

CONTAINERDAYS 2026 SINGAPORE · CNCF INCUBATING PROJECT

10x Your Neocloud's GPU Utilization with HAMi

What a neocloud sells, and what each product needs from the GPU

Reza Jelveh

Head of Global Market, Dynamia

 github.com/fishman  linkedin.com/in/rezajelveh

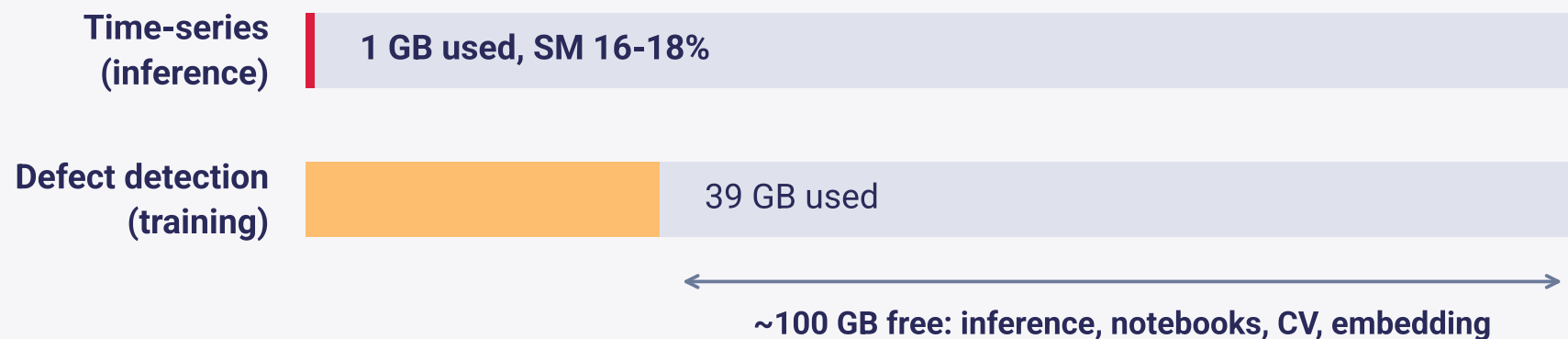


One job, 1 GB of a 141 GB H200

Allocated is not used

GPU MEMORY: ALLOCATED VS USED

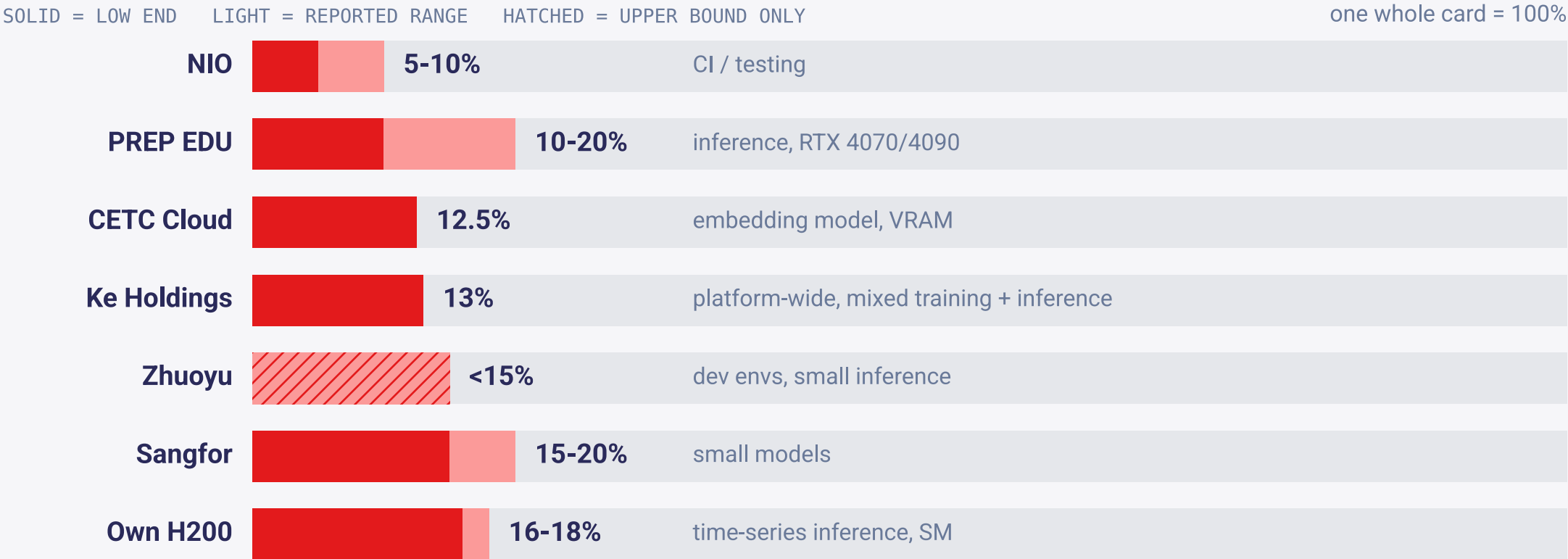
one whole H200 = 141 GB



- ▶ The time-series job leaves **140 GB and over 80% of the SMs** doing nothing
- ▶ Many cards have **no MIG** at all: L40, L40S, A6000
- ▶ Kubernetes gives one choice: nvidia.com/gpu: 1. The whole card, or nothing.

Seven teams measured the same 5-20%

Utilization before sharing, as each team reported it (workload, platform and VRAM measures differ)



Different industries, chips and countries. Most of every card: paid for and idle.

Why DRA alone can't fix it

Asking for part of a GPU doesn't stop a job using all of it

Device plugin

`nvidia.com/gpu: 1` is an integer: the whole card or nothing. The 1 GB job gets 141 GB.



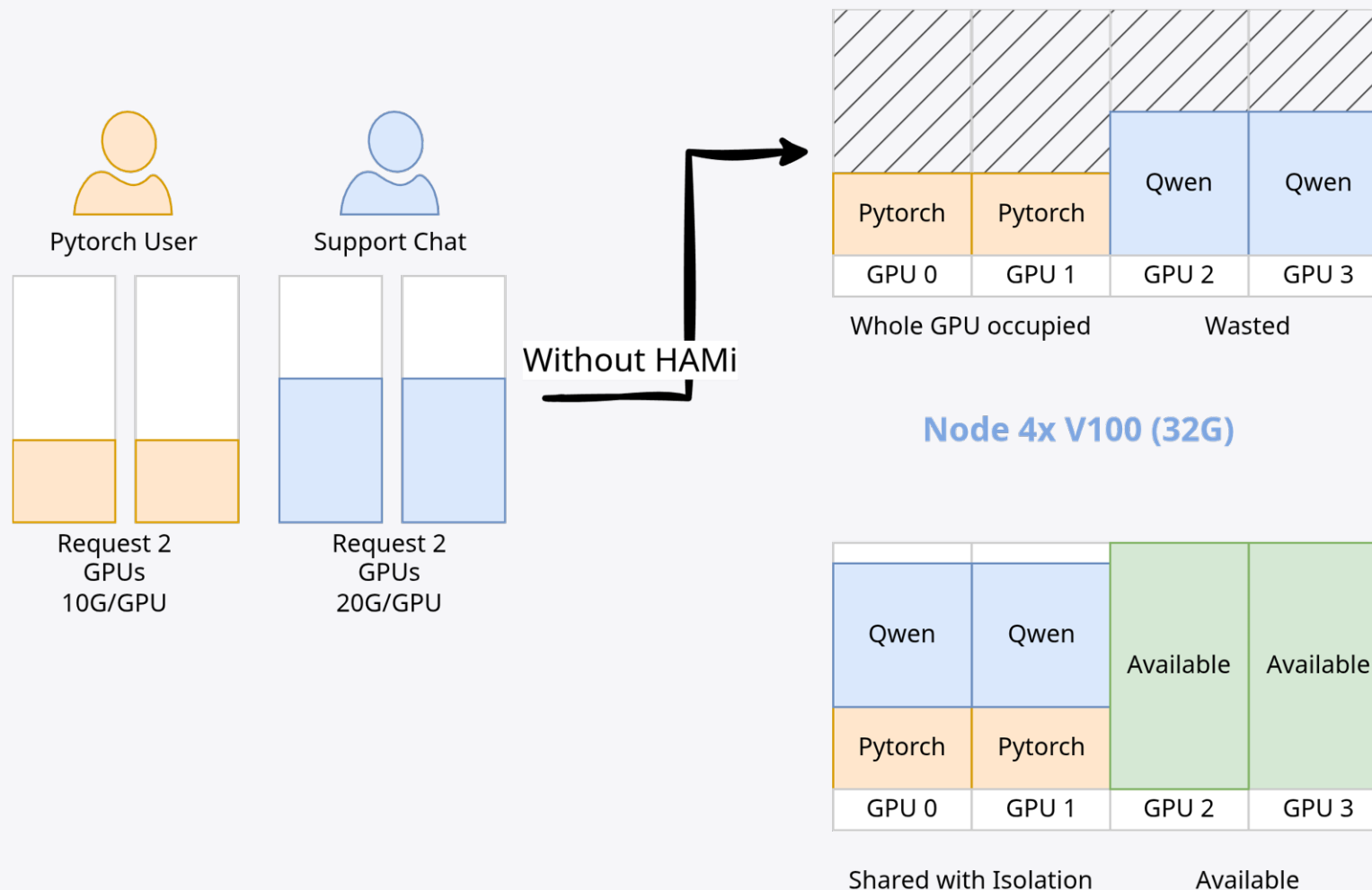
DRA alone

- ▶ **Request:** a claim can ask for MiB and %
- ▶ **Limit:** nothing stops the pod using more memory or compute
- ▶ **Pack:** nothing places shares onto one card

"GPU sharing without smart scheduling leads to suboptimal placement; scheduling without GPU virtualization leaves nothing to schedule." Sangfor

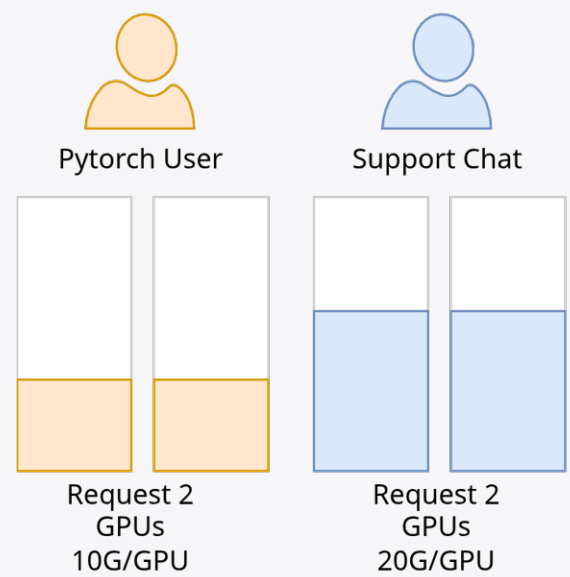
What is HAMi

Static allocation, one GPU per task



What is HAMi

Fractional vGPUs, multiple tasks per device



With HAMi



Pytorch	Pytorch	Qwen	Qwen
GPU 0	GPU 1	GPU 2	GPU 3

Whole GPU occupied

Wasted

Node 4x V100 (32G)

Qwen	Qwen	Available	Available
Pytorch	Pytorch		
GPU 0	GPU 1	GPU 2	GPU 3

Shared with Isolation

Available

HAMi Capabilities

Six things HAMi brings to GPU scheduling

Heterogeneous Management

Manage GPU, NPU, MLU, and other accelerators in one workflow.

Hard Isolation

Slice memory and compute with hard isolation at runtime.

Advanced Scheduling

Binpack, spread, and topology-aware placement policies.

Kubernetes Native

Kubernetes-native APIs, DRA, and CDI support.

Resource Isolation & QoS

Memory and core quotas for fair, stable sharing.

Unified Monitoring

Consistent metrics and visibility across vendors.

What a neocloud sells

A GPU-first cloud: each product needs something different from the GPU

GPU capacity

Whole or fractional: sharing, topology-aware placement

Notebooks, dev envs

Isolation between users, per-user limits, idle reclaim

Inference

Endpoints and model APIs: memory-aware packing, autoscaling

Batch and training

Queues, fair share, gang scheduling, priorities

Mixed fleet

Any vendor underneath, one request API

Tenants and billing

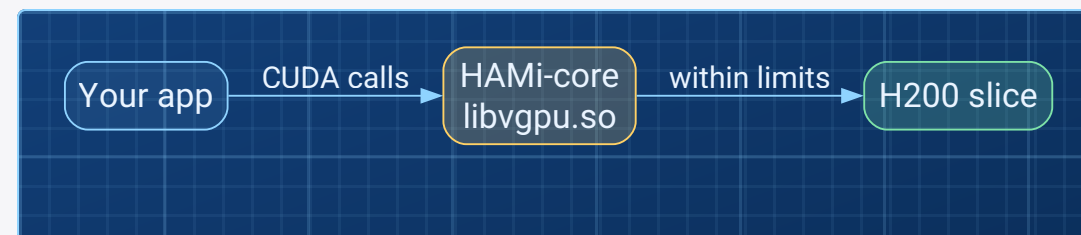
Quotas, metering, accurate per-GPU usage records

Fractional GPUs: split the card

How vGPU interception caps memory and compute

- ▶ Pod asks for **MB** of memory and % of compute
- ▶ Scheduler picks a card **UUID** with headroom
- ▶ Device plugin injects **libvgpu.so** via `ld.so.preload`
- ▶ HAMi-core intercepts CUDA: allocations capped at the slice, launches paced by a % **token bucket**
- ▶ `nvidia-smi` in the pod shows **the slice, not the card**

```
resources:  
limits:  
  nvidia.com/gpu: 1  
  nvidia.com/gpumem: 15000    # MB  
  nvidia.com/gpucore: 30     # % of SMs
```



Fractional GPUs: two fences on one card

Memory is a hard wall, compute is a pace

Memory: MB slices

MEMORY: ONE H200, 141 GB



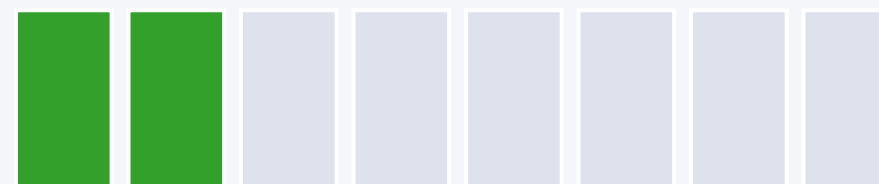
notebook
(training)

time-series
1 GB used

next tenants

Compute: paced on every launch

COMPUTE: % OF SMs (ILLUSTRATIVE)



←→
pod A: 25%

75% free for other pods

- ▶ **Memory is the hard wall:** an OOM cannot wait, so it stays inside the slice
- ▶ **Compute is a pace:** a shortage becomes delay, no burst above the share

Notebooks: the price of a guaranteed slice

Per-user limits cost something: overhead as measured, ours and theirs

+20 min

1 h training run capped to
70% cores (own H200)

within 5%

throughput on 8 GB, 20%
compute (CETC)

+46.7%

card throughput,
partitioned (CMB)

1/6 GPU

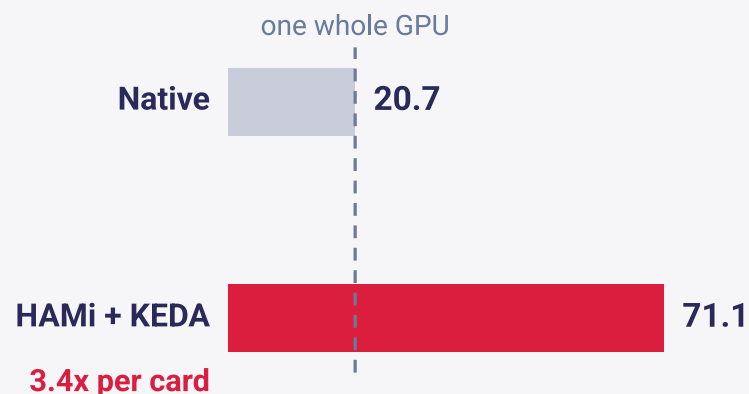
beat 1/8: finer is not
better (NIO)

- ▶  **+20 min** buys a reserved share, not a "maybe free" one
- ▶  **Size memory first:** an OOM cannot wait, a compute shortage becomes delay (NIO)
- ▶  The cap protects **stability, not throughput:** hard throughput isolation is MIG

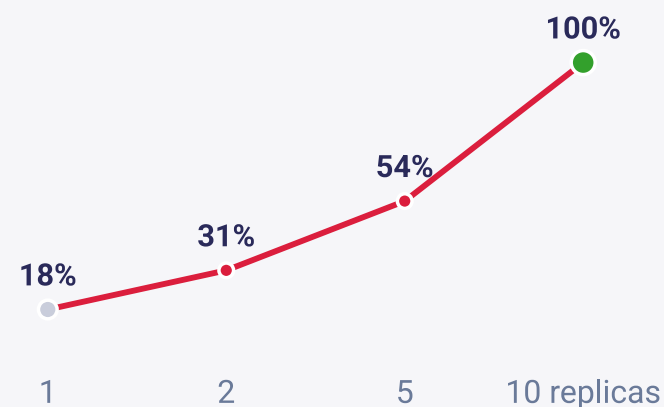
Inference and model APIs: 1 to 10 replicas

Fix the SLA, pack replicas by memory, autoscale, fill the empty card

REQ/S PER CARD @ P95 160 MS



SM UTILIZATION VS REPLICAS

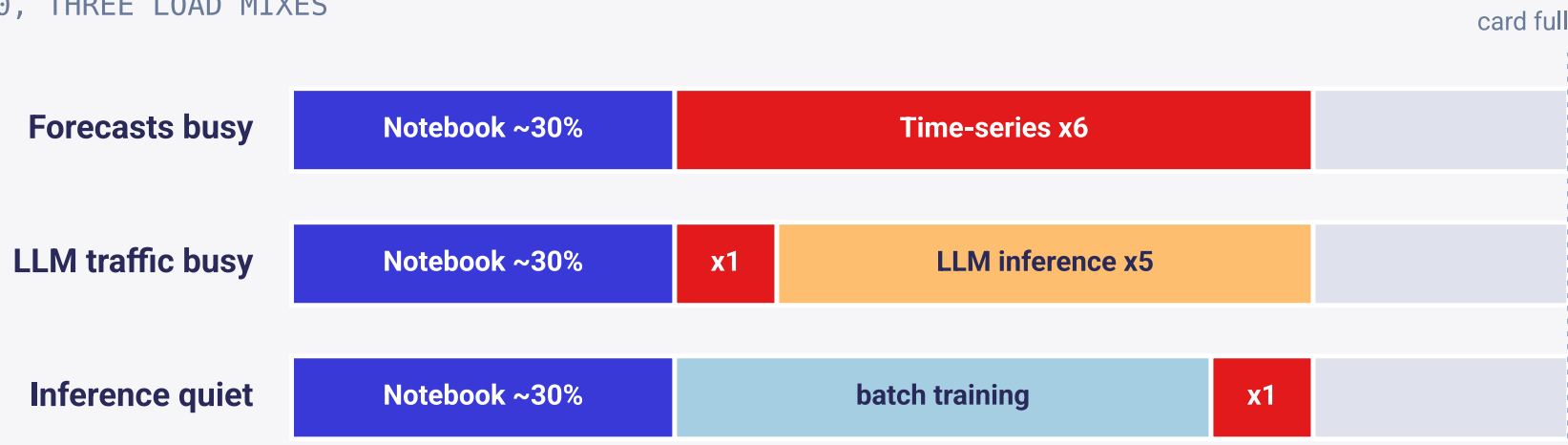


- ▶ One pod uses **18%** of the SMs. Ten replicas on the same card: **100%, same p95**
- ▶ HAMi does not make one pod faster: **it fills the card that sat empty**
- ▶ Bigger batches did not help: **batch size 32 to 512**, GPU still at 7-10%
- ▶ Model APIs, same move: **many models per card** (Sangfor: 1 to 8+ per GPU)

Batch and training: the scheduler decides

Notebooks, inference and training share one card; placement and queues decide who gets it

ONE H200, THREE LOAD MIXES



- ▶  **Placement:** per-card capacity, binpack or spread per node and per card, topology-aware
- ▶  **Hand-off:** KEDA scales replicas, so freed slices go to whoever peaks next
- ▶  **Queues, gang, fair share:** your brain on the same fences: Volcano, Kueue, Koordinator, KAI

What production users don't share

Share small, bursty, interactive. Keep big training whole.

Workload	How it runs	Who
Large training	Whole card	Ke, NIO
Large LLM serving	Whole card	CETC, Ke
Small or bursty inference	HAMi slice	Ke, Sangfor, DaoCloud, SF, PREP EDU, NIO
Dev envs, notebooks, CI/test	HAMi slice (MIG for hard isolation)	CETC, Zhuoyu, NIO, SF, own
Embedding, rerank, simulation	HAMi slice	CETC, NIO

Profile workloads first, not a preset ratio (CETC).

Mixed fleet: the second card isn't NVIDIA

Every vendor's chips first-class: one pool, one API, a different limiter per vendor

Hardware	Compute cap
NVIDIA, Ascend	User-space interception, 1% token bucket
AMD CU masking, NVIDIA MPS	Hardware-honoured, static
Kunlunxin vXPU, Hygon vDCU, MIG	Hardware-fenced, coarse

nvidia.com/gpucores: 30 # same shape,
huawei.com/Ascend910C-core: 20 # any vendor

⤴ More than 8 clusters avoided

China Merchants Bank: Kunlunxin, Ascend, NVIDIA, PPU, Moore Threads, MetaX in **one pool** (counterfactual, per CMB)

🏗 Topology counts

Paired allocation on supernodes: **100% hardware pool utilization**, cross-machine scheduling **-30% (CMB)**

Requests: where DRA fits, and falls short

The request changes; what caps the pod on the card doesn't

✓ What DRA gives

- ▶ **HAMi-DRA** webhook rewrites `nvidia.com/gpu` into **ResourceClaims**
- ▶ K8s **1.34+**, Consumable Capacity gate + CDI
- ▶ Typed capacities: MiB, % cores



⚠ Where it falls short

- ▶ **Reserves** a slice, but nothing on the card **stops the pod using more**
- ▶ Shows what was **reserved**, not what is **used**
- ▶ No CNCF case study runs it in production yet

Keep the Pod API stable now, move to DRA underneath when your Kubernetes and scheduler are ready.

Sell and meter: the slice becomes the product

DaoCloud D.run and DCE: the box became a cloud

10,000+

GPU cards under HAMi, D.run + DCE

80%+

avg utilization per card

20-30%

lower GPU opex

How DaoCloud turns cards into products:

✂ 1. Define

vGPU unit: fixed
memory + compute

🏷 2. Package

SKUs, pay by
workload size

📦 3. Fill

Controlled over-
subscription

👥 4. Govern

Quotas + RBAC per
vGPU (DCE)

Operate it: allocation is not utilization

Quotas and metering need both numbers

≡ Meter both

Bill from allocation
(scheduler : 9395),
plan from usage
(vGPUmonitor : 9394).

👤 Tenant quotas

ResourceQuota on
gpumem and
gpucore per
namespace.

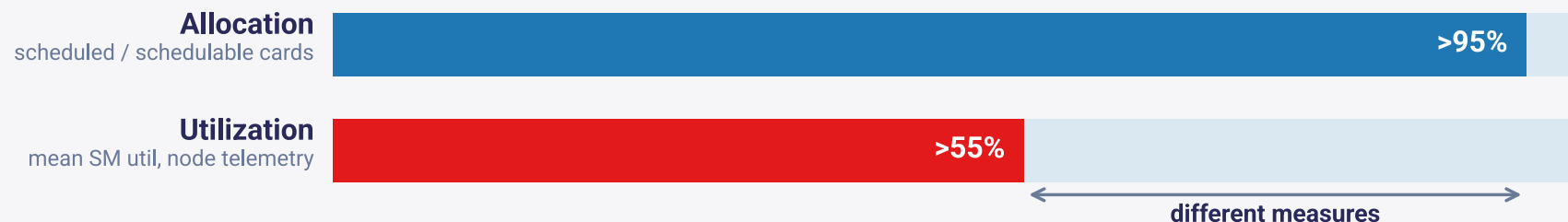
🔄 Upgrades

NIO: blue-green
device plugin, phased
rollout, perf checked
per version.

📈 Scale ahead

SNOW: KEDA scales
on queue saturation.
700% spike handled,
2x fewer GPUs.

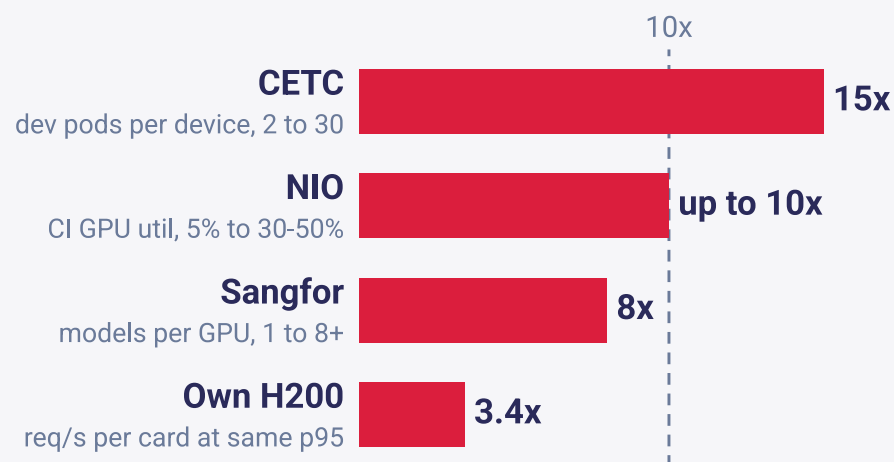
ZHUOYU, SAME FLEET, TWO NUMBERS



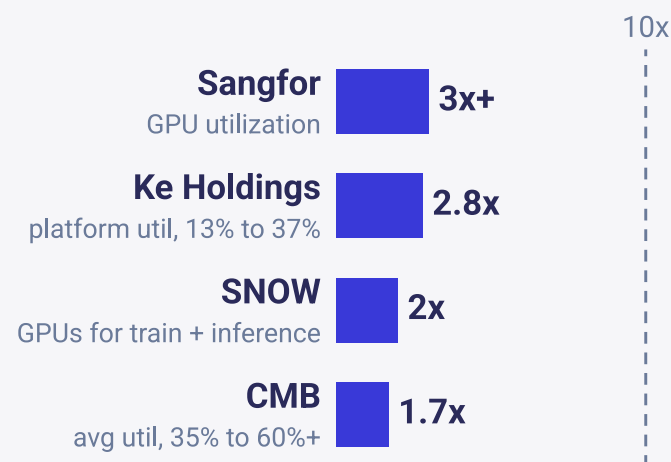
It's not just one H200

Per-workload gains up to 15x, platform-wide 1.7-3x

PER WORKLOAD



WHOLE PLATFORM



- ▶ **Whole-card baselines in the case studies: 5-20%.** NIO's CI went 5-10% to 30-50% (headline 10x, text ~4x)
- ▶ Whole platforms land at **1.7-3x**. Our reading: training already uses its cards

Honest limits

No vendor pitch: where sharing stops

Not a security boundary

CUDA_DISABLE_CONTROL, static CUDA, raw ioctls bypass the preload. Trusted tenants only

MIG for hard isolation

NIO: algorithm dev on MIG. Untrusted tenants: MIG or whole cards

Training stays whole

Ke, NIO keep large training on whole GPUs; CETC keeps LLM serving exclusive

Finer is not better

NIO: 1/6 GPU beat 1/8

Interception has edges

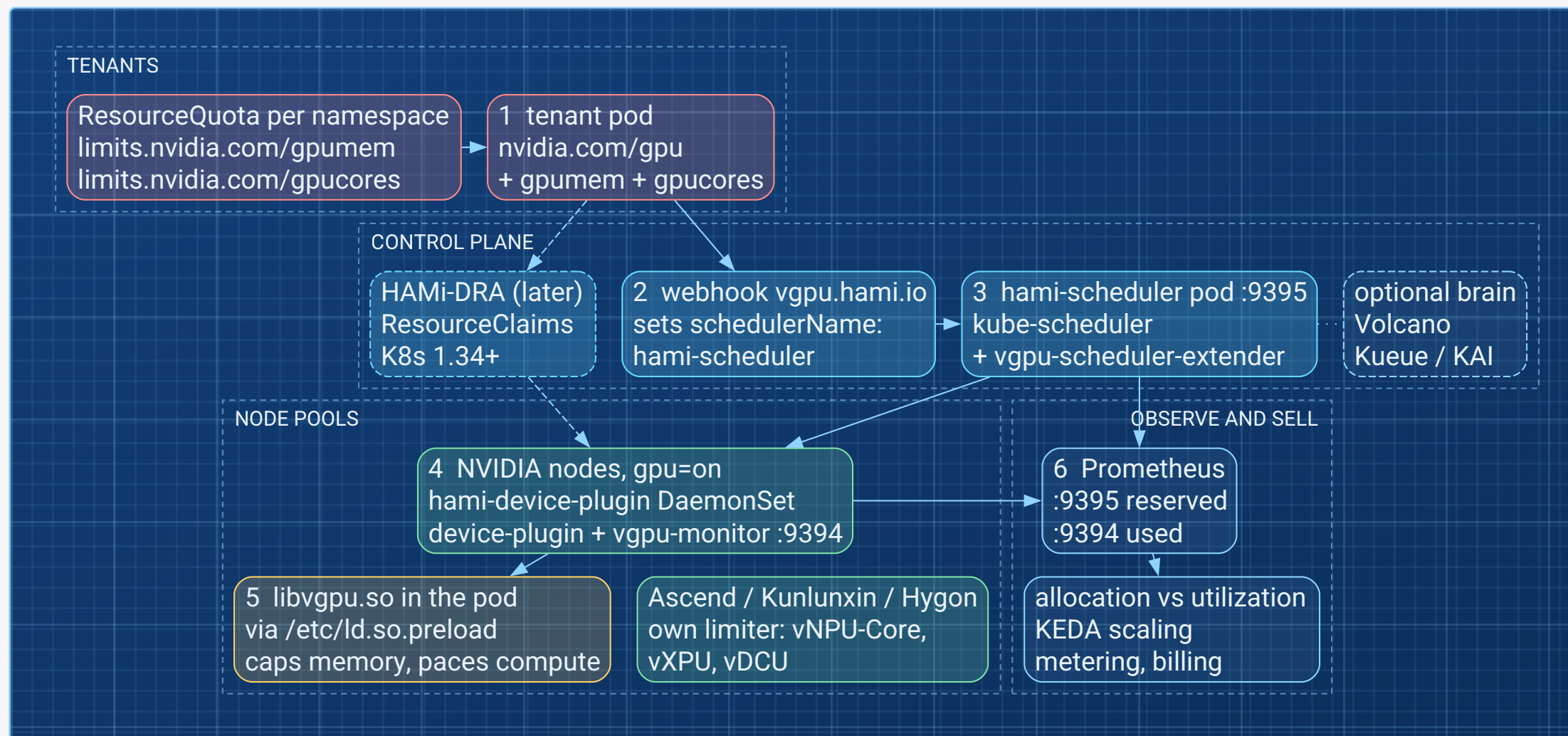
Toolchain clashes, validate every upgrade (NIO)

Results are stack results

KEDA, Kueue, Fluid share the credit (CMB). SNOW credits KEDA for its GPU-time cut

Blueprint: GPU box to GPU cloud

The reference architecture, HAMi chart v2.10.0



Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Incubating, Production Ready

4.9k

GitHub Stars

500+

Contributors

20+

Contributor Countries

74

Public Adopters



Ecosystem & Device Support



Adopters



CNCF INCUBATING · HUNDREDS IN PRODUCTION · 4,900+ GITHUB STARS

Thank you

Questions?

Reza Jelveh

Head of Global Market, Dynamia

🐙 github.com/fishman  linkedin.com/in/rezajelveh

