

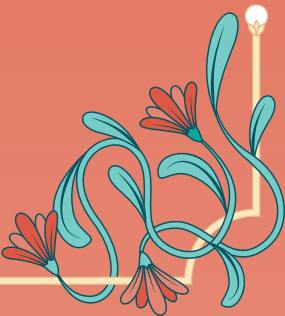
Designing Permissioned AI Agents That Can Run Offline

Layer the limits; keep the decisive ones out of the agent's reach

Reza Jelveh

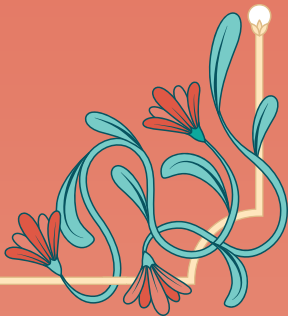
Solution Architect, Dynamia AI - Makers of HAMi

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



Part 1: The Boundary Problem

Agents read files and call tools



Agents Left the Chat Window

Read files

Mail, documents, code, config. Whatever the process can open.

Call tools

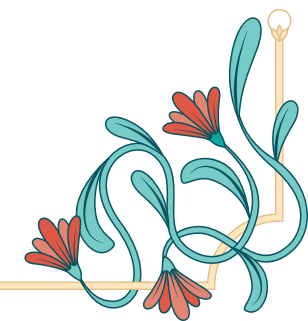
MCP servers, shell commands, REST APIs.

Change state

Send, delete, tag, deploy. Some of it cannot be undone.

Coordinate services

One agent drives several services, each with its own credentials.



Where Cloud-Hosted Agents Break

Private data leaves the device

Every prompt carries context to someone else's server.

Broad API keys

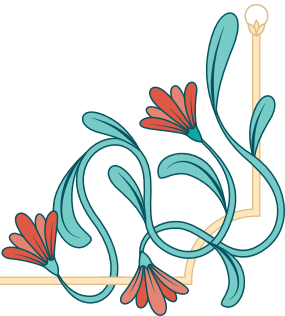
One token authorizes every operation the API has.

Little visibility

The user cannot see what the agent is allowed to do.

No network, no agent

Nothing works when the connection drops or is cut on purpose.



MCP Plugins: Do Not Trust the Tool

You do not know if it wipes your mail

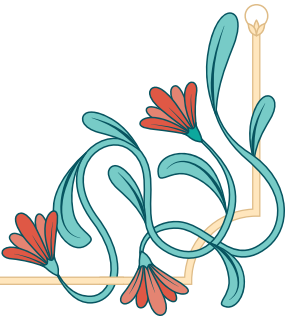
Unknown destructive power

- ▶ OpenClaw started deleting a Meta AI director's inbox, ignoring her confirm-first rule
- ▶ Your MCP tool may expose the same delete: you do not know

You build the harness

- ▶ LLM tries to break your system
- ▶ You hand-roll guards: staging, sandboxing, review

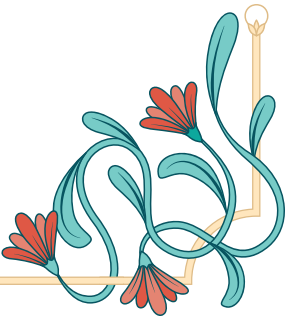
You can audit a plugin, but a defensive security posture is better.



Assume Every Tool Is Hostile

If it can reach the config, it will change it

- ▶ Every tool is risky by default, including the ones you wrote.
- ▶ An agent that can open its config file will find a way to rewrite its own limits.
- ▶ Build harnesses around agents, but assume they will break.
- ▶ Tools that enforce limits help, but you cannot count on every tool doing it.
- ▶ So layer it: in the agent, in the tool where you can, and **around the tool** always.



Threat Model

Who we defend against, and what we protect

Adversary

- ▶ The agent running rogue: a wrong assumption, prompt injection or poisoned weights
- ▶ Whatever the cause, it runs any code inside its sandbox
- ▶ A model server taken over through its API

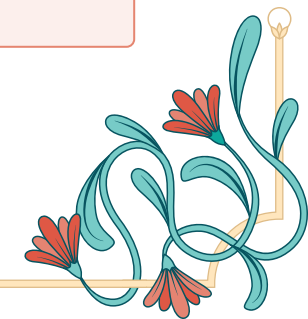
Assets

- ▶ Workspace, secrets and the grant
- ▶ Model weights, other tenants' prompts and KV cache
- ▶ The node and the cluster

Out of scope

- ▶ A malicious cluster admin
- ▶ Hardware and firmware attacks

The threat model also tells you what to skip: agent code never needs the GPU, so agent sandboxes need no GPU passthrough.



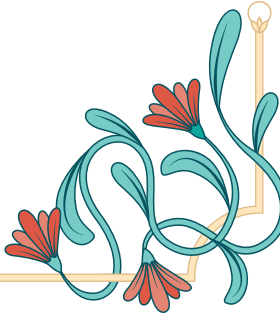
Isolation Is Not All or Nothing

Responsibility decides the boundary, not Kata everywhere

Component	Its job	If it goes rogue	Boundary
Model	Proposes text	Bad proposals, nothing runs	Gateway; no tools, no egress
Harness, supervisor	Decides, holds keys	Must stay trusted	Out of the agent's reach
Tool sandbox	Runs agent code	Anything inside the box	Per-command policy, or a VM
Tool server	Talks to mail, APIs	Its own credential scope	Narrow keys, own identity

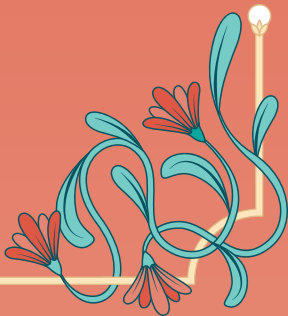
"Isolation draws the boundary. Policy decides what happens inside it." Cong Wang, Sandlock, OSS Europe 2026

Source: *agent-sandbox*; *OpenShell architecture*; *Sandlock*



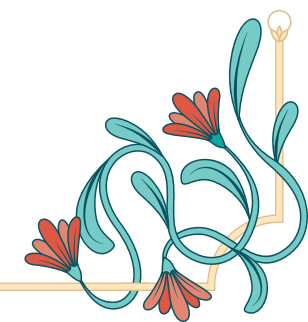
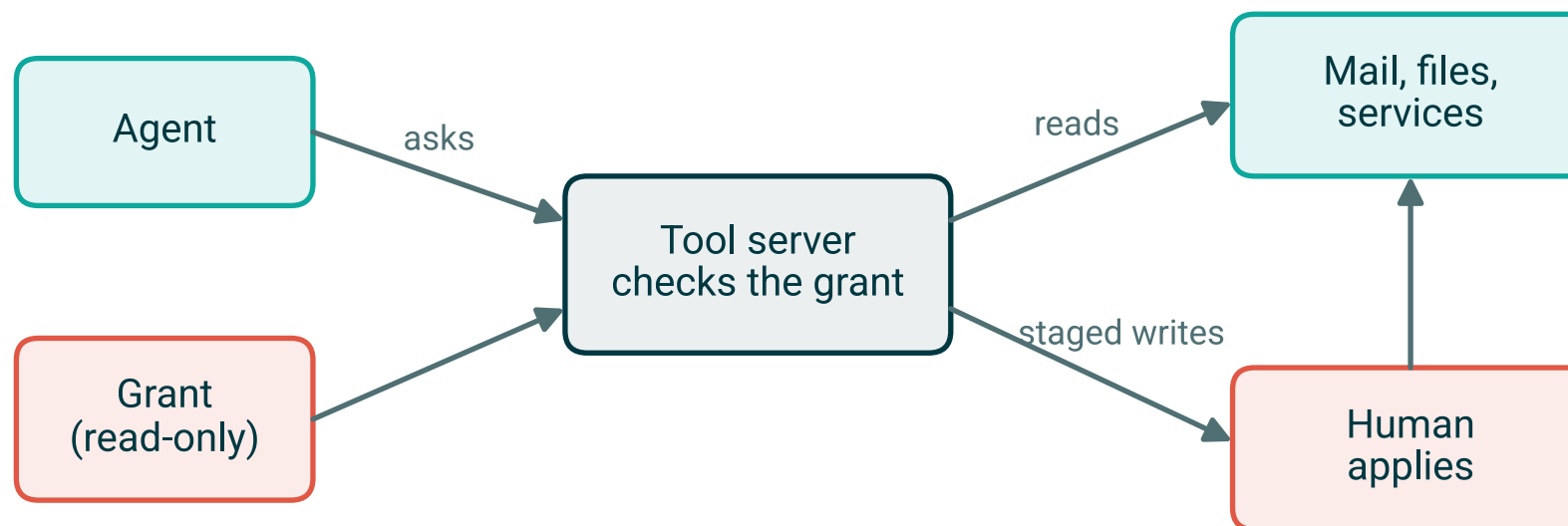
Part 2: Layers the Agent Cannot Reach

Deny by default, then defend around the tool



Separate Reasoning From Action

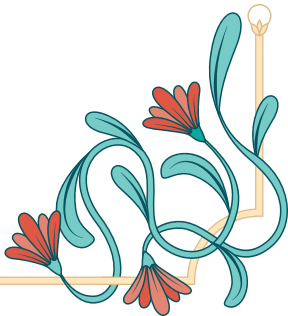
The model proposes, the tool server decides



Deny by Default

The tool serves nothing until you grant it

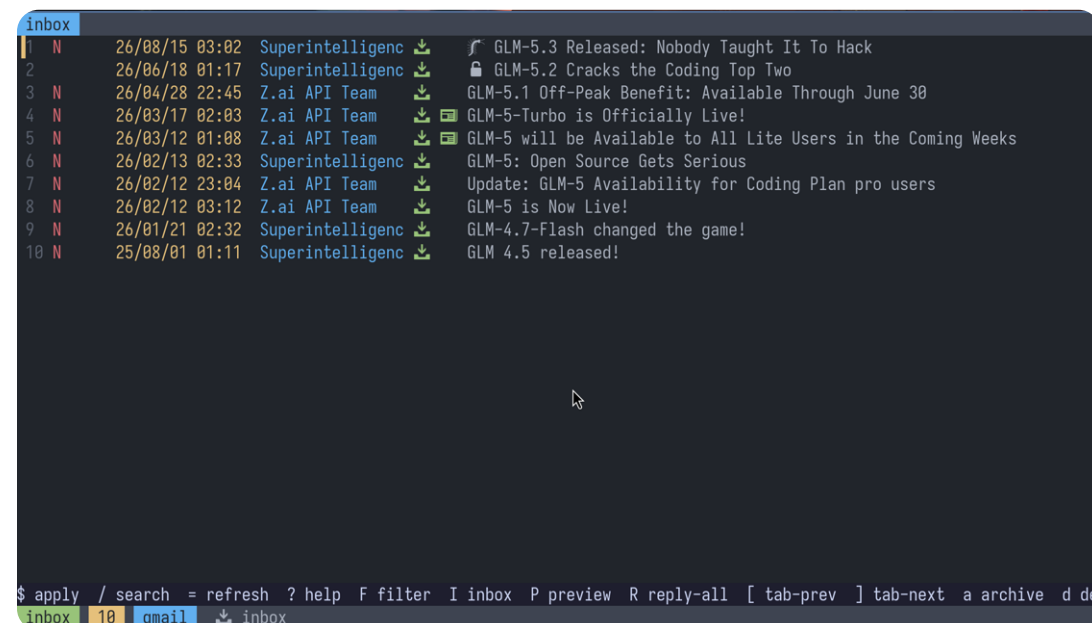
- ▶ Start from an empty grant: nothing is visible until you name it.
- ▶ Never load data the tool does not need; then it cannot leak.
- ▶ A misspelled grant stops the server instead of silently widening it.



One MCP Server Built This Way

notmutt, a terminal mail client

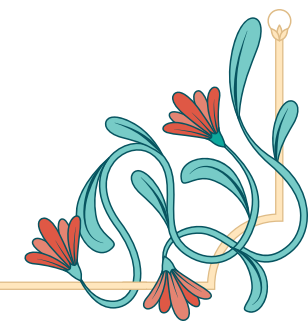
- ▶ Nothing is visible until the config grants it.
- ▶ The agent can stage changes; only I apply them.



```

inbox
1 N 26/08/15 03:02 Superintelligence GLM-5.3 Released: Nobody Taught It To Hack
2 26/06/18 01:17 Superintelligence GLM-5.2 Cracks the Coding Top Two
3 N 26/04/28 22:45 Z.ai API Team GLM-5.1 Off-Peak Benefit: Available Through June 30
4 N 26/03/17 02:03 Z.ai API Team GLM-5-Turbo is Officially Live!
5 N 26/03/12 01:08 Z.ai API Team GLM-5 will be Available to All Lite Users in the Coming Weeks
6 N 26/02/13 02:33 Superintelligence GLM-5: Open Source Gets Serious
7 N 26/02/12 23:04 Z.ai API Team Update: GLM-5 Availability for Coding Plan pro users
8 N 26/02/12 03:12 Z.ai API Team GLM-5 is Now Live!
9 N 26/01/21 02:32 Superintelligence GLM-4.7-Flash changed the game!
10 N 25/08/01 01:11 Superintelligence GLM 4.5 released!

$ apply / search = refresh ? help F filter I inbox P preview R reply-all [ tab-prev ] tab-next a archive d de
inbox 10 gmail inbox
  
```



Keep the Grant Out of Reach

The agent must not be able to edit its own limits

Config owned by someone else

Another user, a read-only mount, or a separate container. Never a file the agent can write.

No shared disk

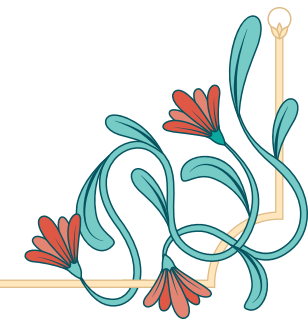
An agent that can read the maildir does not need the MCP server.

Keys stay outside

A host-side proxy swaps a placeholder for the real token. The agent never holds the key.

Separate identities

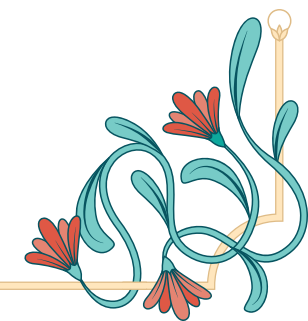
Model, tool server and destructive tools each run with their own account and permissions.



Assume the MCP Server Breaks Too

Defensive design around the tool

- ▶ The server is code: bugs, a bad update, or a tool you did not write.
- ▶ Give it the narrowest credentials; read-only where reads are enough.
- ▶ Run destructive tools in a VM or a separate account, with a backup first.
- ▶ Keep writes reversible: stage, review, apply, and keep the undo.
- ▶ Cap how much one call may touch: a hundred messages, not the mailbox.



Agent Sandboxes Already Do This

Three isolation levels, the same rules

Process

Landlock, seccomp, bubblewrap. Starts in milliseconds, cheap enough per command. Sandlock, srt (Claude Code's sandbox), nono, ai-jail.

Container

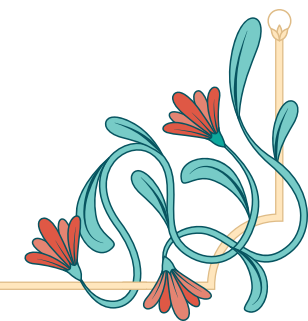
Docker or rootless Podman; dropped capabilities and egress proxy are opt-in. yolobox: protection from accidents, not container escapes.

MicroVM

Firecracker or libkrun. Vendors claim sub-second boot. smolvm, microsandbox, matchlock.

- ▶ Keys stay on the host; a proxy injects them per request (matchlock, nono, drydock).
- ▶ The stricter tools deny egress by default, with an allowlist (srt, smolvm, drydock).
- ▶ Only a diff leaves the sandbox, after you approve it (drydock).

Source: *awesome-agent-sandbox*; *srt*; *yolobox*; *matchlock*; *drydock*

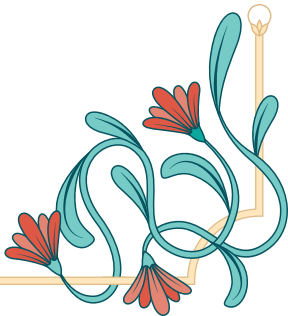


Agent Sandboxes on Kubernetes

Three layers, each one can be missing

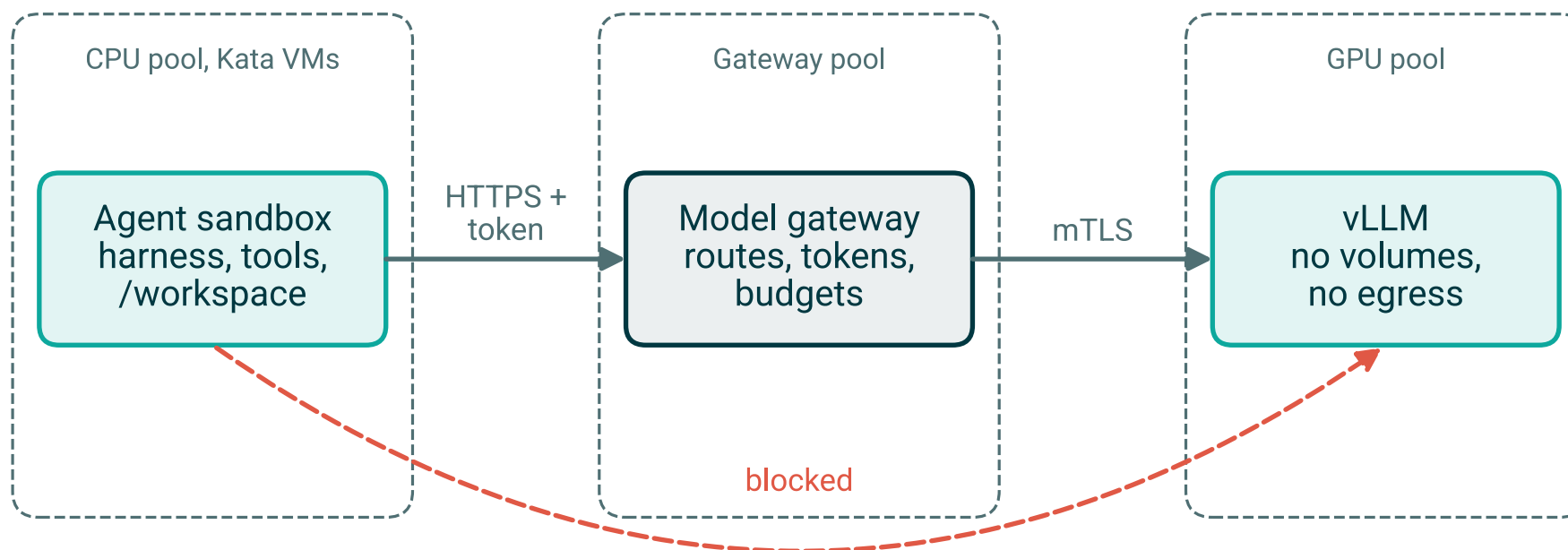
- ▶ **Runtime isolation:** gVisor or Kata, picked by RuntimeClass.
- ▶ **Orchestration:** kubernetes-sigs/agent-sandbox v1.0.x: Sandbox, SandboxTemplate, SandboxWarmPool.
Default egress: public internet.
- ▶ **Policy:** OpenShell (default-deny egress, credentials injected at the proxy), agentgateway (tool authorization).
- ▶ A NetworkPolicy on a CNI that ignores it is decoration.
- ▶ Kelos runs Claude Code with `--dangerously-skip-permissions` in ordinary pods.

Source: *agent-sandbox threat model; OpenShell runtimes; agentgateway; Kelos endpoint*



Keep Inference Away From the Code

The model needs text, not your filesystem



Segment the Model Network

The sandbox reaches the gateway, and nothing else

Subnets and a firewall

One subnet per node pool. The firewall sits outside the cluster, so it holds even if a sandbox owns its node.

A VM per sandbox

Kata on separate nodes. A container escape does not land on a node that can reach the GPUs.

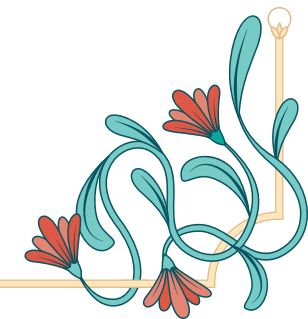
Default-deny NetworkPolicy

Sandboxes may only call the gateway; vLLM only accepts the gateway. Needs a CNI that enforces it.

Model gateway

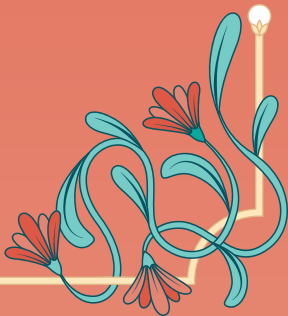
Inference routes only, a token per sandbox, token budgets, mTLS to vLLM. Admin endpoints stay unreachable.

Close the side doors too: no service account token, no cloud metadata endpoint, no kubelet, no vLLM multi-node ports.



Part 3: Least Privilege for Compute

One device, several agents



Edge Agents, Starved Compute

Offline means the model runs on hardware you own

Limited memory

A Jetson has 4-128 GB of unified memory, shared with the OS. One agent stack can eat it all.

Tight power budgets

No 1200 W data-center GPU at the edge. You get 50-500 W: Strix Halo, DGX Spark, an RTX 5090.

Unattended

No cluster admin on call. It has to work after setup.

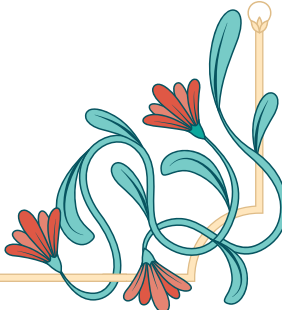
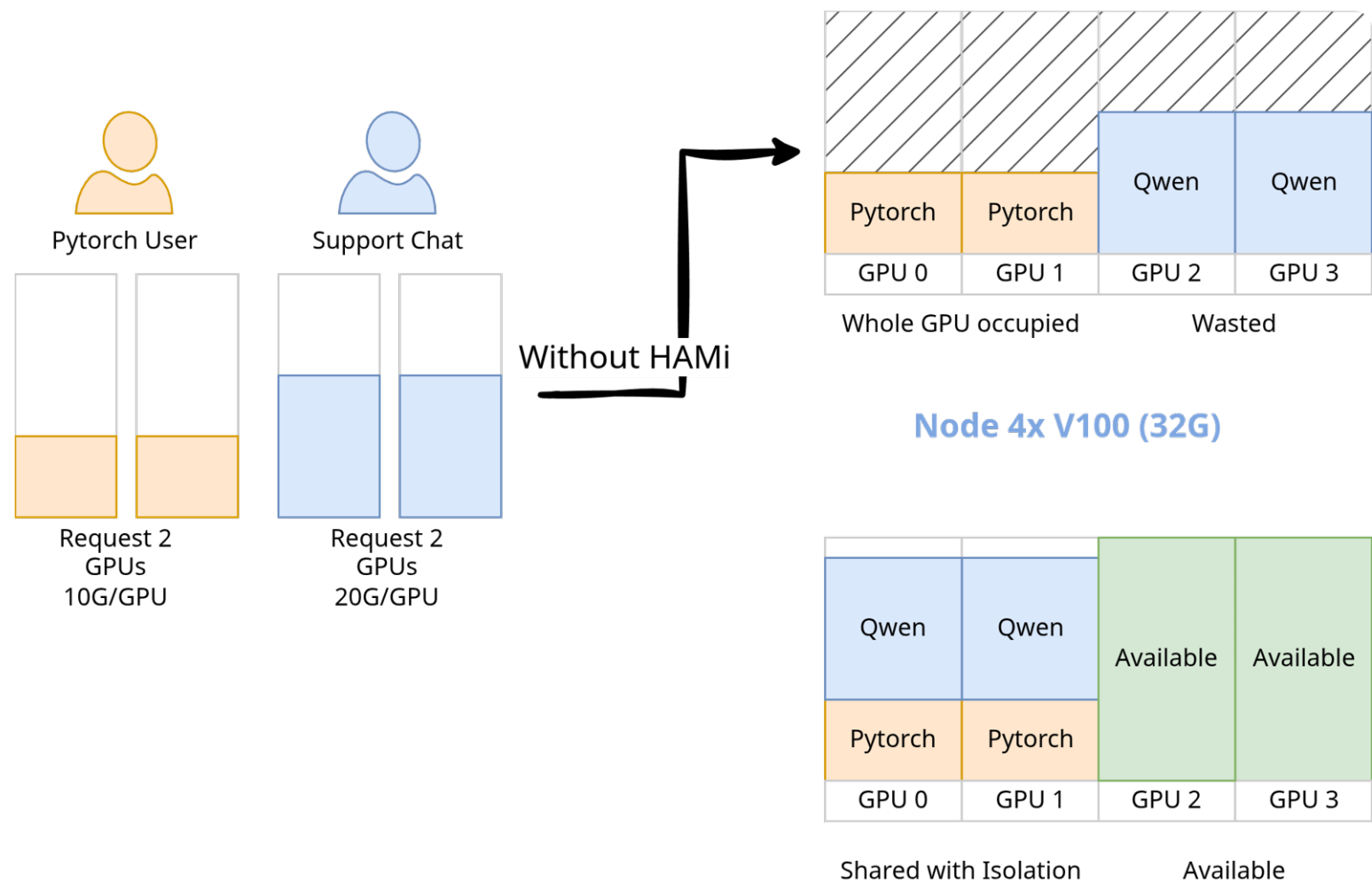
No elastic scaling

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



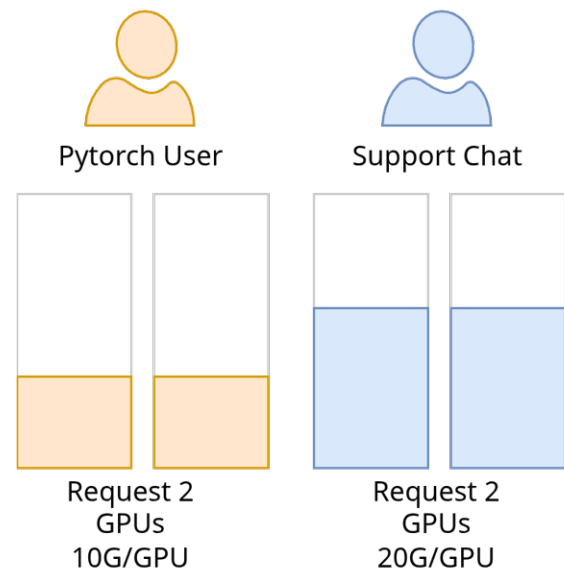
What is HAMi

Before: one device, one task

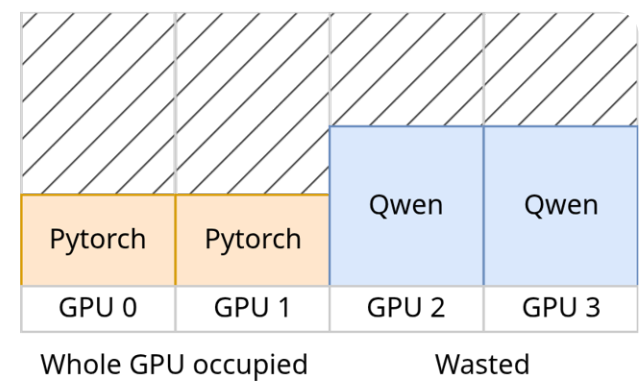


What is HAMi

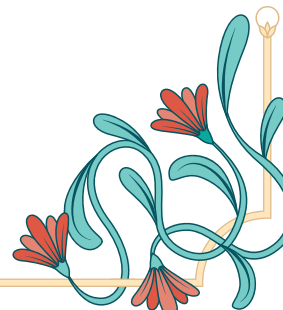
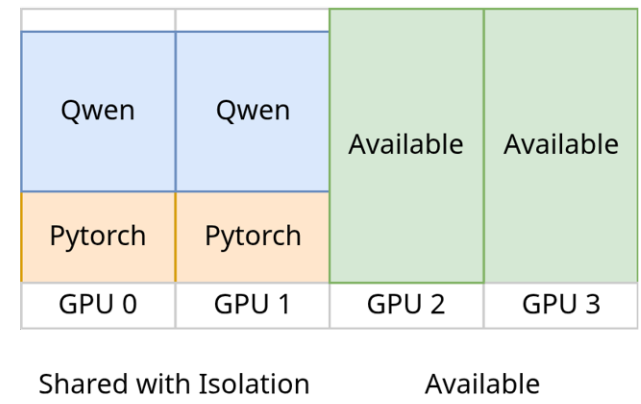
After: one device, many agents



With HAMi



Node 4x V100 (32G)



Least Privilege for the GPU

One greedy agent must not starve its neighbors

⚠ Without HAMi

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

✓ With HAMi

Each agent sees only its slice. Every allocation is checked against it.

🕒 Per-agent limits

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

Footnote: destructive tools are safer in a VM, since escaping a container is easier. GPUs in VMs are possible but costly; agent sandboxes do not need them.



Share a GPU Only Within One Trust Domain

Slicing protects availability, not secrets

Workload	GPU	With HAMi
Agent sandboxes	None	No device at all
Your model servers, same data class	Shared slices	gpumem, gpucore, binpack
Different tenants or data classes	MIG or a whole GPU	vgpu-mode: mig, use-gpuuid, spread
Strictest	Separate nodes	Separate node pools

Even MIG leaves side channels. vGPUmonitor shows who shared which GPU, as evidence, not prevention.



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.

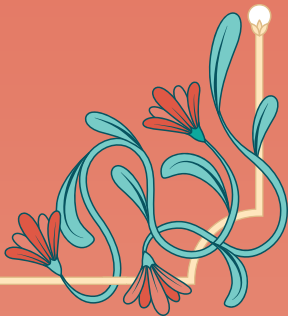
Unified Monitoring

Consistent metrics and visibility across vendors.



Part 4: Local and Offline

Run it on hardware you own



Keep Sensitive Context Local

Three steps to a multi-agent edge device

1. Pick a device

Jetson-class GPU: CUDA compatible, so HAMi can slice it.

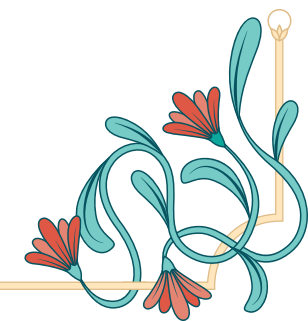
2. Slice it

Memory and compute per agent, with hard limits.

3. Schedule agents

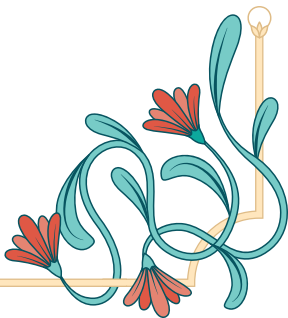
Binpack many agents onto one device; spread when latency matters.

- ▶ Prompts, documents and mail never leave the device.
- ▶ **Olares** (github.com/beclab/olares): open-source personal cloud OS on K3s, ships a HAMi fork for GPU slicing.
- ▶ Or run on k3s or k0s directly: Kubernetes APIs without an ops team.



Takeaways

- ▶ Assume every tool is hostile; layer limits in the agent, the tool and around both.
- ▶ Let responsibility decide each boundary: not everything needs a VM.
- ▶ Separate reasoning from action: the model proposes, the tool server decides.
- ▶ Grant nothing by default; keep the grant out of the agent's reach.
- ▶ Keep inference and tools on separate nodes; the sandbox reaches only the model gateway.
- ▶ Give each agent its own compute slice, on hardware you own.



Community & Adopters

Devices, integrations, and who uses HAMi

Open Source, CNCF Backed, Production Ready

5.2k

Github Stars

325k

Docker Pulls

500+

Contributors

27

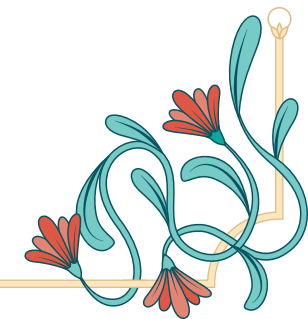
Contributor Countries



Ecosystem & Device Support



Adopters



THANK YOU

Questions?

github.com/Project-HAMi/HAMi - github.com/fishman/notmutt

Reza Jelveh

Solution Architect, Dynamia AI - Makers of HAMi

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

