)
- Content!
  - Export more models through torch2webnn and onnx2webnn
- Harden the exporters and backends

NVIDIA

Questions?

Get engaged!

https://github.com/rustnn

Markus Tavenrath, Principal Engineer Developer Technology

NVIDIA