

CLOUD NATIVE AI +

INFERENCE DAY

NORTH AMERICA



GPU Portability Was Broken

But Now There's WebGPU

Bailey Hayes - CTO, Cosmonic

Mendy Berger - Software Engineer, Cosmonic

Presenters

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA



Bailey Hayes

CTO & Co-founder, Cosmonic
WASI co-chair
Bytecode Alliance TSC and Board

github.com/ricochet

linkedin.com/in/baileyhayes/



Mendy Berger

Software Engineer, Cosmonic
wasi-webgpu champion

github.com/MendyBerger

linkedin.com/in/mendyberger

Quick show of hands

✋ Who here is running AI models?

✋ ...and running Kubernetes with GPUs?

✋ ...and the same AI workload runs on your laptop and in your cluster?

An entire universe of models

Vision | Translation | Embeddings | Code

Speech | **LLMs** | OCR | Time series

Small fine-tuned models | Classifiers | Image generation

1M+ models on Hugging Face

🧑‍💻 🧑‍💻 🧑‍💻 Your developers want to try a lot of them, quickly.

…that have to run in two places

Developer laptops

- macOS · Apple GPU (Metal)
- Windows · NVIDIA, AMD, Intel
- Linux · NVIDIA, AMD

In the way

CUDA · ROCm · Metal

An image per GPU vendor

Operators configured per cloud

Your cluster

- Data center · NVIDIA nodes
- Edge · smaller GPUs
- Node pools · mixed hardware

One artifact, both sides

Developer laptops
Apple · Windows · Linux

✓ runs here

Wasm component

Your cluster
NVIDIA · edge

✓ runs here

💻 Try the whole universe of models on the laptop you already have

🚢 Ship the exact same artifact to NVIDIA in your cluster

⚡ Hold the GPU only while serving: serverless AI

What is Wasm?

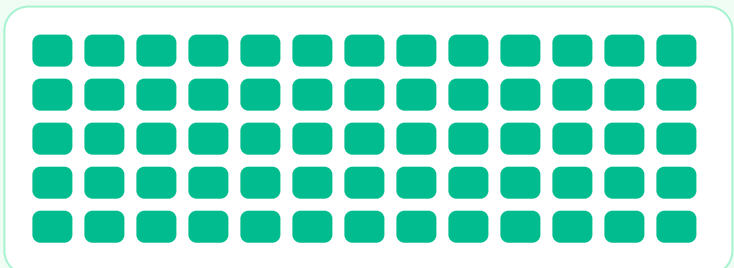
- **Compile once, run anywhere**: any CPU, any OS
- **Deny by default**: it gets nothing you don't explicitly hand it
- **Capabilities** are declared, never assumed

Containers

86MB-1.3GB image size

Idle infra cold starts

Wasm

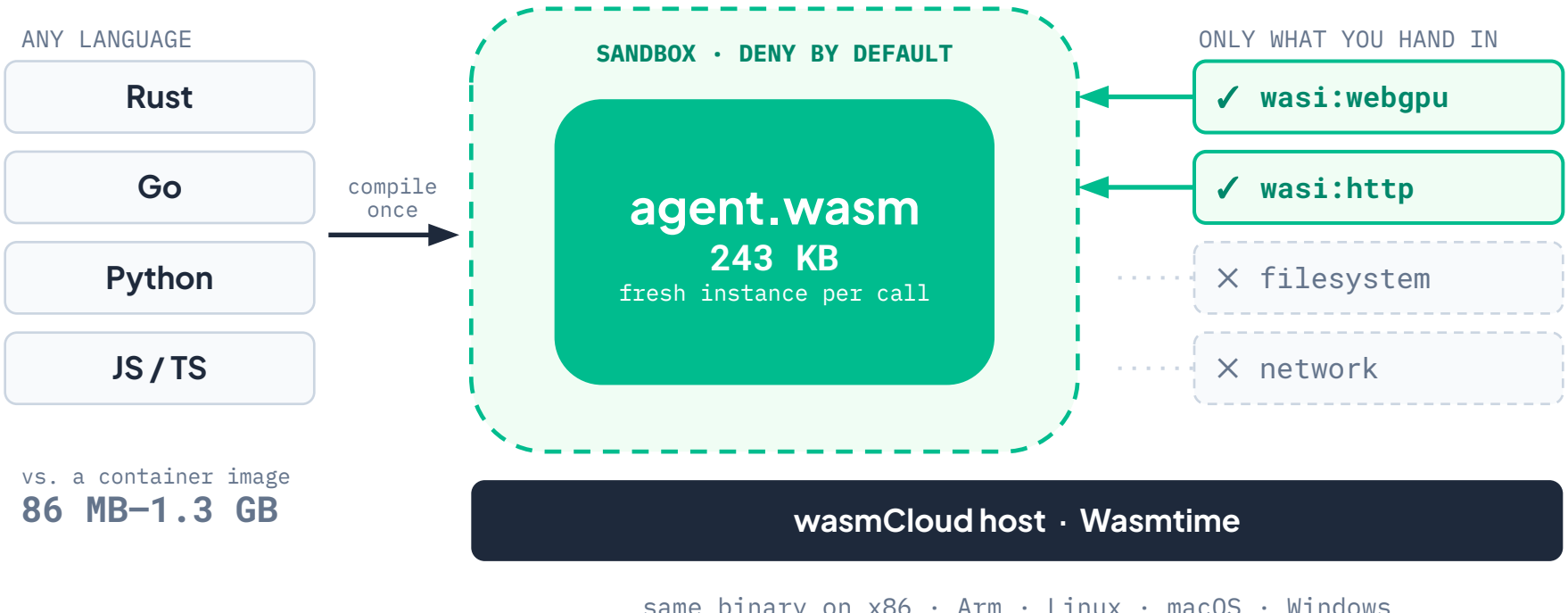


243 KB image size

Scale to zero fresh instance per call

What is Wasm?

Compile once, run anywhere. Deny by default. Capabilities are declared, never assumed.



Is it fast?

The sandbox

−4 %

generation speed as a Wasm component, against the same WebGPU build run natively. Prompt processing: within noise.

Qwen2.5 7B on an NVIDIA L4 · arewefastyet.wasmcloud.com/gpu

WebGPU against CUDA

60 %

of CUDA's generation speed today: 32 vs 54 tokens/s. Prompt processing is further behind; more on that later.

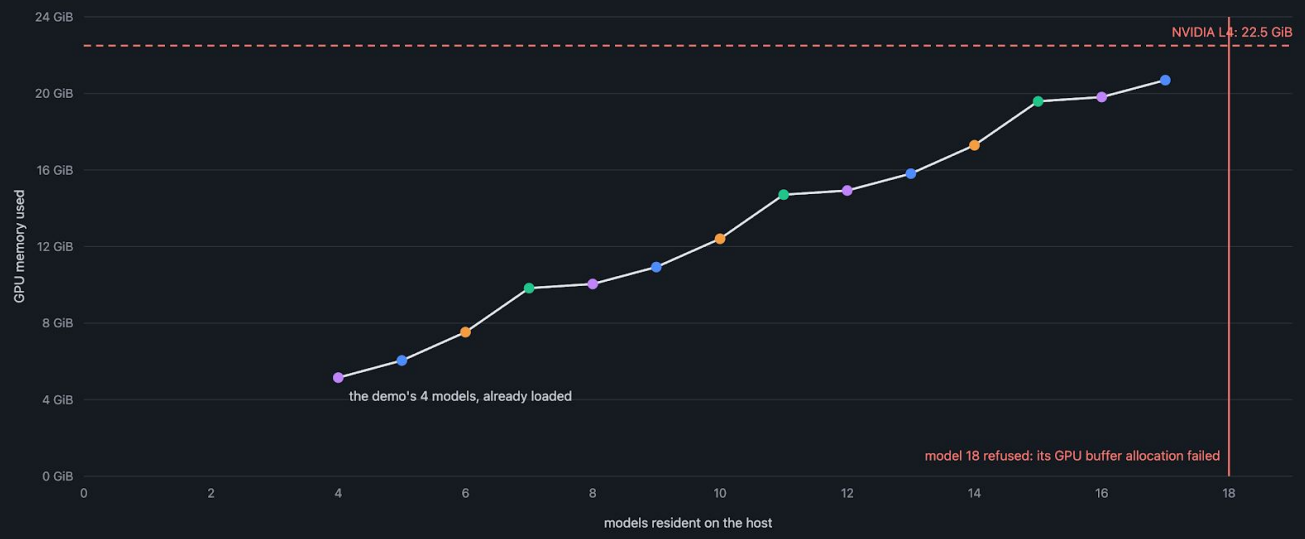
Same model and GPU, on llama.cpp's CUDA backend

Why WebGPU? Density on one GPU

How many models fit on one GPU host?

One Control host on an NVIDIA L4. Each dot adds a model workload and loads its model through wasi:webgpu. The ramp stops when a model can't load.

17	1	20.7 GiB	17/17
models resident on one host	control-host process, 1 GPU	of 22.5 GiB GPU memory	answered at once, GPU peak 100%



Model 18 (density-x18, Llama 3.2 1B) faulted alone; the 17 resident models kept serving.

What is sandboxing?

Containers start open

- See their whole filesystem image
- Can open any network connection
- Locked down afterwards: seccomp, network policies, eBPF, or a VM like Kata Containers

Components start closed

- No files, no network, no environment variables
- Each capability is granted, one by one
- Works alongside Cilium and network policies, without depending on them

The GPU is the hardest thing to sandbox. Browsers already do it.

What is WebGPU

WebGPU: is to the GPU what Wasm is to the CPU
Sandboxed, cross-platform, close to the metal

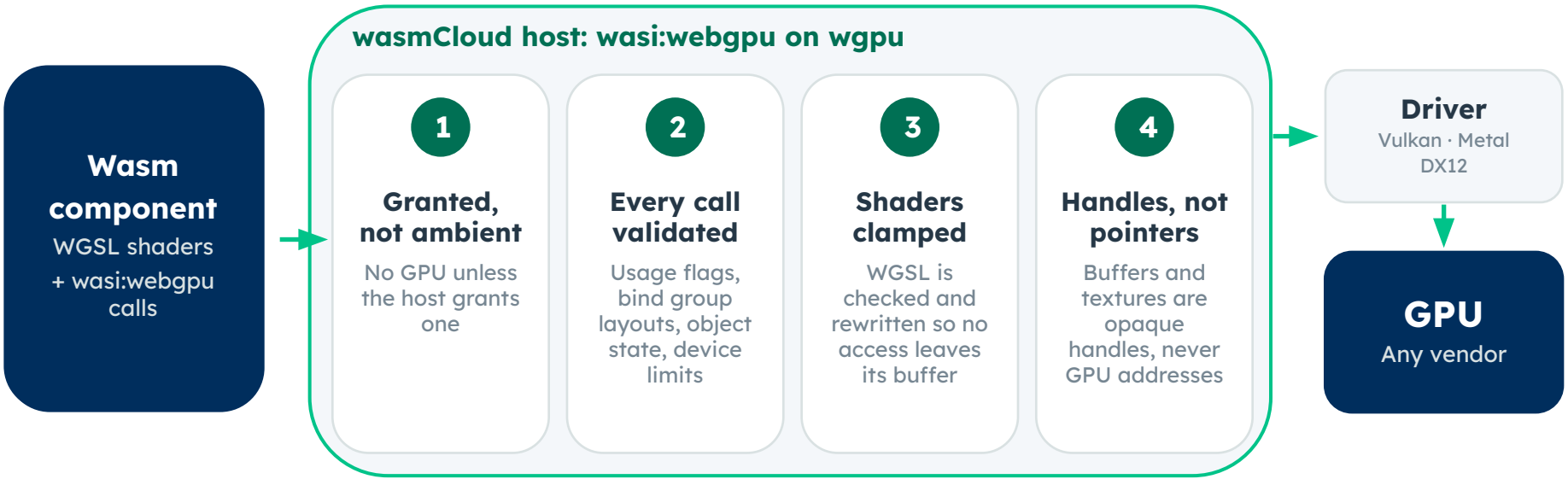
`wasi:webgpu`: brings WebGPU to WASI
One sandboxed app, both CPU and GPU

wgpu: the Rust implementation underneath
Battle-tested by Firefox · Servo · Bevy · Burn · COSMIC desktop

Firefox servo BEVY Burn COSMIC

The WebGPU sandbox

Every GPU call from a component crosses four checks in the host before it reaches the driver



5 **Zero-initialized memory**
A workload never sees what the last one left behind

6 **Resource limits**
Caps on what any one workload can allocate

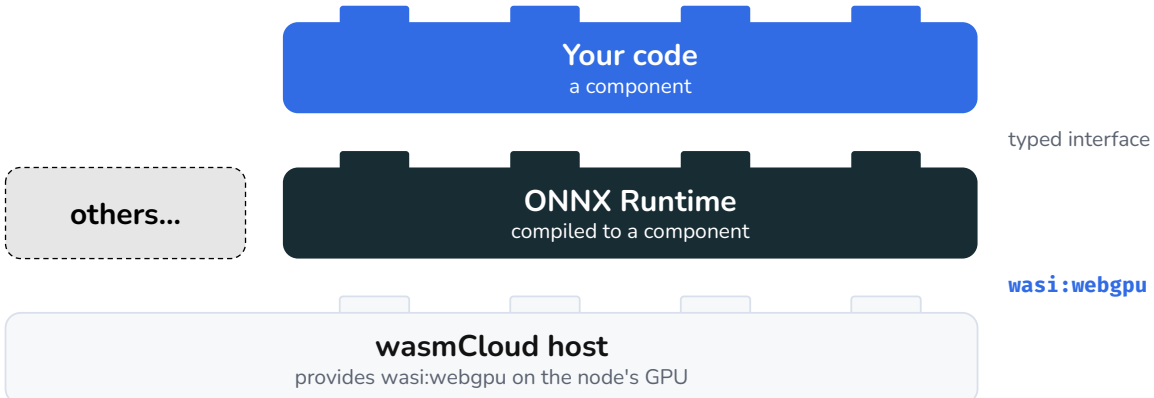
Built so any web page could use your GPU safely. A multi-tenant cluster has the same trust problem.

ONNX: running models in your cluster

ONNX RUNTIME

- What is ONNX?
- How it runs today: containers, KServe, a Python ONNX service
  - Vendor lock-in, brittle operators
- The same model as a Wasm component on CNCF wasmCloud

Components snap together

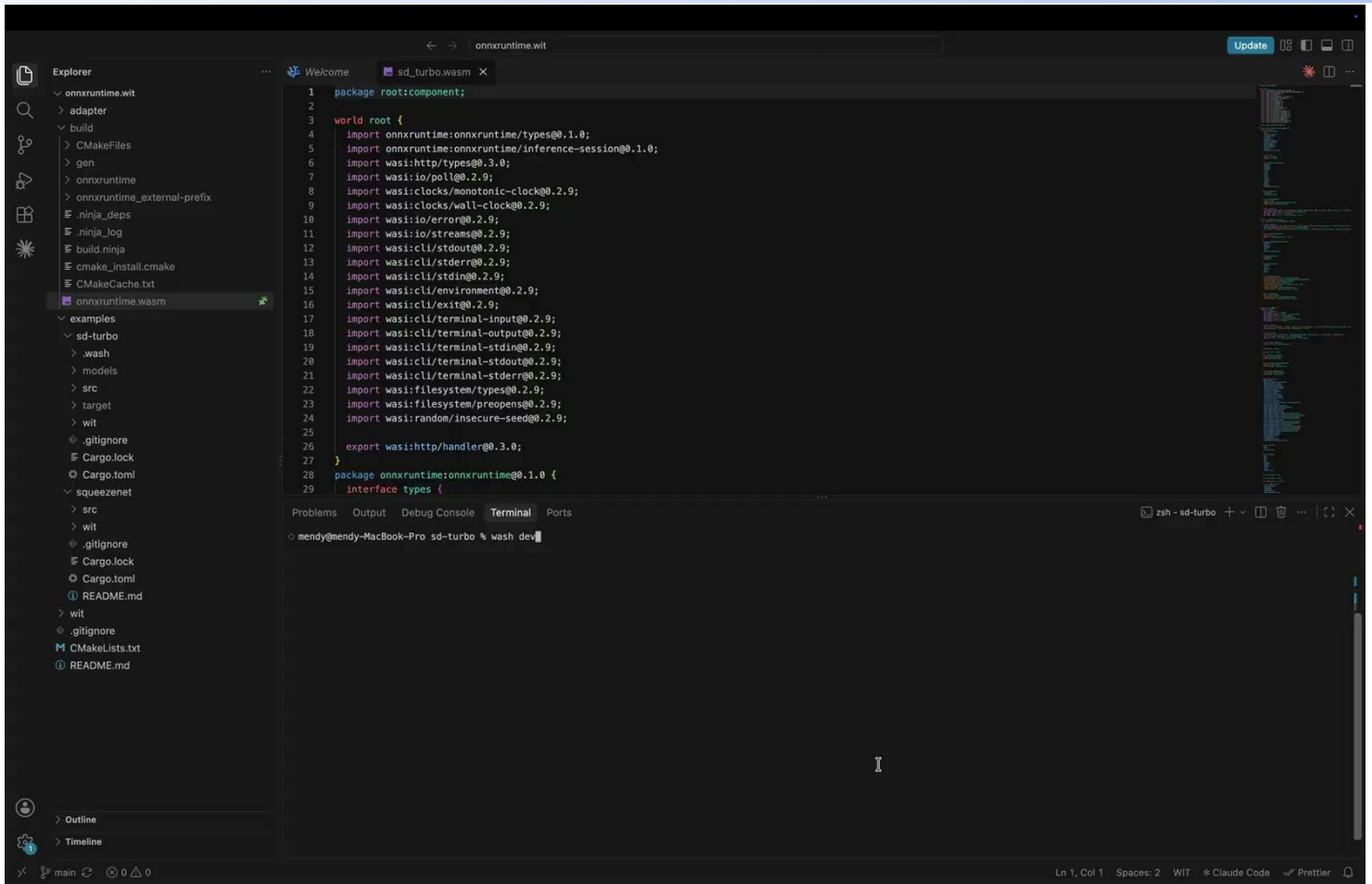


Demo: ONNX Runtime

Live demo



Backup video: ONNX Runtime



The code: one kernel, every GPU

CUDA · NVIDIA only

```
__global__ void add(const float *a,
    const float *b, float *c, int n) {
  int i = blockIdx.x * blockDim.x
        + threadIdx.x;
  if (i < n) c[i] = a[i] + b[i];
}
```

HIP, for ROCm · AMD only

```
#include <hip/hip_runtime.h>
__global__ void add(const float *a,
    const float *b, float *c, int n) {
  int i = blockIdx.x * blockDim.x
        + threadIdx.x;
  if (i < n) c[i] = a[i] + b[i];
}
```

WGSL, for WebGPU · any GPU

```
@group(0) @binding(0)
var<storage> a: array<f32>;
@group(0) @binding(1)
var<storage> b: array<f32>;
@group(0) @binding(2)
var<storage, read_write> c: array<f32>;

@compute @workgroup_size(64)
fn add(@builtin(global_invocation_id)
       id: vec3u) {
  let i = id.x;
  if (i < arrayLength(&c)) {
    c[i] = a[i] + b[i];
  }
}
```

Same shader in the browser (WebGPU) and in a component (wasi:webgpu).

How TensorFlow.js reaches every GPU

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

TensorFlow · Python

```
y = tf.matmul(a, b)
```

TensorFlow.js · JavaScript

```
const y = tf.matMul(a, b);
```

Backends, one for each kind of hardware. The code above doesn't change.

CPU | CUDA | **WebGPU** | WebGL | TPU

↓

Through wasi:webgpu, in a component

NVIDIA · AMD · Intel · Apple · Qualcomm

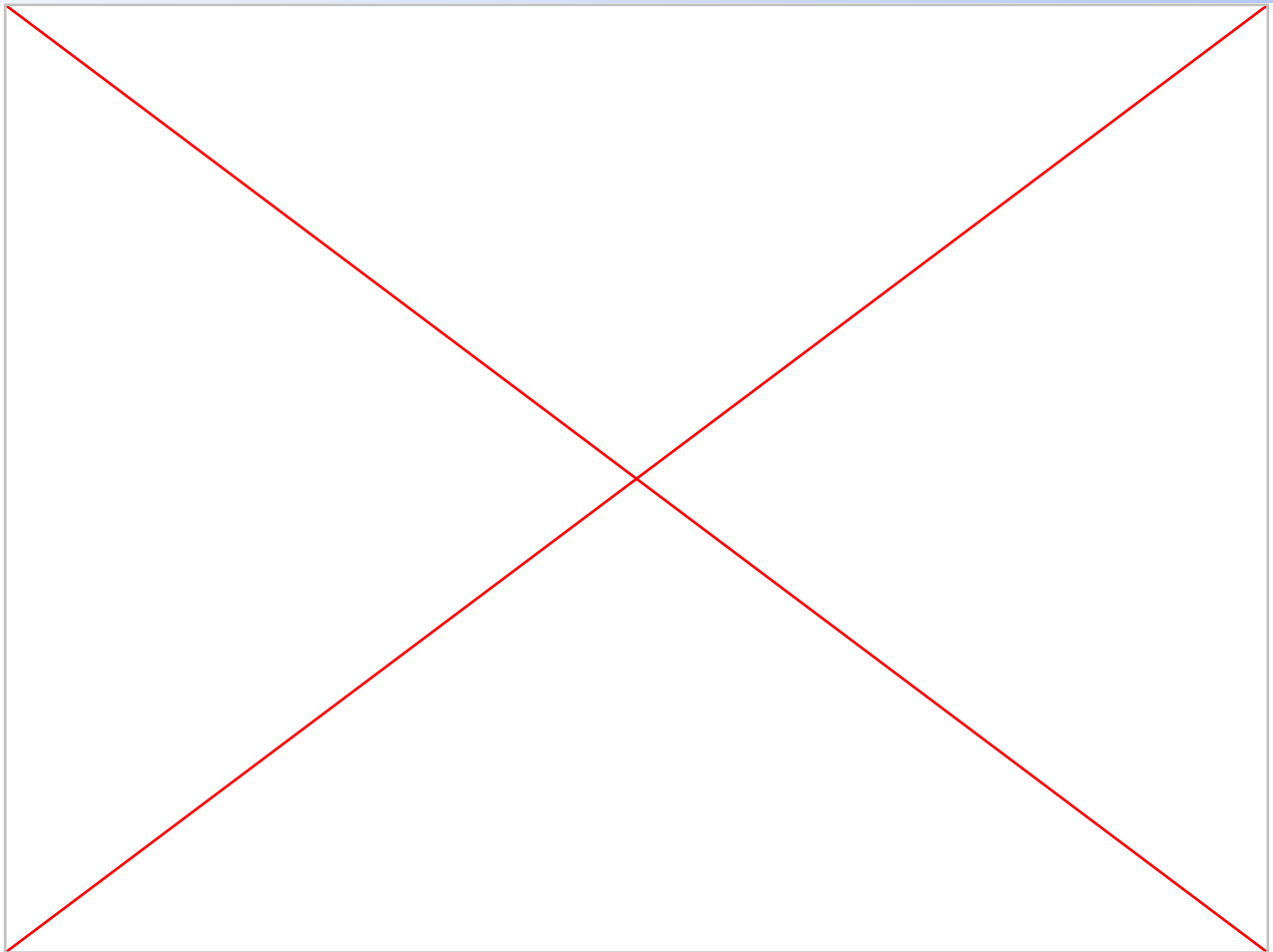
One backend covers every GPU.

Demo: TensorFlow.js image filter

Live demo



Backup video: TensorFlow.js image filter



Closing the gap with native

Feature	What it speeds up	Standardized	WasmCloud Support
Float 16	Half-size weights: less memory, more speed	Yes	Yes
Subgroups	Threads sharing data without memory trips	Yes	No
BFloat 16	Half-size weights with float 32 range	Proposed	No
FP8	Quarter-size weights, even less memory	Proposed	No
Subgroup matrix	Matrix math on tensor-core hardware	Proposed	No

llama.cpp



- We upstreamed `wasi:webgpu` support into llama.cpp
- Runs your existing GGUF models: no retraining, no conversion
- Wasm/wasi memory is limited to 4 GB (for now)

  ...but model weights live on the GPU, so bigger models still fit
- The same binary runs in browser

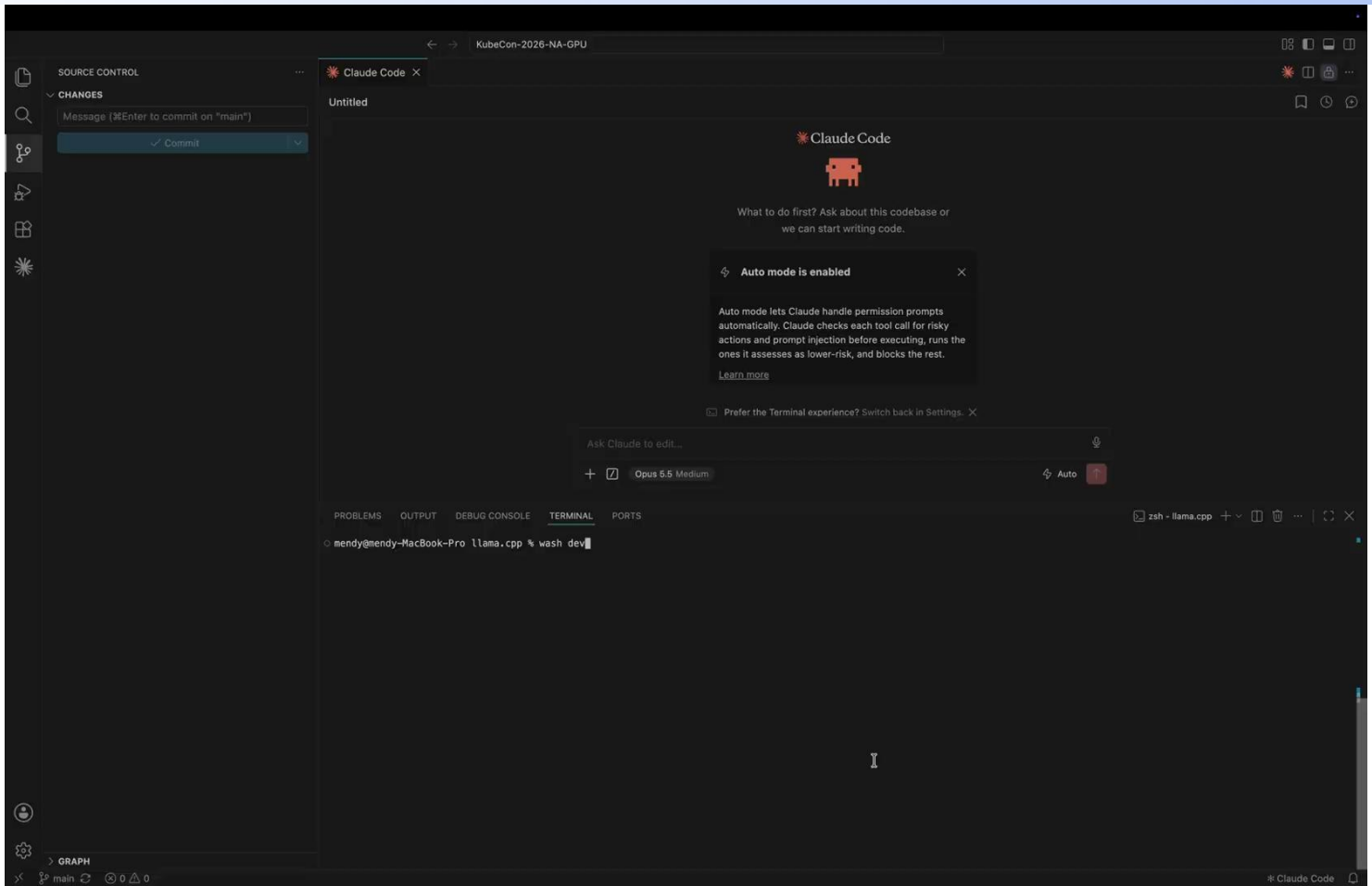
Demo: llama.cpp

Live demo



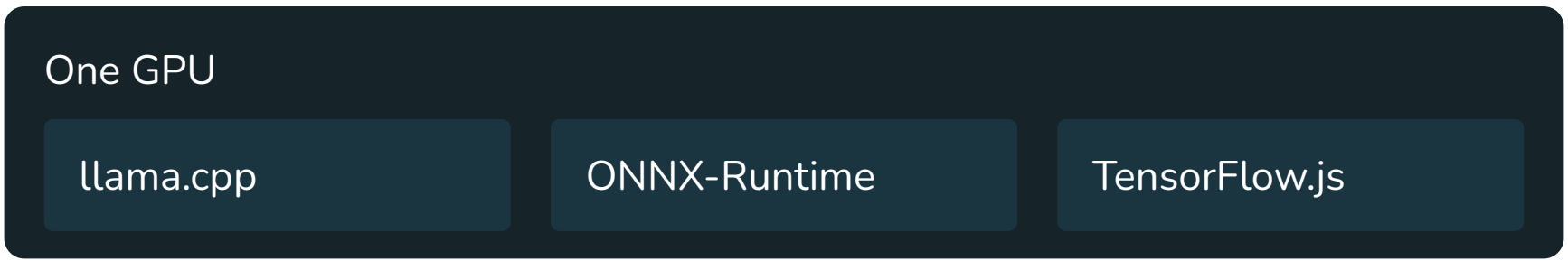
Backup video: llama.cpp

CLOUD NATIVE AI + INFERENCE DAY
NORTH AMERICA



Serverless AI: one GPU, many workloads

- The ONNX model and image filter are still running alongside llama.cpp
- Entirely different workloads, sharing the same GPU



Sandboxing on Kubernetes with WebAssembly

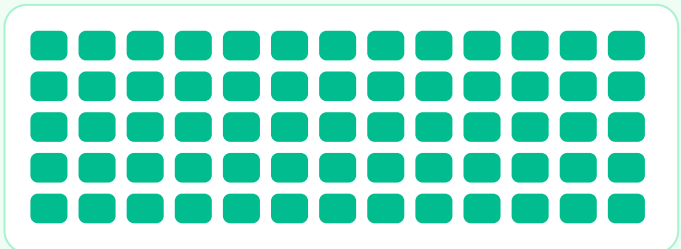
CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Production harnesses, containerized

86MB–1.3GB image size

Idle infra cold starts

A Wasm agent harness



243 KB image size

Scale to zero fresh instance per call

https://arewefastyet.wasmcloud.com/density.html

Harness images: 86 MB to 1.3 GB compressed across 11 harnesses (about 0.2 to 1.1 GB on disk). Agents usually also run in a sandbox, whose image adds 0.5 to 2+ GB and whose default RAM is 512 MiB (E2B) to 1 GiB (Daytona). For scale: Firecracker adds 5 MiB or less for a small microVM, and a small harness can idle near 24 MB (Goose).

Measured from public registry manifests, linux/amd64, 2026-10-29; E2B, Daytona and Firecracker docs.

Do more with less



Models share the GPU

One GPU

- llama.cpp
- ONNX Runtime
- TensorFlow.js

Each in its own sandbox

Agents pack densely

million agents

on one host, at 24 KB each

A 243 KB harness, and a fresh instance for every call

https://arewefastyet.wasmcloud.com/density.html

More models and more agents, on the hardware you already have.

How wasmCloud runs on Kubernetes

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

01 Workload

A custom resource you apply with kubectl, like a Deployment

```
kind: WorkloadDeployment
components:
- agent
- notes-tool
```

02 Runtime operator

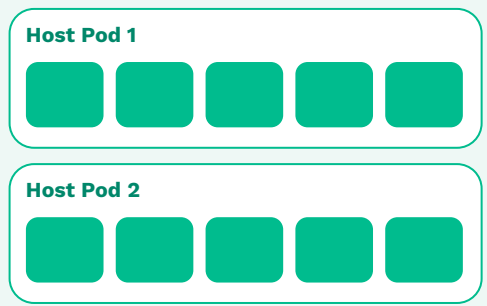
Watches Workloads and places them on hosts, like a Deployment controller

Reconciles & Schedules

Automated lifecycle management

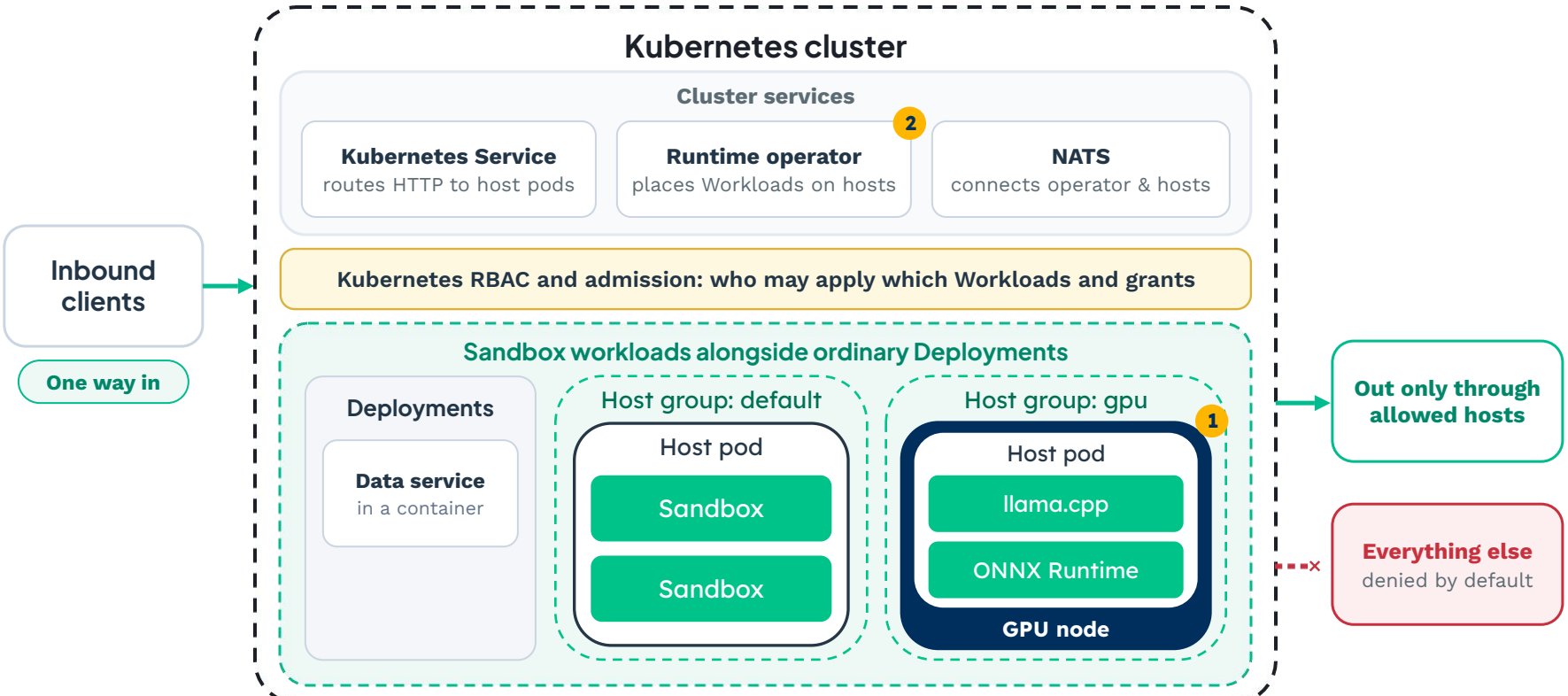
03 wasmCloud hosts

Pods that each run many sandboxes



Runs on your existing Kubernetes cluster

CNCF wasmCloud



1 The gpu host group is scheduled onto GPU nodes by node label; its pod requests nvidia.com/gpu or a DRA claim.

2 The runtime operator places GPU Workloads on that host group, matched by hostSelector.

GPU setup: once per node, not once per image

The GPU still has to reach the pod. What changes is where that setup lives.

Containers: GPU setup in every image

Model service	Model service	Model service
ONNX Runtime	ONNX Runtime	ONNX Runtime
CUDA libs	**ROCm libs**	**Metal path**
NVIDIA image	AMD image	Mac build

A vendor build of every model service, each one rebuilt, scanned and tested on its own.

wasmCloud: GPU setup once per node

llama.cpp | **ONNX Runtime** | **TensorFlow.js**

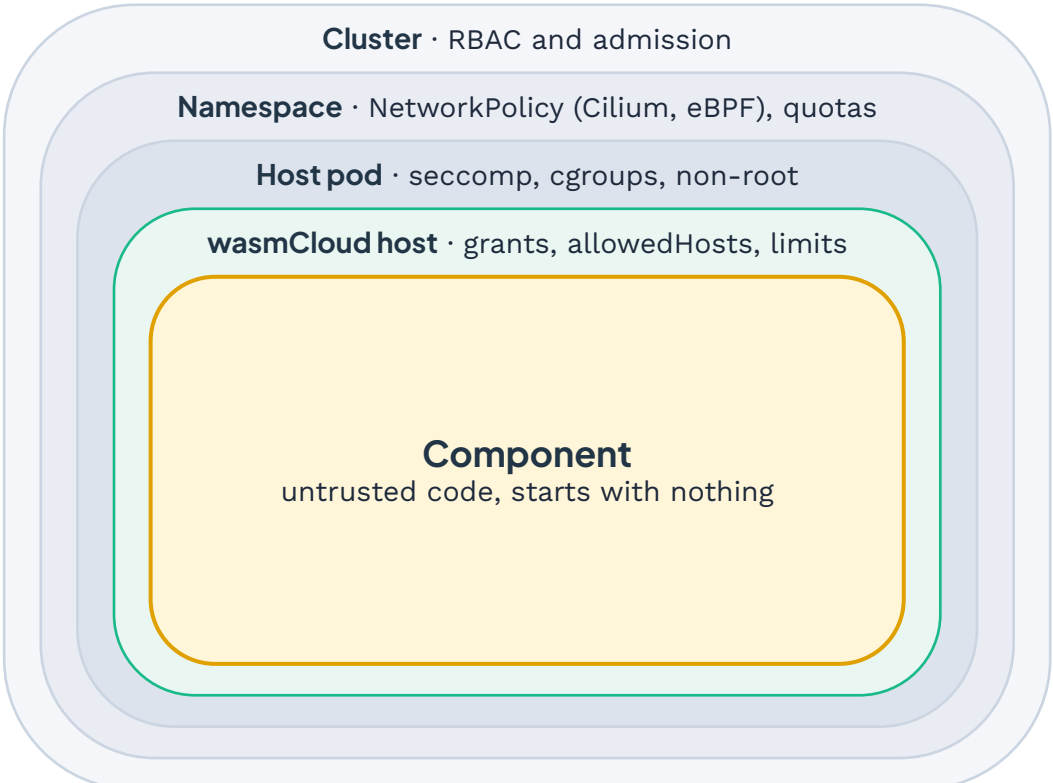
Host pod
Requests the GPU (device plugin or DRA claim) · --wasi-webgpu

GPU node
NVIDIA driver + container toolkit, Vulkan enabled

Set up once per node pool. The same .wasm runs on any vendor's node.

GPU setup moves from every image to once per node.

Kubernetes controls still apply



Components run inside a webAssembly runtime or we call it The Host (what provides the sandbox)

Every control you already run on that pod and its namespace still applies around them. The sandbox adds a deny-by-default layer inside.

Runs on your existing Kubernetes cluster

- wasmCloud hosts run on your GPU nodes
- Workloads are deployed with Kubernetes resources (CRDs)

vLLM, KServe and wasmCloud: different layers

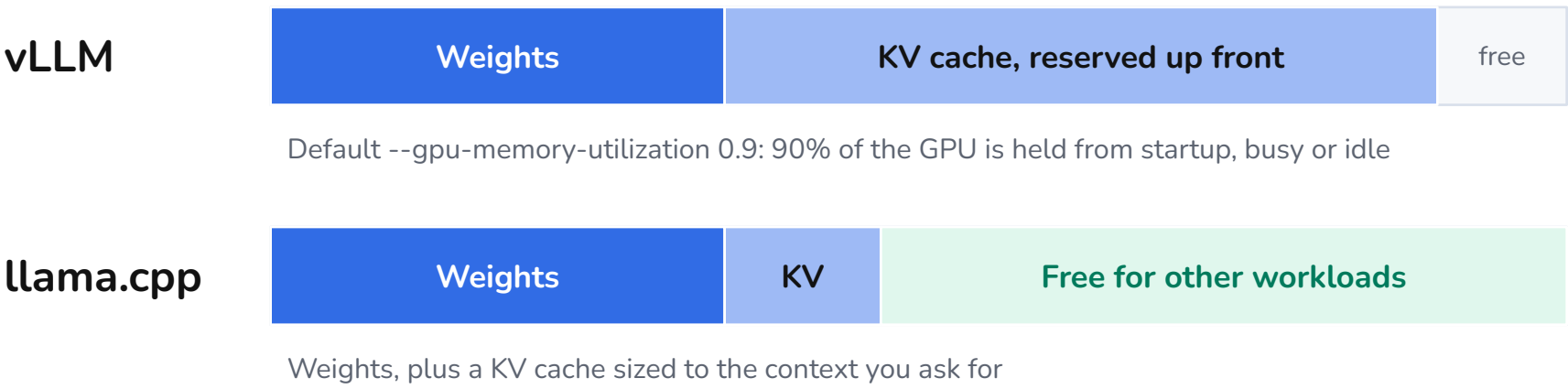
wasmCloud	Runs workloads: the apps and agents that use models
KServe	A serving platform: deploys inference engines on Kubernetes
vLLM	An inference engine: runs a model

A workload can run inference itself through wasi:webgpu, or call the vLLM server you already run.

Maximum throughput for one frontier model on a rack of H100s? Run vLLM, and call it from your components.

GPU memory: vLLM vs llama.cpp

vLLM reserves the GPU when it starts. llama.cpp allocates what the model needs.



Memory a workload doesn't reserve is memory the next workload can use.

[TODO Mendy: replace these illustrative bars with measured numbers, same model on the same GPU]

Compared to other approaches



	Developer laptop	Your cluster
Model execution	The same Wasm component: llama.cpp, ONNX Runtime, TensorFlow.js	
Platform integration	wash dev, on the laptop you already have	wasmCloud on Kubernetes: Workload resources and the runtime operator
Compute	Apple, Intel, AMD or NVIDIA GPU, through wasi:webgpu	NVIDIA nodes and edge GPUs, through wasi:webgpu

vLLM

One model, on a GPU it reserves. Use it for maximum throughput on frontier models.

wasmCloud

The whole stack, the same from laptop to cluster. Workloads share the GPU.

Emerging technology

Today

- WebGPU is slower than Vulkan
- It needs a few more features to get close to parity, like subgroups and subgroup matrix

Only WebGPU

- Portable across graphics backends: Vulkan, Metal, DirectX 12
- Runs in the browser and outside it

Even where it isn't the right choice today, we think it will be.

Why not just Vulkan?

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

One codebase across GPUs is not one artifact that many tenants can safely share.

	The artifact	Isolation	Sharing a GPU
Vulkan · IREE · SYCL One codebase across GPUs	A native build per OS and CPU	The OS process, nothing finer	One app per process
Containers + MIG/DRA Today's Kubernetes answer	An image per vendor stack: CUDA, ROCm	Container per pod	Pods get a slice of a device
wasi:webgpu On wasmCloud: this talk	**One .wasm for any OS, CPU or GPU vendor**	**A WebGPU sandbox per workload**	**Many workloads share one GPU**

Vulkan sits underneath us, not instead of us: on Linux, wgpu runs on Vulkan.

Try it, break it, tell us

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Get started

- wasmcloud.com
- github.com/WebAssembly/wasi-webgpu: the WASI proposals behind wasi:webgpu
- llama.cpp: github.com/ggml-org/llama.cpp/pull/27069

Come find us

- wasmCloud community calls: every Wednesday, with agendas, notes and recordings. wasmcloud.com/community
- wasi-gfx meetings: every Tuesday, 17:00 to 18:00 UTC, open to anyone. github.com/wasi-gfx

links, the slides and the demo source from QR

https://github.com/cosmonic-labs/KubeCon-2026-NA-GPU

Not just GPUs. Sandbox everything!

Hands-on with Bailey and Eric: build a component, run it on Kubernetes, and scope its sandbox.

Sandboxing on Kubernetes with WebAssembly: A Hands-on Tutorial

Wed, Nov 11 · 4:10–5:30 PM MST · Hyatt Regency, Level 4, Regency Ballroom A