

Turning Apple Silicon into Production GPU Nodes for LLM Inference on Kubernetes

Christopher Maher
Director, AI/ML Operations at Ashley Furniture Industries
Creator and maintainer, LLMKube



This laptop is a GPU node in my cluster

```
$ kubectl get inferenceservices
NAME                    PHASE   MODEL                          REPLICAS
demo-qwen38-metal       Ready   demo-qwen38-q4km-metal         1
demo-qwen38-cuda        Ready   demo-qwen38-q4km-cuda          1
demo-qwen38-5060        Ready   demo-qwen38-q4km-cuda2         1
```

Same CRDs. Same Service. One of these runs natively on a Mac.

Who has a Mac with 64 GB or more on their desk?

From hospital desktops to AI inference

Building it

Leading the teams that build it

1

2008–2015

Healthcare

Hospital IT: desktop to systems engineering

2

2016–2019

Aviation, then a startup

Automation engineering

3

2020

Airline

Led four teams on customer-facing technology

4

2022

Tech

SRE and developer velocity

5

2025

Manufacturing and retail

AI/ML operations

Fall 2025: I fell in love a second time. Local AI on my own hardware.

Scope: inference on mixed hardware you already have. Not buying GPUs.

The best accelerators were on my desk

The cluster

Linux servers, no GPUs

Everything runs on Kubernetes

An operator for llama.cpp, the start of LLMKube

→

The desk

2× MacBook Pro, 1× Mac Studio

Apple Silicon GPUs, unified memory

Invisible to the cluster

Can Kubernetes orchestrate models on my Macs? I went looking, and nothing did it.

Why a Mac can't just join the cluster

1 No Metal in containers

Containers on macOS run in a Linux VM. The Metal GPU isn't there.

2 No device plugin

Nothing tells the scheduler a GPU exists, or how much memory it has.

3 Memory the kubelet can't see

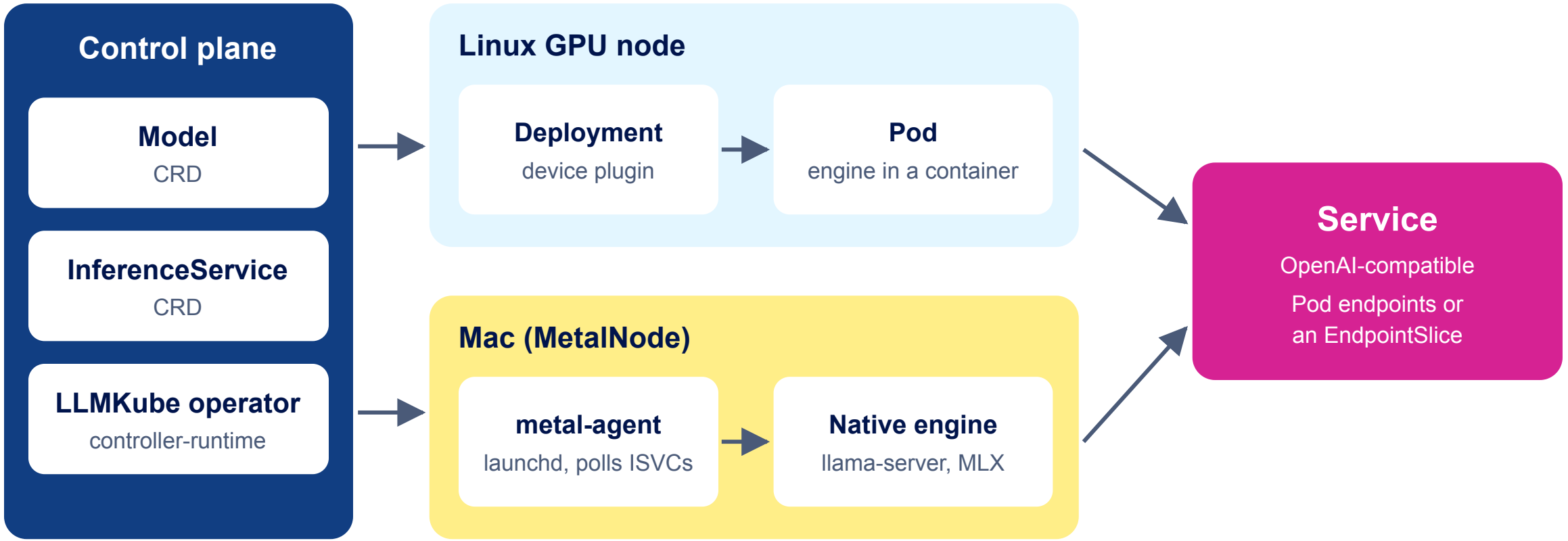
GPU and CPU share one pool. Pod limits don't bound what the GPU takes.



Bring the workload to Kubernetes

Pattern 1

Two paths, one API



The agent pattern

1 Kubernetes can't see

The Mac's GPU, its memory, or a process outside a container

2 The agent does

Polls the Metal InferenceServices in its namespace, starts the engine natively, publishes the EndpointSlice

3 Consumers see

An ordinary Service. Routers, gateways and clients don't change

4 We keep it honest

A Lease heartbeat, and Reachable is verified by dialing back to the Mac

Demo: a Mac joins in two commands

KubeCon CloudNativeCon North America 2026

```
# laptop 2: the cluster
$ llmkube metal token create m5-max -n metal
llmkube://eyJ2IjoxLCJzZXJ2ZXIiOi…    single use, 30 minutes
```

```
# this Mac
$ llmkube metal join llmkube://…
Mac m5-max joined: the cluster reaches it at 10.0.0.47:9443 (log: …)
$ kubectl get metalnodes -n metal
NAME     HOST IP     MEMORY   CHIP           REACHABLE   VERSION
m5-max   10.0.0.47   128Gi    Apple M5 Max   True        0.10.17
```

A single-use string · the Mac's own credential · Reachable proven by dialing back

Example output; captured live in the demo.

KubeCon CloudNativeCon
North America 2026

The runtime is a field, not a fork

Pattern 2



One operator, many engines

```yaml
apiVersion: inference.llmkube.dev/v1alpha1
kind: InferenceService
spec:
  modelRef: qwen38-27b
  runtime: llamacpp
  replicas: 1
```

In the cluster, a backend declares **image, port, args and probes**. On a Mac, a **native executor** runs the engine.

Swapping the engine is a spec change.

Metal (agent)

- llama.cpp
- mlx-server
- oMLX
- TensorFold

CUDA and ROCm (pods)

- llama.cpp
- vLLM
- SGLang
- TGI

Speech

- PersonaPlex (Moshi, CUDA)

Health checks without a kubelet

Pods

1. Startup probe covers the model load
2. Liveness restarts a wedged engine
3. Readiness gates traffic

Macs

1. The agent health-checks the engine
2. A crash or a sleeping Mac drops out of the Service
3. Agent restart adopts the running engine: no reload



Memory is the scheduler

Pattern 3



I didn't lose a process. I lost the laptop.

Discrete GPU

CPU
system RAM

GPU
its own VRAM

Run out of VRAM: one process dies

Unified memory

CPU + GPU
one pool

Run out: the machine goes down

Apple Silicon · NVIDIA GB10 · AMD Strix Halo

June 9, 23:21: a kernel panic on my M5 Max

1

3-4 models resident

Already loaded through the agent

2

One more starts

Not downloaded yet, so its size is unknown

3

Check fails open

“proceeding without check”

4

`--mlock` wires every weight

macOS can't page any of it out

5

Kernel panic

Watchdog timeout, wired memory exhausted

My own laptop, my own lab, my own code.

Check before you start, and fail closed

What changed

Unknown size: refuse, don't guess

Size from Content-Length before the download

`enforce` is the default mode

Count memory already committed to running engines

```
# metal-agent
--memory-check-mode enforce
# refusal, as an Event
InsufficientMemory
```

#640 → PR #641 → v0.8.2
the next day

#2112
cumulative admission

Demo: ask for more than the Mac has

KubeCon CloudNativeCon North America 2026

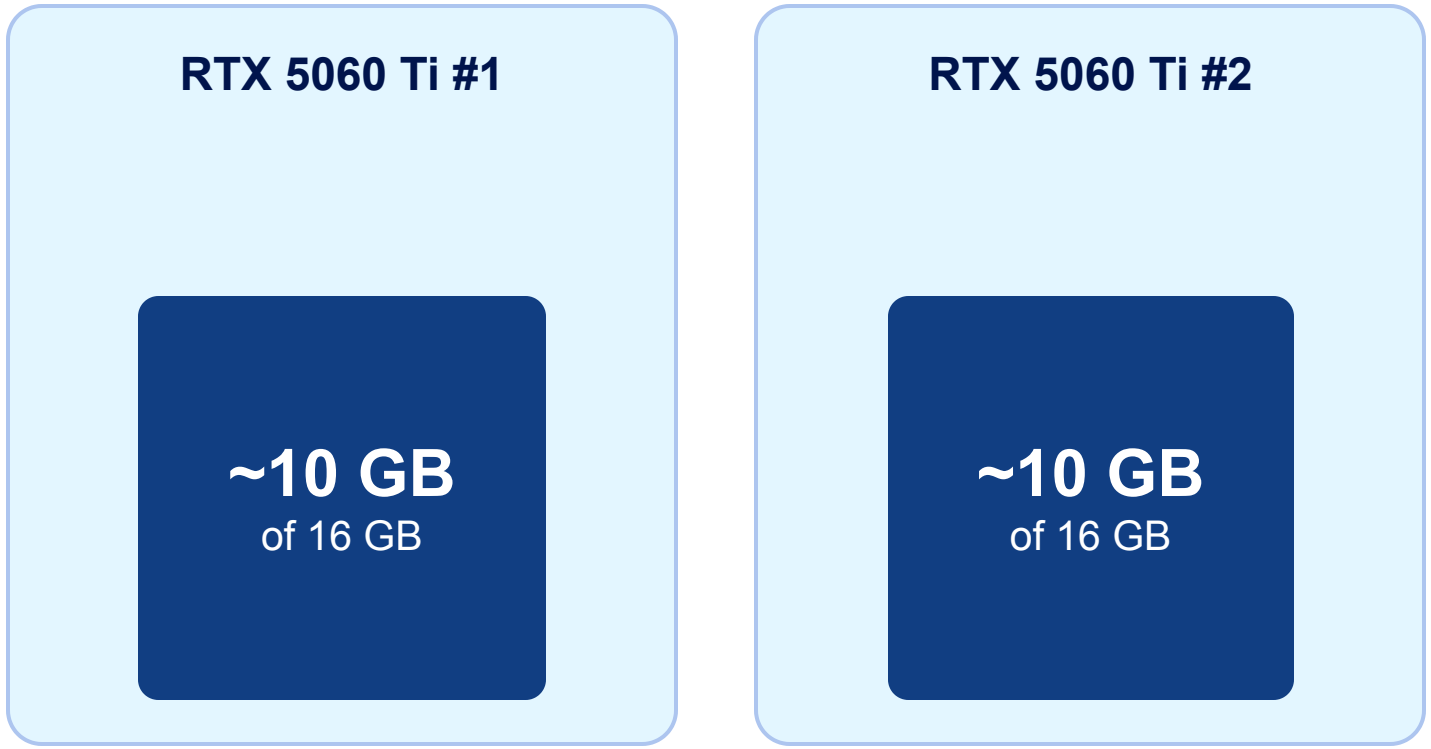
```
$ kubectl apply -f too-big.yaml
$ kubectl get events --field-selector reason=InsufficientMemory
TYPE      REASON              OBJECT
Warning   InsufficientMemory   inferenceservice/too-big
  Refusing to start: estimated 112.4 GB required,
  budget 96.0 GB (128.0 GB total * 75%)
```

```
$ kubectl apply -f fits.yaml && kubectl get isvc fits
NAME   PHASE   MODEL        REPLICAS
fits   Ready   qwen38-27b   1
```

Refused before anything loads, so there is nothing to crash.

Example output; captured live in the demo.

On discrete GPUs, split the model by layer

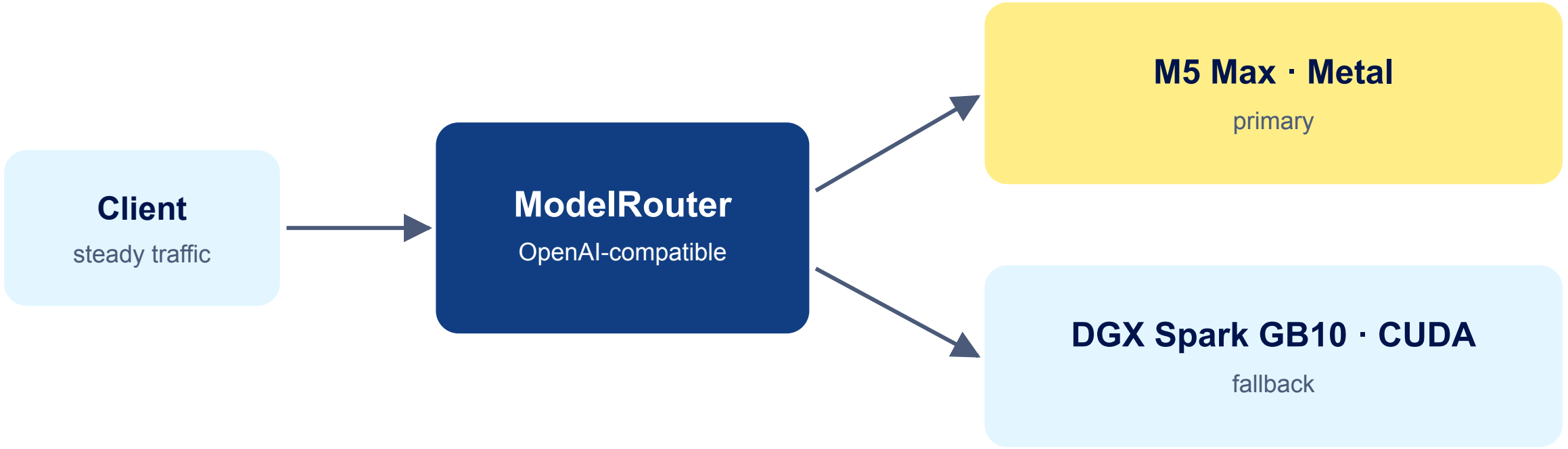



```
# llama.cpp args
--split-mode layer
--tensor-split 1,1
```

Qwen3.8-27B Q4_K_M

19 GB of weights across two 16 GB cards

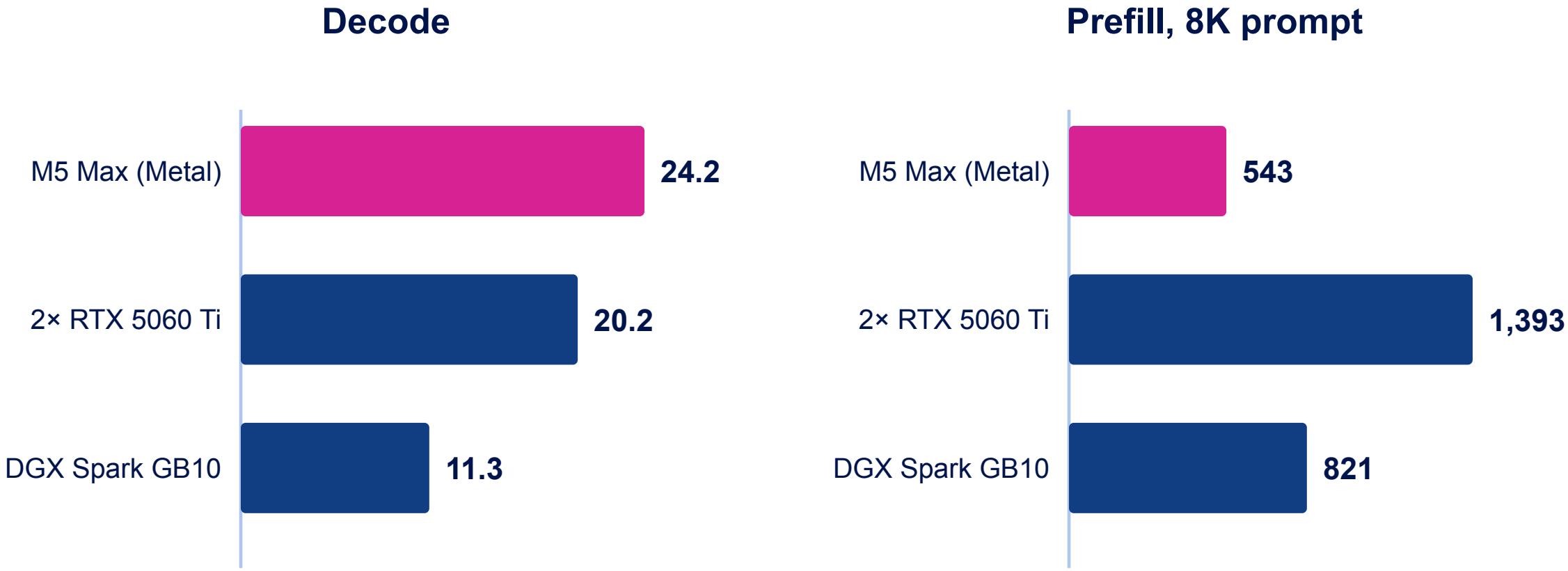
Demo: one API, pull the plug



Rehearsed: 199 of 199 requests OK · Mac back 19 s after the agent restarted · no model reload

Same model, same build, three machines

Qwen3.8-27B dense Q4_K_M · llama.cpp b11146 · no speculative decoding · single stream · median of 3 · tok/s



What the numbers say

Decode follows memory bandwidth

~440 GB/s

read in decode on the M5 Max, rated 614 GB/s, about 90% of what it sustains

Prefill follows compute

1,393

tok/s prefill at 8K on 2× 5060 Ti

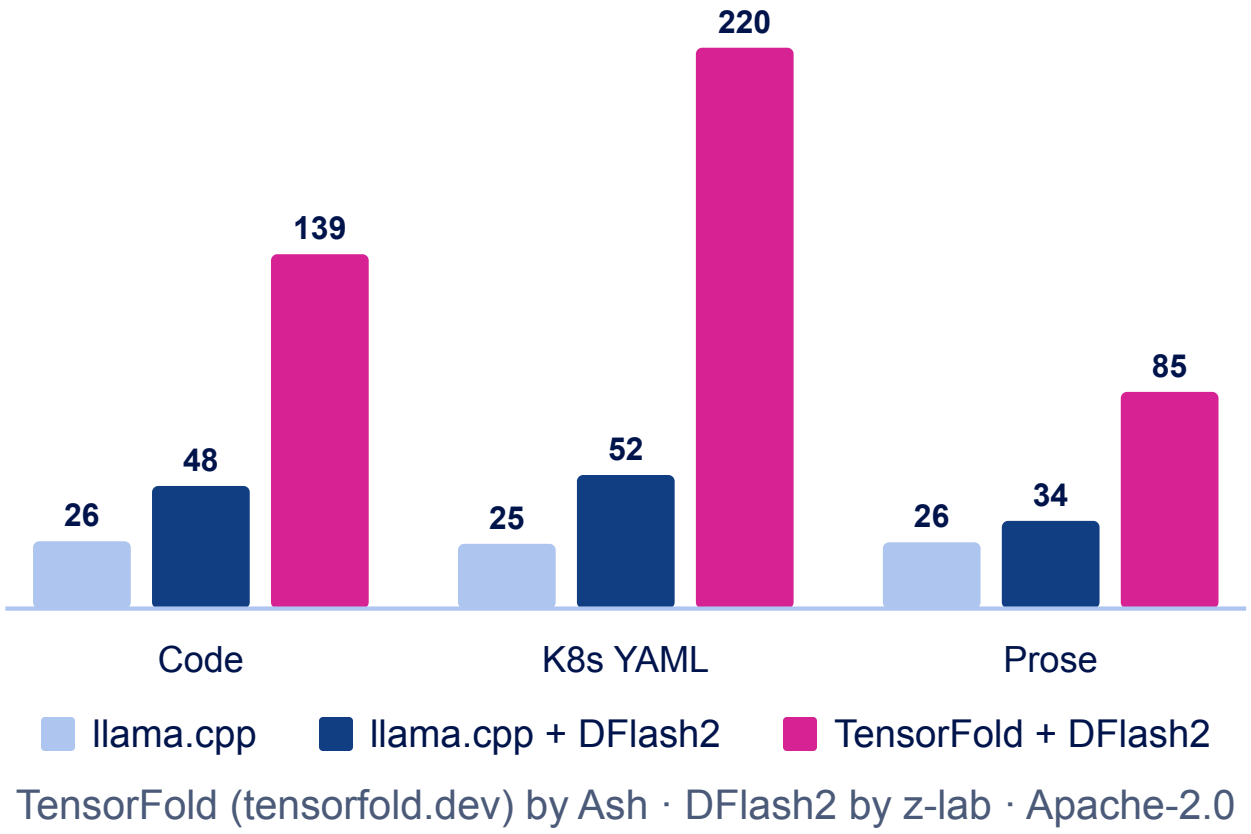
Big unified pools win on what fits

131K

context on 128 GB; the 32 GB pair ran 32K

Same Mac, one field: 3x to 8x faster decode

Qwen3.8-27B 4-bit · M5 Max 128 GB · greedy · 256 tokens · single stream · DFlash2 is the draft model · tok/s



TensorFold (tensorfold.dev) by Ash · DFlash2 by z-lab · Apache-2.0

```
spec:
  modelRef: qwen38-27b-mlx
  runtime: tensorfold
```

Through the K8s Service

227.5 tok/s

K8s YAML · 144.6 code · 89.9 prose

Byte-identical to undrafted output

What doesn't transfer from pods

1 Isolation

A process on the host, not a container. The agent's trust boundary matters.

2 Packaging

Native binaries pinned per host, not images.

3 Sleep and updates

Lids close and macOS updates. The endpoint has to go away cleanly.

4 Local Network Privacy

macOS can block the agent from the LAN until someone allows it.

Running it for real

1 Observe both the same way

PodMonitor scrapes inference pods

Mac engines are scraped through their relay pod

One Grafana panel across CUDA and Metal

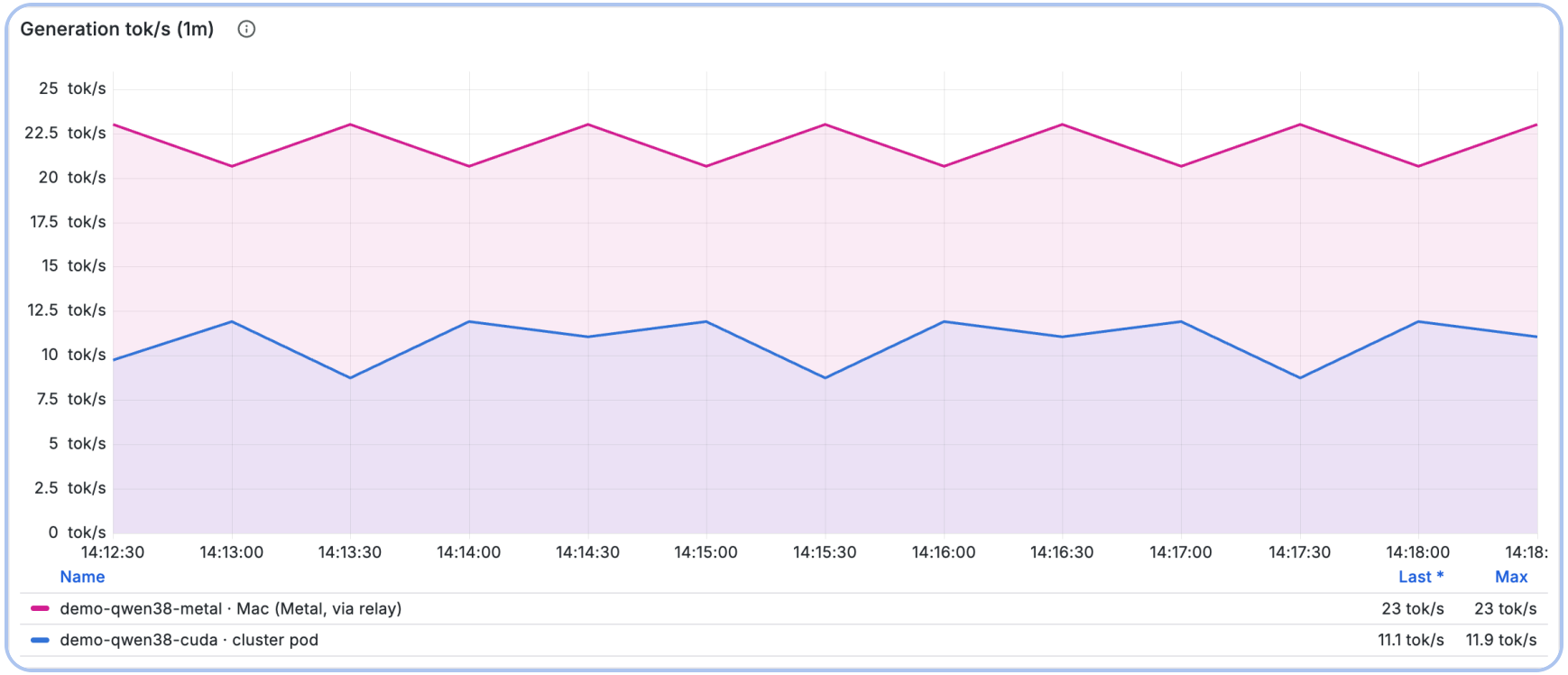
2 Operate Macs like nodes

Signed, notarized agent from a Homebrew tap

Join in two commands, with a short-lived, single-use token

Admission policy limits what the agent can write

One PodMonitor, one panel, both paths



The Mac's engine is scraped through its relay pod, the GB10's directly: same PodMonitor, same llama.cpp metrics.

Live capture from my lab

Back to the laptop

KubeCon CloudNativeCon North America 2026

“If something you need doesn't exist, build it, and build it in the open.”

You never know who it will help, or what it will let them do that they didn't know was possible.

Three patterns to take home

1

Bring the workload to Kubernetes

When the accelerator can't be containerized, use an agent and an EndpointSlice, not a fake pod.

2

The runtime is a field, not a fork

One runtime field lets one operator drive llama.cpp, vLLM, MLX engines and speech models.

3

Memory is the scheduler

Check before you start, fail closed, and count what's already running.

Try it on your own Mac

```
# on the cluster
llmkube metal token create my-mac -n metal
# on the Mac
brew install defilantech/tap/llmkube \
  defilantech/tap/llmkube-metal-agent
llmkube metal join llmkube://…
```

Practitioners: join a Mac to your cluster.
Operator authors: borrow the agent + EndpointSlice pattern.

github.com/defilantech/LLMKube


KubeCon
CloudNativeCon
North America 2026
Thank you.
Questions?