OPEN SOURCE SUMMIT KOREA 2026 - EMBEDDED & OPEN AI

Simplifying AI for Edge Compute with HAMi

Slicing one device, running many agents

Reza Jelveh
Solution Architect, Dynamia AI - Makers of HAMi
github.com/fishman linkedin.com/in/rezajelveh

DYNAMIA.AI
Creators of HAMi

Part 1: The Problem

The model is not the hard part, the compute is

서울

Not All Edge Compute Is Agents

Most edge inference today is CV and classic ML

Most edge inference is CV and classic ML, not LLMs. Agents are the new slice.

CV and classic inference

- YOLO, classification, OCR, embeddings
- 1-50M params, millisecond latency
- Runs on NPUs and embedded GPUs today

LLMs

- Autoregressive text generation
- Weights plus KV cache in memory
- Needs GPU-class silicon

Agents

- LLMs in a tool-use loop
- Several concurrent per device
- Memory-hungry, latency-tolerant

One deployment pattern for all three: same device sharing for small-model inference and agent tasks.

Edge Agents, Starved Compute

Open-source agents need edge hardware

Open-source agents like Hermes and OpenClaw can reason and use tools. Edge deployment is still hard: not the model, the compute underneath.

Limited memory

A Jetson-class device has 8-64 GB of unified memory, shared with the OS. One agent stack can eat it all.

Tight power budgets

No 300 W data-center GPU at the edge. You get 5-40 W, often battery or solar.

Unattended

No cluster admin on call. Prometheus may scrape, but no one watches the dashboards. It has to work after setup.

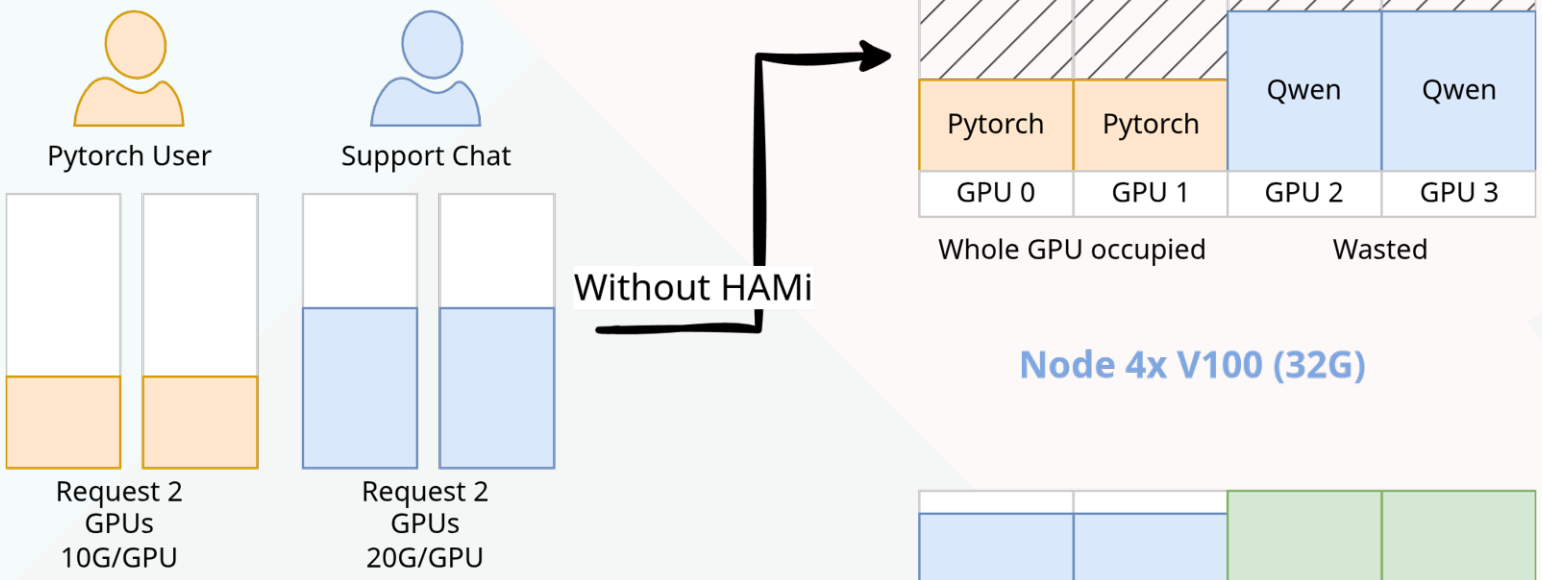
No elastic scaling

Cloud load grows: add GPUs. Edge load grows: nothing to add. The deployed box is all you get.

To run agents at the edge, fix the compute layer first

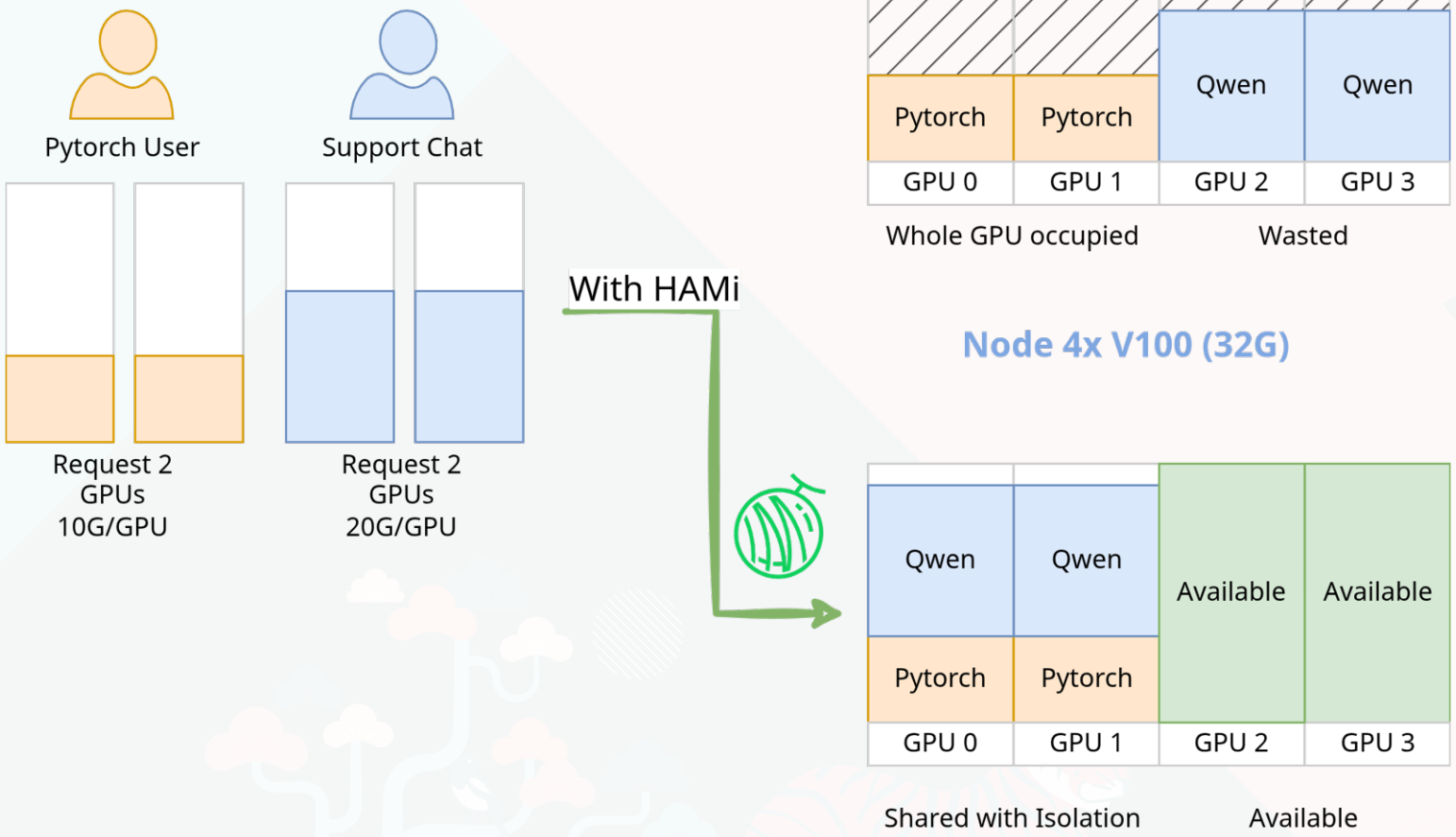
What is HAMi

Before: one device, one task



What is HAMi

After: one device, many agents



The Edge Compute Challenge

What breaks, what HAMi needs to solve

DYNAMIA.AI
Creators of HAMi

Problem

- One model per device, most of it idle
- Fixed memory budgets: install once, never share
- One bad agent takes down the device
- Requirements vary wildly: tiny CV filters to large agent LLMs

Requirements

- Guaranteed to run: unattended, no ops team
- Runs stably: hard limits, one agent cannot kill the device
- Right device per workload, big or small: NPU for CV, GPU-class for LLMs, one API
- Same deployment patterns: no manifest changes per device
- Fine-grained slices: carve memory, time-share when demand exceeds

DYNAMIA.AI
Creators of HAMi

Part 2: The Solution

One scheduling plane across heterogeneous accelerators

서울

DYNAMIA.AI
Creators of HAMi

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.

GPU Sharing

Dynamic fine-grained device slicing

DYNAMIA.AI
Creators of HAMi

- **NVIDIA, Ascend, Cambricon, Hygon, Iluvatar** supported
- **Fine-grained:** as small as 1MB device memory, 1% computing cores
- **Transparent to tasks:** no code changes required
- **Hard resource isolation** inside containers
- One API across vendors: same YAML, any accelerator

```yaml
# Ascend 910C: 8GB + 20% compute
resources:
  limits:
    huawei.com/Ascend910C: "1"
    huawei.com/Ascend910C-core: "20"
    huawei.com/Ascend910C-memory: "8192"

# NVIDIA: 3GB on any GPU
resources:
  limits:
    nvidia.com/gpu: 1
    nvidia.com/gpumem: 3000
```

Types of Slicing

Four ways to share one device

Memory slicing

- Carve memory per tenant
- HAMi: 1 MiB per pod
- Native: fixed partitions (MIG, vXPU)

Compute slicing

- Cap compute at a % of the device
- HAMi: 1% per pod

Time slicing

- Workloads take turns
- Pure software

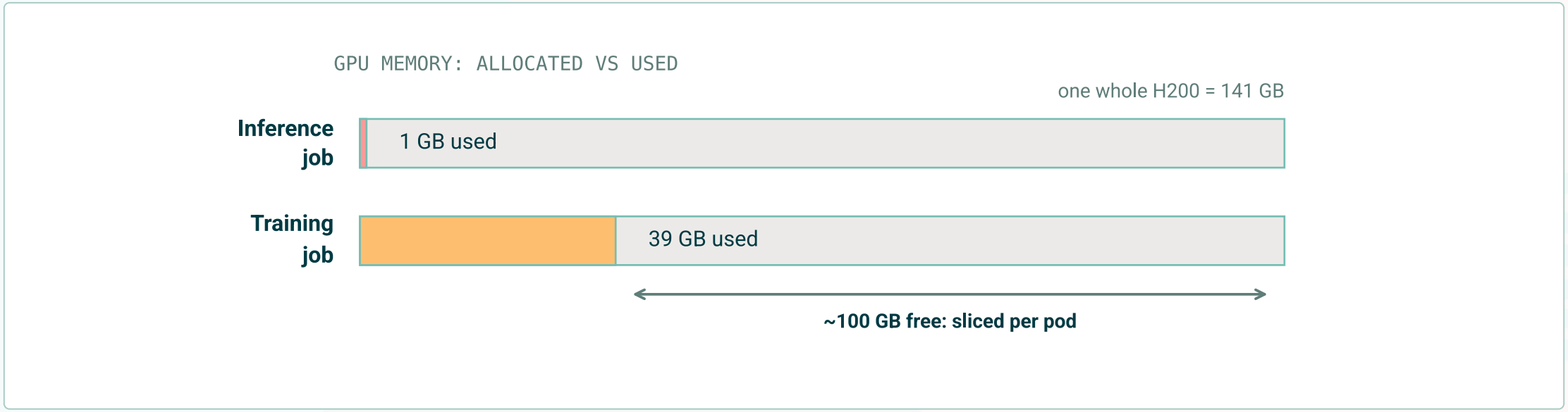
Hardware partitioning

- MIG, SR-IOV, vXPU, vDCU
- Static, strongest isolation

Time slicing is pure software: reverse engineering possible, better if the vendor SDK supports it.

Memory Slicing

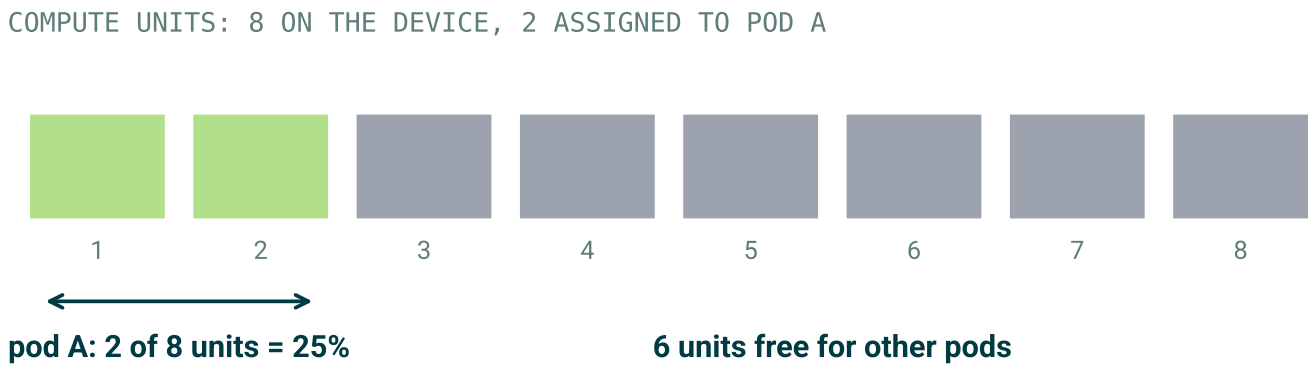
Carve unused memory into per-pod slices



- One job holds a whole card: 1 GB used of 141 GB
- Slicing frees the rest: MiB-level partitions per pod

Compute Slicing

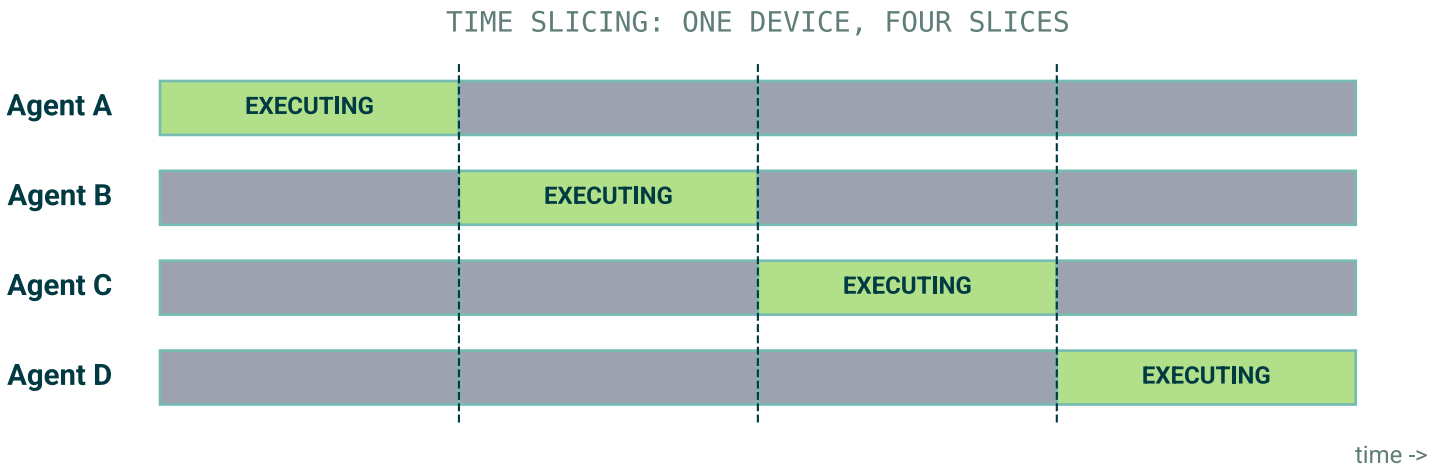
Percent of compute per pod, enforced on every call



- Each pod gets a fixed % of the device's compute units
- Hard ceiling on every kernel launch: no bursting above the slice
- Unused units stay free for other pods

Time Slicing

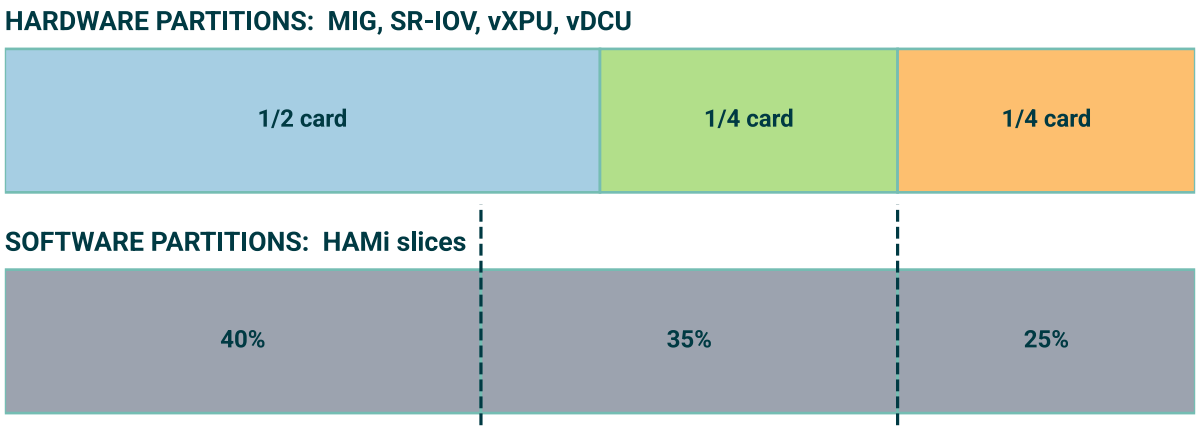
Workloads take turns at kernel boundaries



- Rotate compute access over time
- Kernel boundary is the switch point: no mid-kernel preemption
- Preemption optional: without it, long kernels run to completion

Hardware Partitioning

Fixed slots, strongest isolation, least flexible



- MIG, SR-IOV, vXPU, vDCU: hardware partitions, silicon-enforced
- Memory and compute fenced per slot: strongest isolation

Making NPUs Schedulable

What it takes to share them

- **Capacity reporting:** publish memory MiB and compute % per device
- **Allocation:** binpack and spread policies fit agents into the gaps
- **Enforcement: the hard part.** CUDA has a hijack point (the runtime library). NPU SDKs are closed compilers: no hijack, so limits live in the SDK runtime or scheduler-side
- **The clean path is DRA:** typed capacities in ResourceSlices, scheduler allocates, no code changes
- **Honest status:** HAMi shares NVIDIA, Ascend, Cambricon, Hygon, Iluvatar today. Jetson and these NPUs are the frontier, not a shipped feature

Why Jetson Has No MIG

Hardware partitioning is a data-center feature

No hardware support

MIG needs partitioning logic in the silicon. Data-center GPUs (A100, H100) have it. Orin's embedded Ampere die does not.

Never enabled in JetPack

MIG has never been enabled for Orin. JetPack 7.0 adds it only on Jetson Thor T5000, as a technology preview.

CUDA context cap

Orin is capped at ~32 concurrent CUDA contexts. Software slicing is the only path to many tenants.

MIG on Jetson: not a software toggle. The hardware is not there.

Small Models Do Not Need MIG

Edge agents fit in slices, not fixed hardware partitions

DYNAMIA.AI
Creators of HAMi

Edge agents run small models

Llama-3.2-1B/3B, Qwen2.5-0.5B to 3B, Hermes-3-3B: 1-8 GB with context. They fit in slices, not partitions.

MIG profiles are coarse

Smallest A100 slice is 1g.5gb: a fixed 5 GB partition. No dynamic repartition, no sub-slicing.

Granularity mismatch

On an 8 GB Jetson, one fixed 5 GB partition leaves nothing for anything else. MiB slices fit 10+ agents.

Dynamic, not static

MIG partitions are fixed at boot. Edge demand changes: agents come and go. Software slicing repartitions live.

MIG solves a data-center problem with fixed profiles. The edge needs dynamic, fine-grained carving.

Jetson vs NPUs: DeepX, Axelera and Furiosa

Different partitioning, different schedulability

	Jetson AGX Orin	DeepX DX-M1	Axelera Metis	Furiosa RNGD
Architecture	CUDA GPU, unified memory	Proprietary NPU	RISC-V + in-memory compute	TCP NPU, 8 PEs
Peak compute (INT8)	275 TOPS (sparse)	25 TOPS	~214 TOPS	512 TOPS
Power	15-60 W configurable	1-5 W	3.5-15 W	180 W
Perf/watt	1x baseline	~20x vs GPGPU (vendor)	~15 TOPS/W	~2.8 TOPS/W
Memory	64 GB unified LPDDR5	4 GB LPDDR5 on card	16 MB L1 + 32 MB L2 SRAM, 4-16 GB DDR	48 GB HBM3
Partitioning	None	None	Static 25% L2 per core (1-4)	SR-IOV fixed: 6/12/24/48 GB

RNGD is datacenter class, not edge: 512 TOPS at 180 W is 2.8 TOPS/W, 1.8x the A100 (624 INT8 TOPS at 400 W) and on par with H100 dense FP8 (1979 TOPS at 700 W). Value here: Korean domestic silicon, air-cooled density, the only hardware partitioning in the table. Its edge sibling Warboy (64 TOPS) has none.

DRA: ResourceSlice

Per-node device inventory

- **ResourceSlice:** lists all available devices on a node with their attributes
- DRA drivers publish slices; scheduler reads them to match claims to devices
- Tied to a node via `nodeName`, supports topology and NUMA attributes

DRA: ResourceClaim

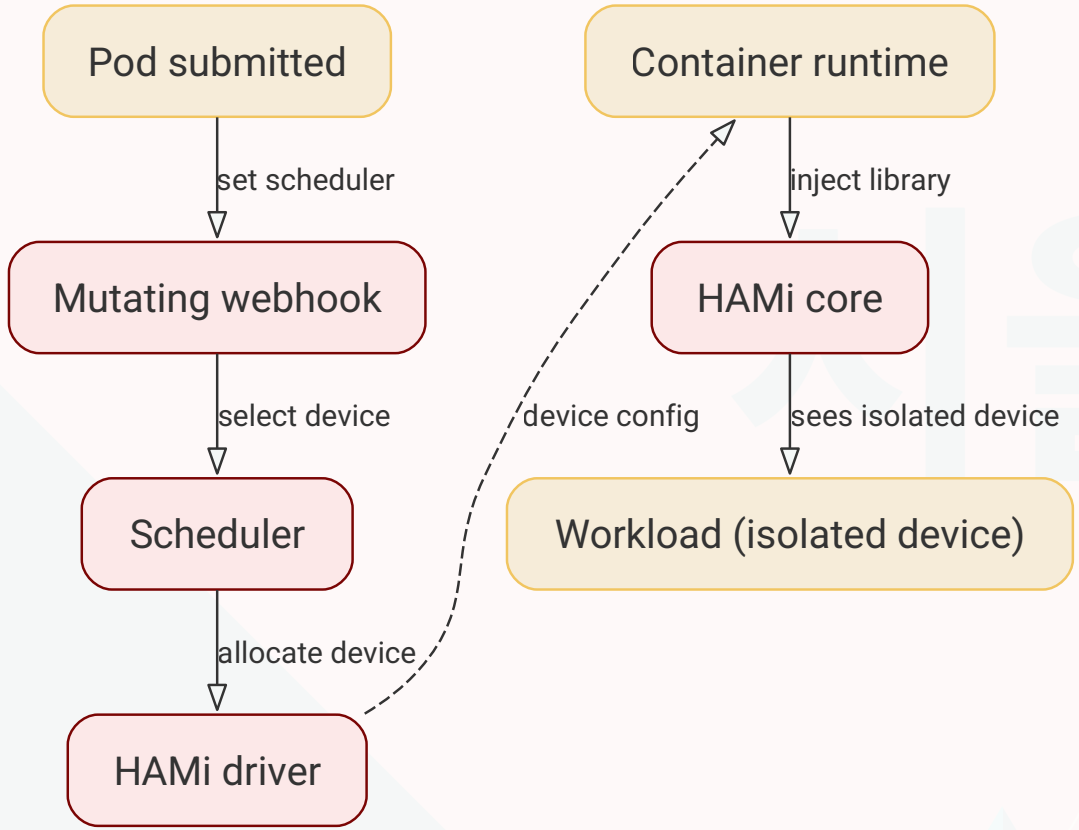
A standardized way to request hardware: not just GPUs. Stable in K8s 1.34.

- **DeviceClass:** groups devices with identical resource models. **ResourceClaim:** a workload's ticket to hardware. **ResourceClaimTemplate:** reusable blueprint, auto-creates a claim per Pod.

How HAMi Works

From pod submission to isolated device

- **Mutating webhook:** sees accelerator requests, routes pod to HAMi scheduler
- **Scheduler:** selects device and node
- **HAMi driver:** generates device config
- **Container runtime:** reads config, injects HAMi-Core library
- **HAMi core:** enforces isolation in-process

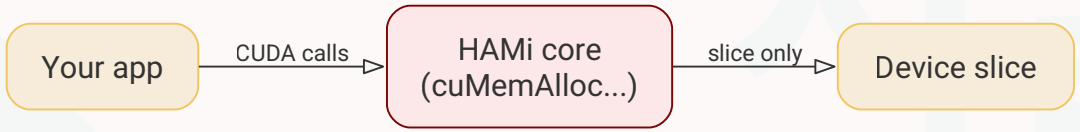


The Magic: Runtime Hijacking

Your app does not change

Your app calls CUDA. HAMi answers with a slice of the device.

- Small library loaded before your app (`LD_PRELOAD`)
- Intercepts CUDA calls, no app code changes
- No kernel modules, no driver changes
- Same pattern per vendor: ACL on Ascend, CNRT on Cambricon



Why Memory Isolation Matters

One bad agent must not kill its neighbors

Without HAMi

Tasks share a device with no borders. One greedy agent eats all memory and kills the neighbors. On a 8 GB Jetson, that is the whole device.

With HAMi

Each task sees only its slice. Memory is a hard limit, enforced by the hijacked runtime calls: every allocation is checked against your slice.

Time-sharing

Idle memory can be swapped to host RAM, so more agents fit. Great for inference, not for active training.

Per-task limits

```
resources:
  limits:
    nvidia.com/gpu: 1
    nvidia.com/gpumem: 3000
```

Consumable Capacities

Device resources as a pool you draw from

Memory and compute, separate axes

Request MiB of memory and percent of compute units. What you use is what you get, no fixed profiles.

Scheduler packs by remaining capacity

Every device reports memory and compute left. Binpack and spread policies fit requests into the gaps.

Enforced on hijacked calls

HAMi-core caps every allocation at your slice. Inside the pod, nvidia-smi shows 0 MiB / 4000 MiB.

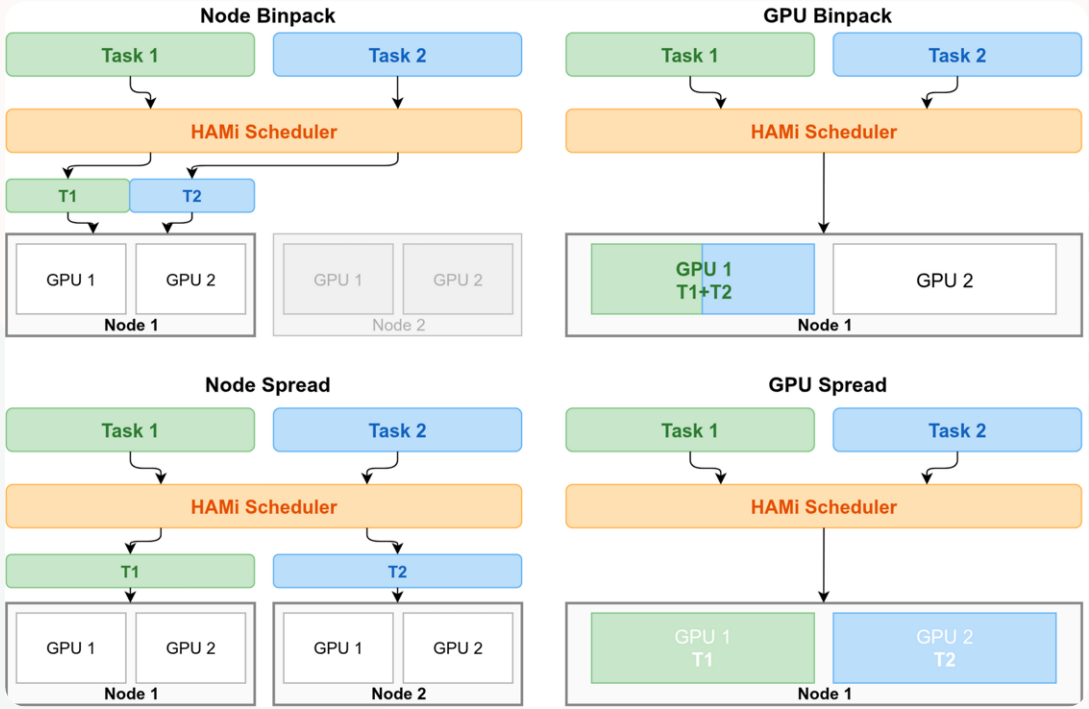
Same semantics on DRA (v2.8+)

Typed ResourceSlice capacities (memory step 1 MiB, cores 0-100). Requests become ResourceClaims.

Scheduling Policies

Binpack & Spread

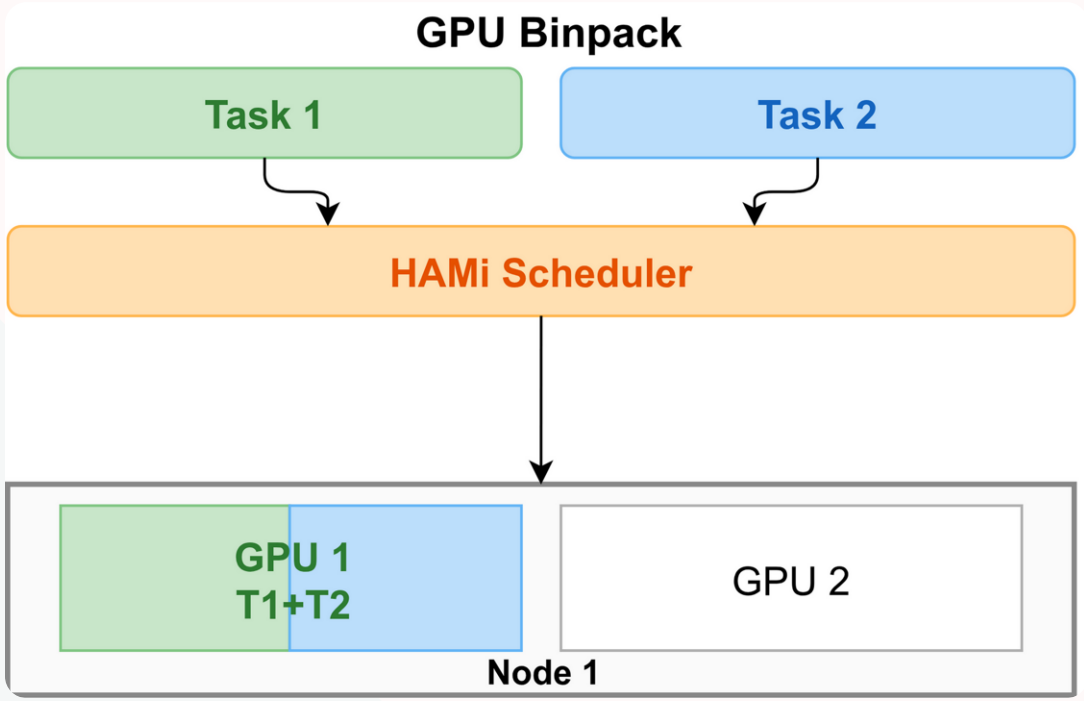
- **Node binpack** frees whole machines: fewer active nodes, less power
- **Node spread** isolates faults: HA across a device fleet
- **GPU binpack** packs many agents onto one device
- **GPU spread** protects tail latency: one agent per device
- Advanced scheduling works with standalone HAMi; DRA mode can use Yunikorn



GPU Binpack

Fill one device, run many agents

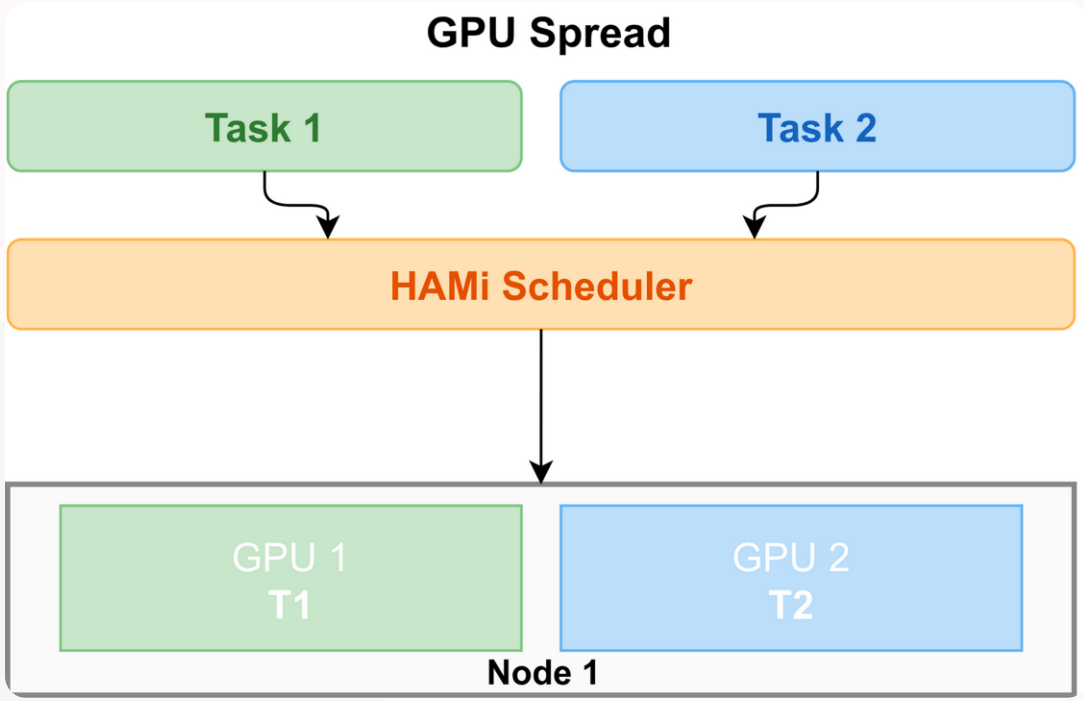
- Several agents share one device, each with its own slice of memory and compute
- Prevents fragmentation: scattered small workloads no longer block whole devices
- Frees whole devices for the job that cannot share
- Best for: fleets of small workloads - agent endpoints, batch jobs, dev and test



GPU Spread

One agent per device, no shared bottlenecks

- One agent per device: compute and memory bandwidth are never shared
- Isolates noisy neighbors: a thrasher on one device does not slow the others
- Protects tail latency: no cross-tenant contention
- Best for: latency-sensitive serving with strict SLOs



DYNAMIA.AI
Creators of HAMi

Part 3: Blueprint

Edge AI without a cloud budget

서울

The Blueprint

Three steps to a multi-agent edge device

1. Pick a device

Jetson-class GPU: CUDA compatibility, the HAMi slicing path. NPUs (DeepX, Axelera) are the frontier: no SDK hook, no slicing today.

2. Slice it

Carve memory and compute into per-agent slices. Hard limits, live repartition, time-sharing when demand exceeds memory.

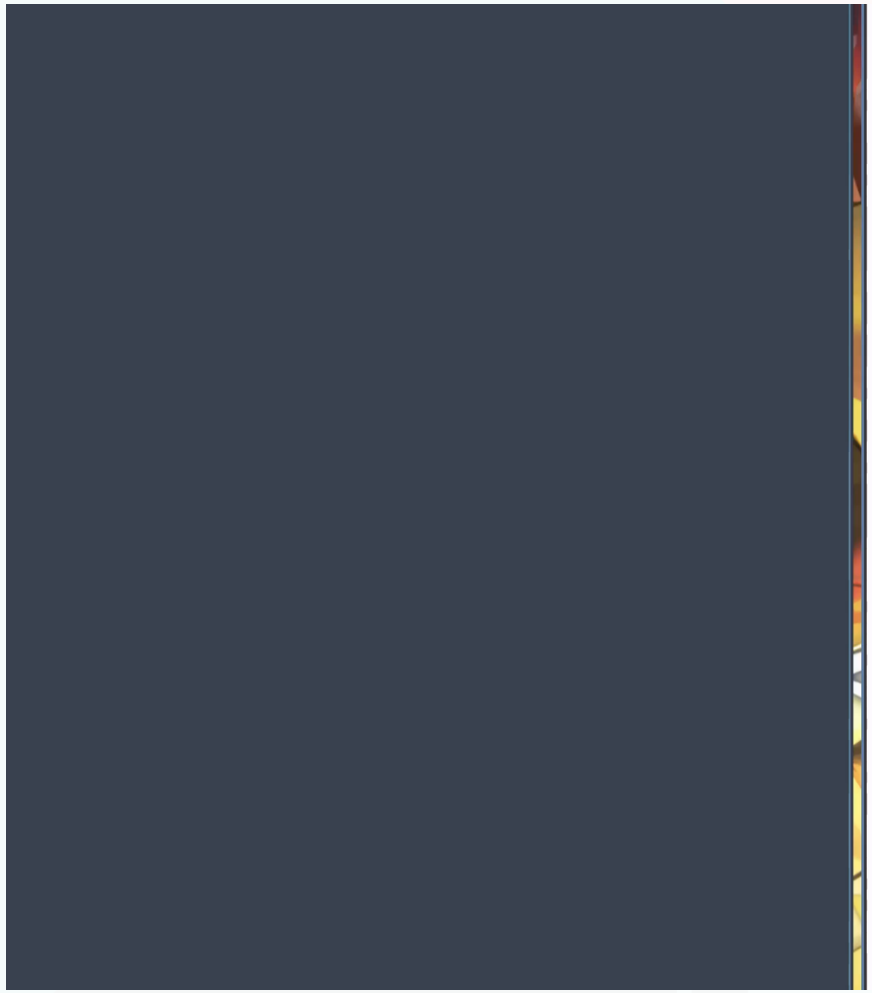
3. Schedule agents

Binpack to pack many agents onto one device. Spread for latency-sensitive serving. Node spread across a device fleet for HA.

- **Deploy on Olares** (github.com/beclab/olares): a k3s-based personal cloud OS that ships this stack pre-installed - agents as containers, MCP built in, local GPU scheduling
- Runs on k3s or k0s directly: Kubernetes APIs without the ops team
- One scheduling plane across heterogeneous accelerators
- Open source: HAMi is a CNCF Incubation project

Demo

3 nodes x 2 A100s: MIG, YOLO, and two vLLMs



Where HAMi Is Today

Production metrics from CNCF case studies

DYNAMIA.AI
Creators of HAMi

0	**1-2 GB**	**4x**	**90%**
Driver, kernel, code changes needed SF Technology	Memory per small-model inference service KE Holdings	GPU utilization: 5-10% to 30-50% NIO	GPU infra managed by HAMi on RTX 4070/4090 Prep Edu

China Merchants Bank - SNOW Corp. - NIO - KE Holdings - DaoCloud - SF Technology - Prep Education - cncf.io/case-studies

DYNAMIA.AI
Creators of HAMi

Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Backed, Production Ready

5.2k	325k	500+	27
Github Stars	Docker Pulls	Contributors	Contributor Countries

kubernetes VOLCANO Kueue koordinator KAI Scheduler COZYSTACK

Ecosystem & Device Support

NVIDIA Ascend Cambricon 寒武纪 HYGON 海光 天数智芯 Iluvatar CoreX METAX 沐曦 摩尔线程 MOORE THREADS 昆仑芯 KUNLUNXIN Enflame 燧原科技 aws 瀚博半导体

Adopters

4Paradigm 百度智能云 贝壳 招商银行 中国移动 China Mobile 中国联通 China unicom DaoCloud DYNAMIA H3C HUAWEI LinkedIn 马上消费 NIO PPIO PREP SAP SF TECHNOLOGY 顺丰科技 思特奇 SI-TECH SNOW viettel

DYNAMIA.AI
Creators of HAMi

THANK YOU

Questions? Try HAMi

github.com/Project-HAMi/HAMi

Got devices we do not support yet? We would love to play with them.

Reza Jelveh

Solution Architect, Dynamia AI - Makers of HAMi

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