

CLOUD NATIVE AI + INFERENCE DAY

NORTH AMERICA

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Prefill here, decode there
Kubernetes-native LLM inference with KAITO and llm-d

Andy Zhang — Microsoft
Linbo He — Microsoft

About us



Andy Zhang

Principal Software Engineer, Microsoft AKS team
I maintain KAITO and CSI drivers.
GitHub: andyzhangx



Linbo He

Principal Software Engineer, Microsoft AKS team
I maintain KAITO and OpenYurt.
GitHub: rambohe-ch

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Agenda

We'll go from inference basics to running P/D on Kubernetes.

01 **Inference basics: prefill and decode**

See how each phase affects latency.

02 **Why P/D disaggregation**

Weigh the benefits against the costs.

03 **llm-d and KAITO overview**

See what each project owns.

04 **How KAITO + llm-d supports P/D**

Follow routing and KV transfer.

05 **Per-role autoscaling for P/D**

Scale each role with its own signals.

06 **What's next**

What is the LLM KV cache?

Stored attention keys (K) and values (V) for tokens already processed.



Reuse computation, not the answer

Skip recomputing earlier tokens' K/V.
The current token's query still attends
over cached keys and values.

More context and requests, more memory

Cache grows with sequence length and
concurrent requests. It is request state,
not model weights or an answer cache.

P/D disaggregation transfers the KV cache from the prefill instance to the decode instance.

Prefill: process the prompt and select the first token

Prompt: “What does Kubernetes do?”

Teaching example / Words + punctuation(not a real tokenizer’s output)

What	does	Kubernetes	do	?

Prefill process prompt positions together; each sees only itself and earlier positions.

5 prompt positions

4 words + punctuation

→

Causal prefill

No position sees future tokens.

→

Select “Kubernetes”

Use the final position’s scores.

↓

prefill cache attention keys and values for all 5 prompt positions at every layer.

Prefill use the final position’s scores to select “Kubernetes”. Decode will consume it and cache its K/V.

Decode: consume a token, select the next, repeat

Illustrative output: “Kubernetes helps teams deploy and scale applications.”

Illustrative word-level tokens; not a real model tokenizer.

Step	Consume + read past KV	Append to per-layer KV	Select next output
1	Kubernetes	K/V for “Kubernetes”	helps
2	helps	K/V for “helps”	teams
3	teams	K/V for “teams”	deploy
4	deploy	K/V for “deploy”	and
5	and	K/V for “and”	scale
6	scale	K/V for “scale”	applications

Each selected token becomes the next input. Repeat until the answer is complete.

Client arrivals / ITL = gaps / First 6 words shown / Not to scale



We cache the token we consume, not the one we just select. We reuse past KV.

One request. Two latency targets.

We want a short wait for the first token, and short gaps between the tokens that follow



Prefill → TTFT

How soon does the answer start?

Prefill processes the input and is often **compute-bound.**

Decode → ITL / TPOT

Does the answer keep flowing smoothly?

Decode generates tokens and is often **memory-bandwidth-bound.**

TTFT also includes delivery; ITL is not just decode time. TPOT = avg. time per output token. Sources: vLLM; NVIDIA Dynamo.

Our goal: serve more requests while meeting both latency targets.

SLO goodput = requests per second that meet both targets.

Long prefills can interrupt token delivery

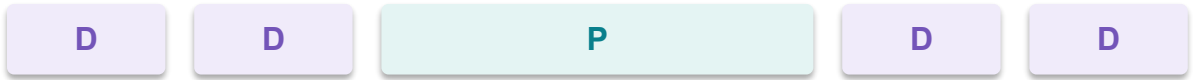
D is ongoing decode. P is prefill for a new request. Where should new prefill request run?

Conceptual schedule · Not a benchmark · Block widths and gaps are not timings

Shared GPU

One worker does both P & D

Prefill can pause decode.



Separate GPUs

Decode can continue while prefill runs elsewhere.

Prefill worker



Requests need a KV handoff.

Decode worker



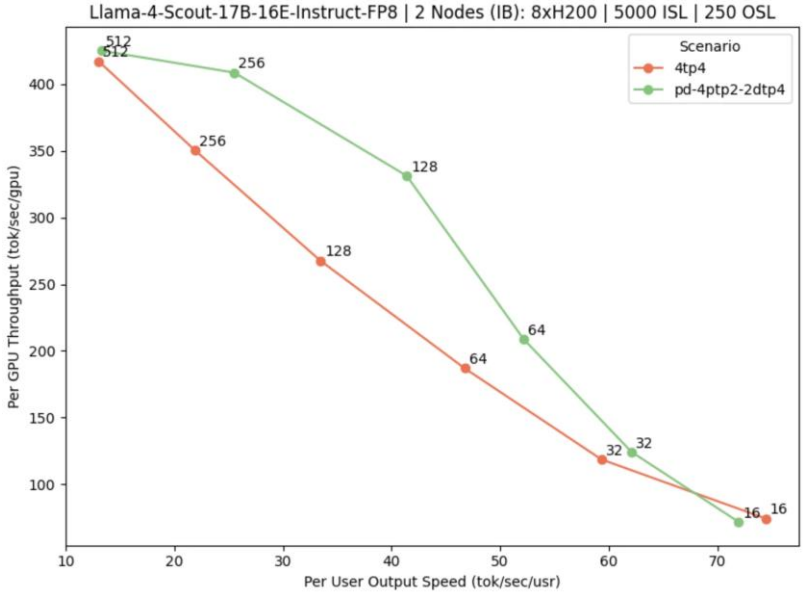
D = decode step; P = prefill for a new request. Chunked prefill can reduce shared-GPU pauses.

Original conceptual illustration; sources: NVIDIA and vLLM (2025) + current documentation.

Separate GPUs remove this execution conflict, but not every source of latency since it requires KV handoff between nodes.

P/D improves throughput with llm-d

Llama 4 Scout · 8*H200 GPUs · on 2 Nodes



X: output speed per user · Y: throughput per GPU · Point labels: concurrent requests

Higher throughput in the middle

Largest benefit around 64–128 concurrent requests in this benchmark.

Without P/D: 4 × TP4

4 mixed workers · 16 GPUs

With P/D: 4P × TP2 + 2D × TP4

8 prefill GPUs + 8 decode GPUs

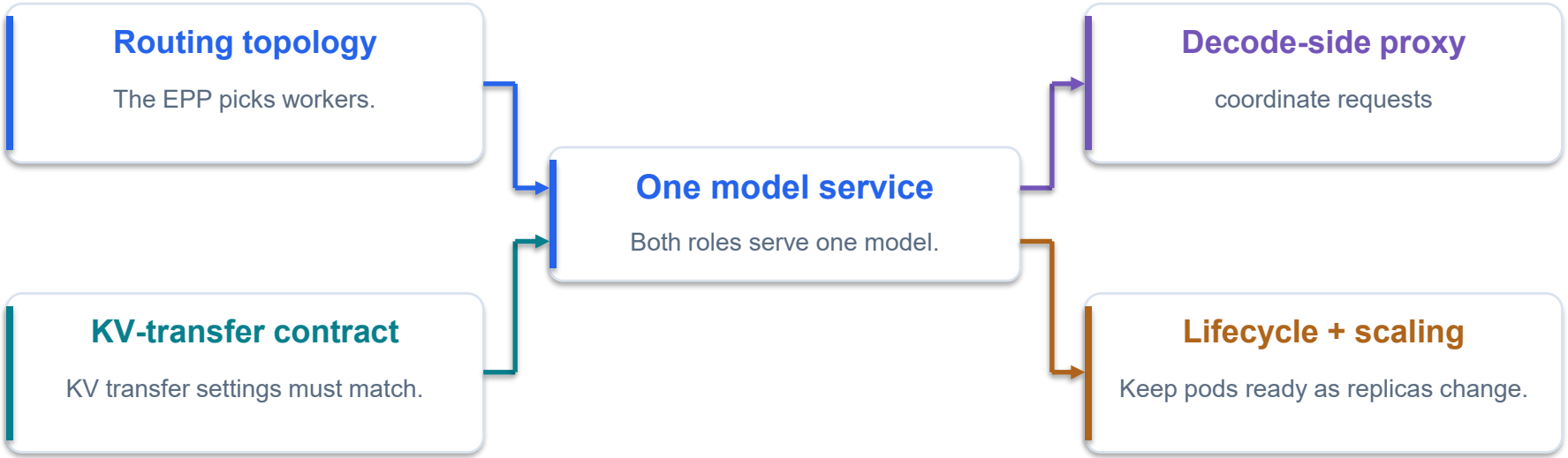
5,000 input / 250 output tokens · FP8

More throughput from the same GPUs, depending on workload.

Source: llm-d v0.2, Jul 29, 2025, Fig. 1. Official chart; external benchmark, not a KAITO result.

The hard part is the Kubernetes wiring to support P/D

Two pools still have to work together as one model service.



Routing, KV transfer, labels, ports and readiness must stay in sync, even when pods restart or we scale.

Production Distributed Inference Architecture

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Application Layer

Web / H5 | Mobile Apps | Enterprise Applications | Third-party Applications | …

Inference Layer

AI Gateway (Business Logic) (AgentGateway , Higress)

Orchestration Layer (LLM-D, KAITO, Dynamo)

LLM-D KAITO

Service Orchestration | Routing & Load Balancing | Scheduling & Autoscaling | Policy & Strategy Management | Elastic Scaling | Observability & Monitoring

Inference Runtime Layer (vLLM, SGLang, Triton)

vLLM

Continuous Batching | Paged Attention | Speculative Decoding | Multi-LoRA | Prefix Cache | …

KV-Cache Offload & Management Layer (LMCache, KVBM, Mooncake)

KV Cache Management	Storage Abstraction (NIXL)	Distributed Memory KV Transport	Storage Backend	DPU Storage Gateway
KVBM LMCache Mooncake …	Unified Interface Tiered Storage Mgmt. Cache Policy Data Consistency	Redhare (Distributed Memory KV Transport) ContextStor (Large Context Storage)	Object / File Storage High-performance Storage Cloud / On-prem Storage	DPU Acceleration Protocol Offload · Data Optimization Security Isolation · High Performance

Kubernetes GPU & Data Scheduling

Model Hub (MatrixHub) | Model Distribution (ModelExpress) | Dataset Management | Data Catalog | Data Processing | Data Quality

Resource Pool Management | Priority & Preemption | Fair Scheduling | Elastic Resource Allocation | Multi-tenant Isolation | Workflow Orchestration

Kubernetes | GPU Operator | Device Plugin | GPU Monitoring | Topology Manager | Node Auto-discovery | Multi-tenant Isolation

Infrastructure Operations

Compute (VM / Bare Metal) | Network (VPC / SDN) | Storage (Distributed) | Security (FW / IDS) | Monitoring (Prometheus) | Logging (ELK) | Automation (Ansible) | CI/CD (GitOps)

MaaS Operations (Token as a Service)

Gateway & Security

- Unified API Gateway (OpenAI Compatible)
- Safety & Content Filtering
- Rate Limiting & Access Control
- WAF / DDoS / Shield

Metering & Billing

- Usage Metering (Tokens / Requests)
- Billing & Invoicing
- Usage Analytics & Reporting

Operations Management

- SKU & Model Management
- Pricing & Policy Management
- Coupon & Promotion
- Quota & Share Management
- Cost Allocation & Chargeback
- FinOps & Cost Optimization

Serving Targets

Cloud | Enterprise

Several open-source projects support P/D disaggregation

Examples, not a ranking. Some projects work together.

Approach	Serving / routing path	Kubernetes packaging
NVIDIA Dynamo	Integrated inference stack	Operator + DynamoGraphDeployment
llm-d [CNCF]	P/D-aware EPP + routing sidecar	Gateway API / InferencePool
AIBrix	P/D-aware gateway + engines	StormService / RoleSet
KServe + llm-d [CNCF]	llm-d serving integration	LLMInferenceService
Ray Serve + KubeRay	P/D app + vLLM KV connectors	RayService + Serve configuration
vLLM Production Stack	vLLM + LMCache / NIXL + router	Helm-managed P/D deployment

CNCF: llm-d — Sandbox; KServe — Incubating. Marked entries are CNCF-hosted projects.

KAITO's goal: make these options easier to deploy and manage, with less manual configuration

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

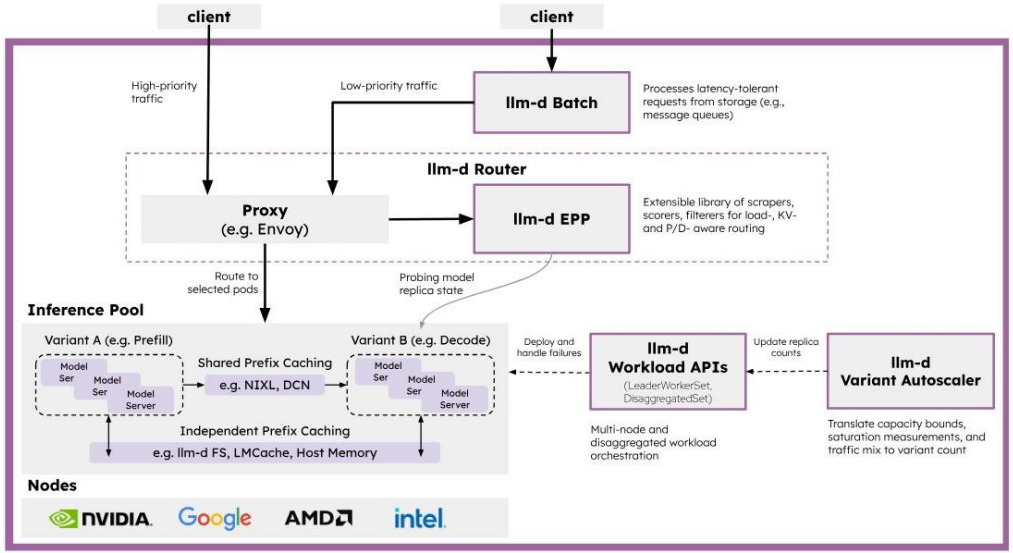
llm-d: framework for running distributed inference on k8s

CNCF Sandbox project

Brings inference runtimes, routing, and KV cache transport together on Kubernetes.

Open integrations

- Gateway API Inference Extension
- NIXL
- KV storage providers



How the pieces fit together

Modular design

Connect inference routing to Kubernetes through Gateway API.

Flexible components

Combine runtimes, routing plugins, and KV transport options.

Tested deployment patterns

No need to figure out every piece from scratch

vLLM llm-d kubernetes

llm-d: capabilities for running distributed inference

These are reusable capabilities, not a performance guarantee.

A cross-vendor ecosystem

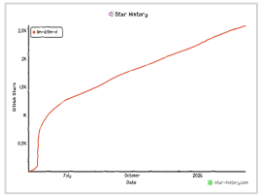
Red Hat · NVIDIA · intel. · CoreWeave · cisco · Hugging Face · IBM · Google · AMD · MISTRAL AI_ · Lambda

Since May 2025

build on the serving runtime, not replace it.

combine reusable llm-d components.

Deployment paths explain the trade-offs.



Community history

This shows project history, not serving performance.

Four areas of capability

State-of-the-art MoE serving

Expert parallelism spreads expert computation across GPUs.

Prefill/decode disaggregation

We separate the phases and transfer KV between roles.

State-aware scheduling

Routing plugins use cache and load signals to pick workers.

Autoscaling architecture

Workload signals and SLOs guide capacity; each integration chooses its policy.

llm-d “well-lit paths”

tested ways to deploy, with clear guidance on what works well and what the limitations are.

Intelligent scheduling

Where should this request run?

KV cache management

Where should reusable state live?

Wide expert parallelism

How should a large MoE model run across GPUs?

P/D disaggregation

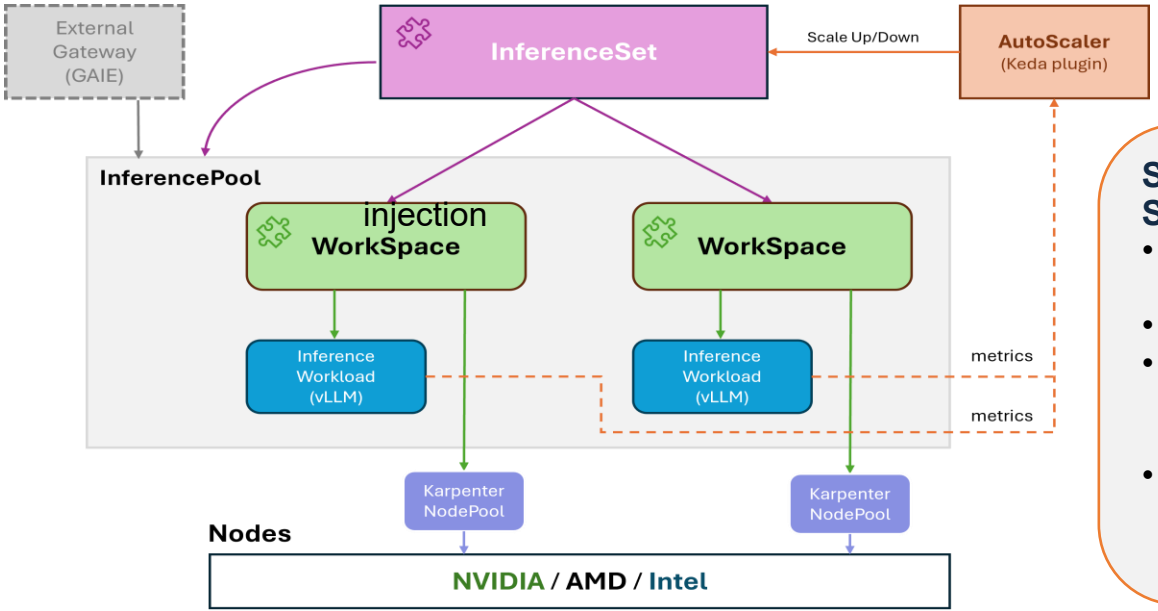
How should prompt and generation capacity differ?

These paths can work together, but we don’t need all of them at once. Results depend on the hardware.

CNCF Sandbox project

What is KAITO (Kubernetes AI Toolchain Operator)?

KAITO automates LLM inference, fine-tuning and RAG with Kubernetes operators.



Specify a model name and GPU SKU

- sizes the nodes to fit the model's memory requirements
- handles model setup and caching
- manages inference workloads and rolling updates on Kubernetes
- uses KEDA to automatically scale inference workloads

KAITO inference workload example

Specify the model name, GPU instance type and replica count.

```
apiVersion: kaito.sh/v1beta1
kind: InferenceSet
metadata:
  name: qwen3-8-27b
spec:
  replicas: 3
  labelSelector:
    matchLabels:
      app: qwen3-8-27b
  template:
    resource:
      instanceType: Standard_NC24ads_A100_v4    # <- GPU SKU
    inference:
      preset:
        name: Qwen/Qwen3.8-27B  # huggingface model ID
```

KAITO works out the initial GPU node count

Specify a model name and GPU SKU. KAITO estimates how many nodes the model needs.

KAITO reserves memory for overhead and KV cache, then fits the weights into the remaining GPU memory.

```
per-GPU budget = GPU memory  ×  0.84            (vLLM memory utilization)
               −  ~2.3 GiB   fixed overhead
               −  KV cache   ( max-model-len × bytes-per-token )

min GPUs    = ceil( model weights  ÷  per-GPU budget )   (weights × 1.02)
node count  = ceil( min GPUs  ÷  GPUs-per-node )
```

Save the result as status.targetNodeCount.
This path uses tensor parallelism within nodes and pipeline parallelism across nodes. It sizes memory, not how much traffic it can handle.

Why bring llm-d into KAITO?

We want P/D routing without replacing KAITO's Kubernetes operating model.

KAITO
MRI → child InferenceSets

+

llm-d
EPP + decode routing sidecar

+

vLLM + NIXL
vLLM runs; NIXL moves KV

Reuse the API model

Keep Workspace and InferenceSet.
Give each role its own replicas,
GPU types and runtime settings.

Integrate, do not rebuild

Reuse llm-d's P/D routing.
KAITO connects the sidecars,
ports and InferencePool.

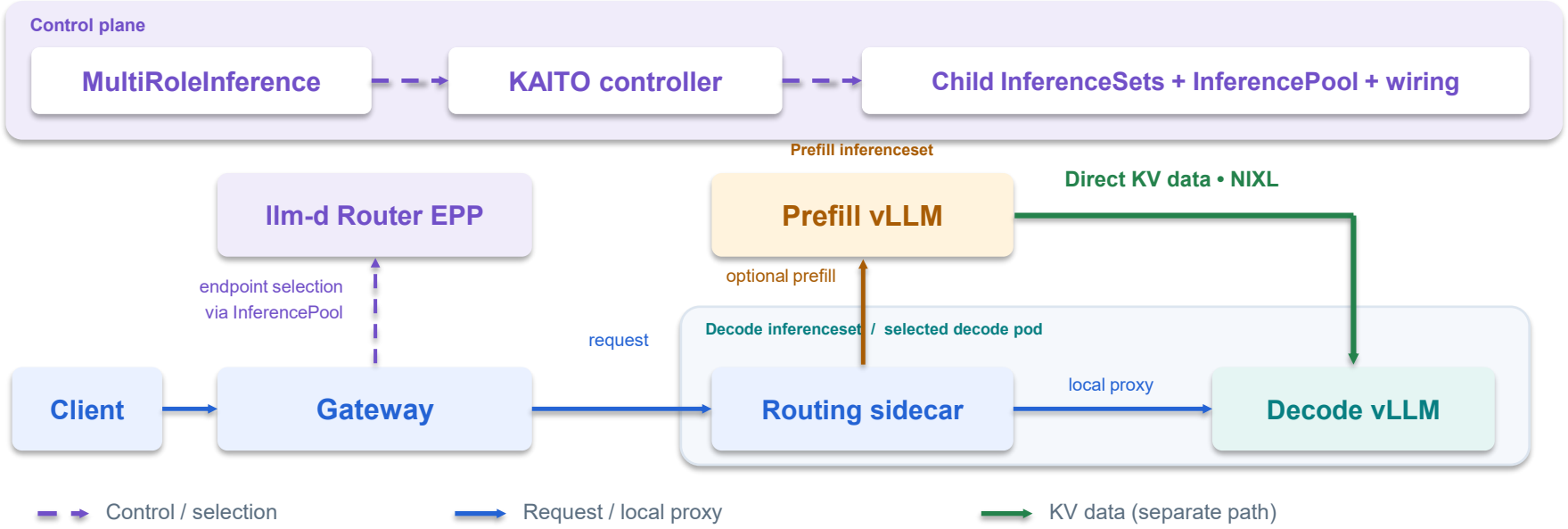
Clear ownership

Use Gateway API and InferencePool.
Keep routing plugins upstream,
not in a second KAITO router.

KAITO manages deployment and lifecycle. llm-d adds endpoint selection and request routing.

One endpoint, separate request and KV paths

KAITO manages the workloads. Requests go through routing; KV cache moves directly between runtimes via NIXL.



The client uses one endpoint. KV data bypasses the gateway, EPP and routing sidecar.

one KAITO MultiRoleInference declares both roles

This example puts one prefill replica on A100 and two decode replicas on H100.

MultiRoleInference / model + role placement

```yaml
apiVersion: kaito.sh/v1alpha1
kind: MultiRoleInference
metadata:
  name: qwen-3-8-27b
spec:
  model:
    name: Qwen/Qwen3.8-27B
  roles:
    - type: prefill
      replicas: 1
      instanceType: Standard_NC24ads_A100_v4
    - type: decode
      replicas: 2
      instanceType: Standard_NC80adis_H100_v5
```

reconcile

qwen-3-8-27b

Qwen/Qwen3.8-27B

One logical service / InferencePool

Prefill role

replicas: 1

A100 / replica 1

Standard_NC24ads_A100_v4

Decode role

replicas: 2

H100 / replica 1

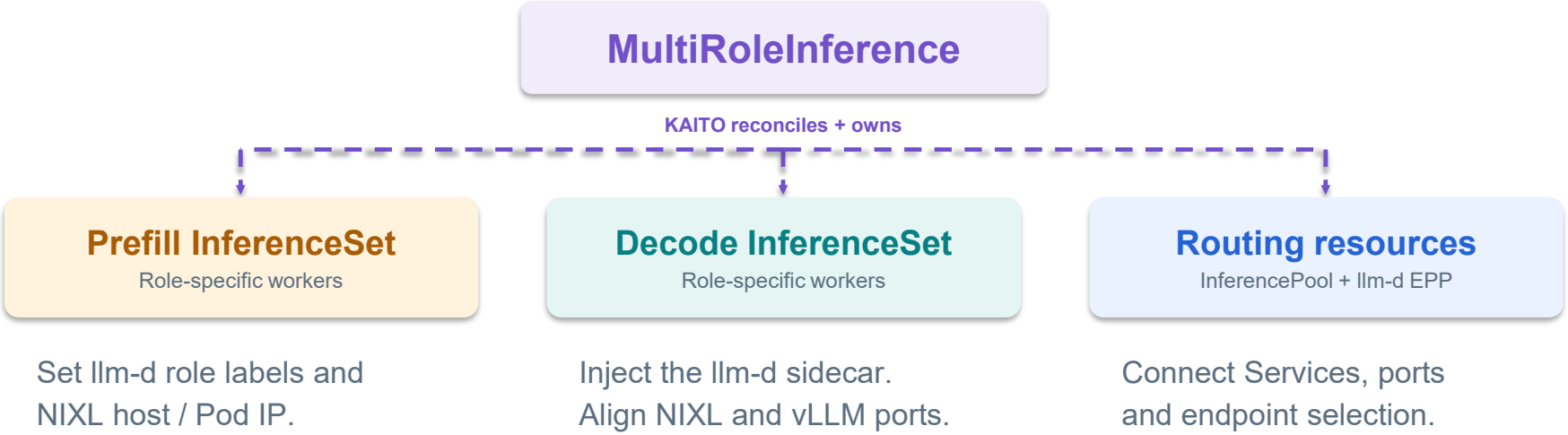
H100 / replica 2

Standard_NC80adis_H100_v5

Both roles serve the same model. Each has its own InferenceSet and capacity settings.

KAITO creates resources and keeps them aligned

KAITO turns the declared topology into workers, routing resources and runtime wiring.

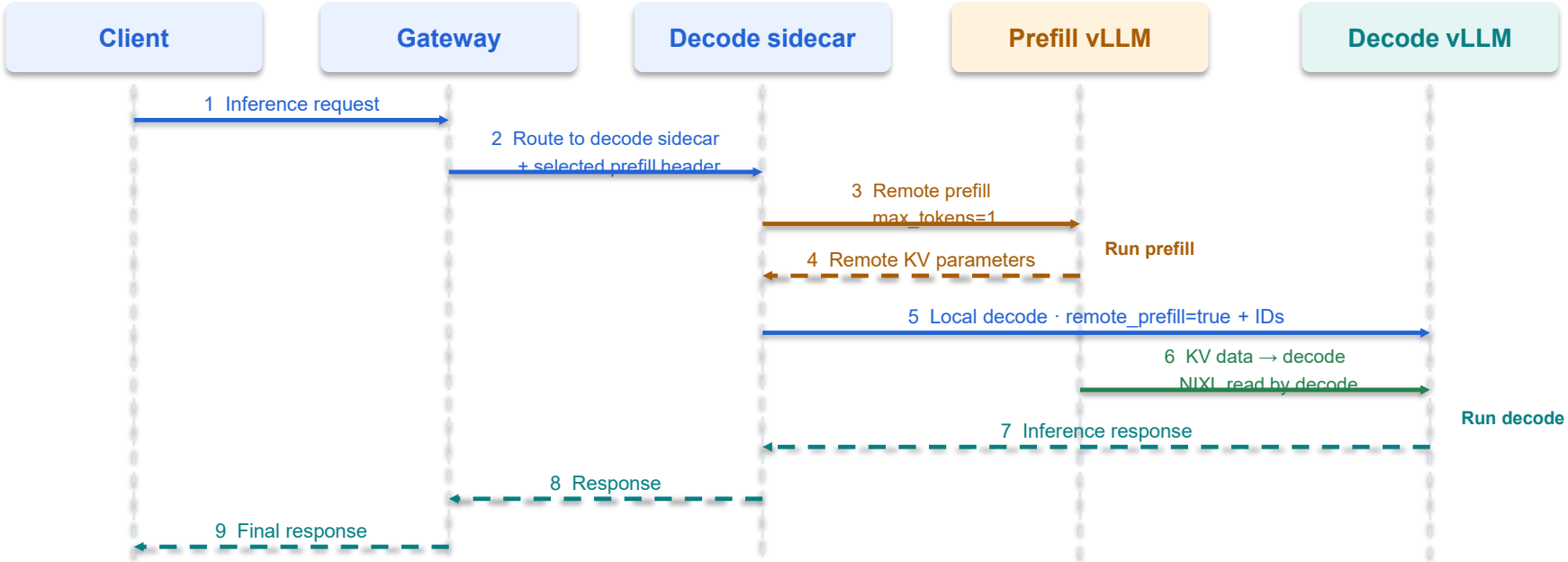


Keep ownership, roles, labels, ports and lifecycle consistent.

KAITO keeps the resources working together; it reconciles them to keep the inference deployment in the desired state.

The decode sidecar coordinates the P/D request

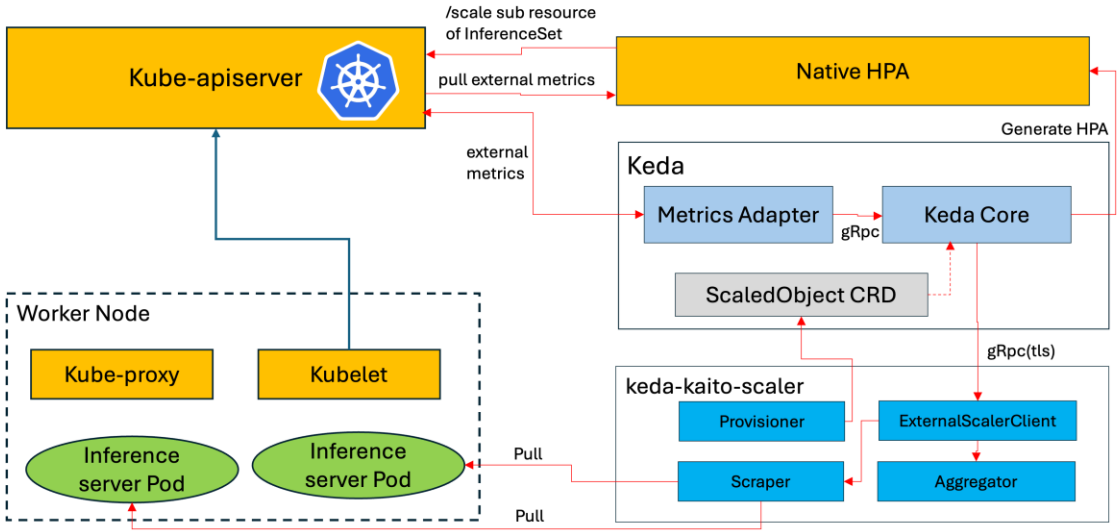
The sidecar calls remote prefill when needed, then invokes local decode.



The sidecar passes calls and KV metadata. The runtimes transfer the actual KV data.

KAITO autoscaling: keda-kaito-scaler

Scale GPU inference with KEDA(Kubernetes Event-driven Autoscaling): no Prometheus required

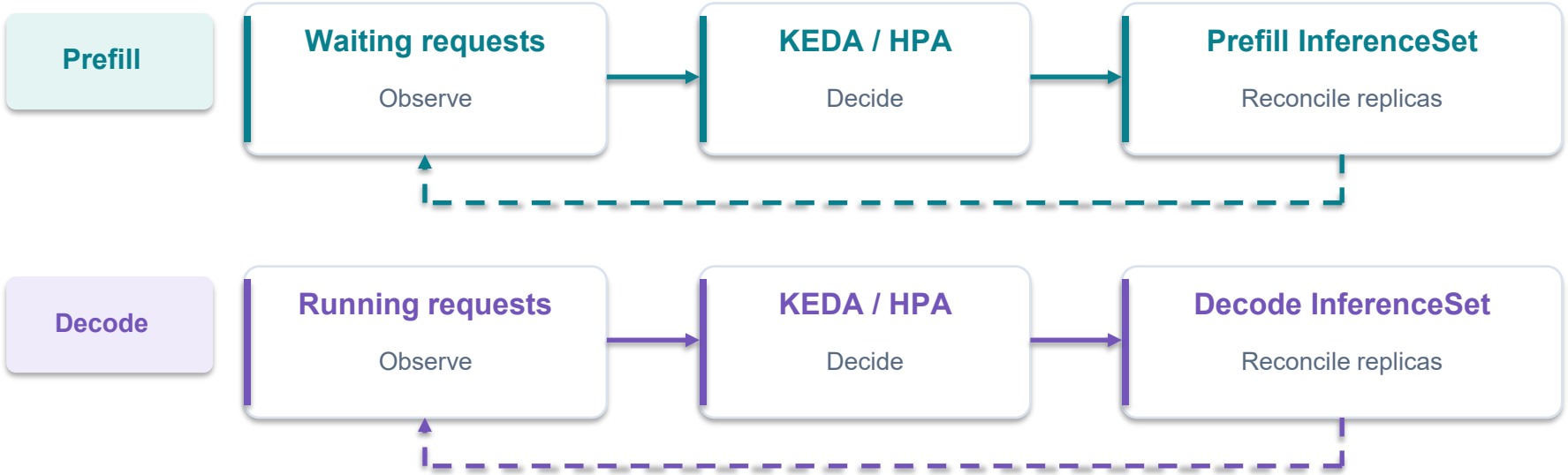


What makes it special

- **Direct metrics** — reads metrics directly from vLLM; no Prometheus required.
- **Gradual scaling** — adds or removes one replica at a time, with cooldown periods.
- **Multiple signals** — can combine queue length and queue wait time; both must cross their thresholds before scaling up.

Let prefill and decode scale independently

each role gets its own signal, threshold and replica target.



Use one autoscaler for each role. Give new pods time to warm up, and scale down gradually

KAITO autoscaling example: scale prefill and decode separately

MRI metadata.annotations

```
autoscaling.kaito.sh/roles: |
  version: v1alpha1
  roles:
    prefill:
      min-replicas: "1"
      max-replicas: "8"
      combinepolicy: AND
      metrics:
      - name: vllm:num_requests_waiting
        type: gauge
        upthreshold: 8
        downthreshold: 1
    decode:
      min-replicas: "2"
      max-replicas: "16"
      combinepolicy: AND
      metrics:
      - name: vllm:num_requests_running
        type: gauge
        upthreshold: 24
        downthreshold: 4
```

Role Averages / Start at P=1, D=2

Waiting is 12; running is 12.

Prefill replicas scale-up; decode holds.

If running later exceeds 24,

decode replicas can scale up on its own.

Use derived signals to explain the load

Derive KV usage, latency and throughput. Check queries before use.

Native input (vllm: prefix)	Derivation	Decision support
kv_cache_usage_perc	max(role samples)	Could guard scale-down
kv_cache_usage_perc	1 - max(role samples)	Shows free fraction, not request capacity.
time_to_first_token_seconds (histogram buckets)	P95 over bucket rates	Shows engine-side TTFT tails.
generation_tokens_total	sum(rate(counter[5m]))	Shows output rate; benchmark it.

KV occupancy is not demand. Throughput alone should not change replica counts.

Where KAITO + llm-d goes next

Next, explore better cache reuse, more inference roles and larger-scale execution.

Near term

Precise cache-aware serving

Route using KV cache events.

Tier cache across HBM, CPU RAM

and optional filesystems.

Next topology

E/P/D

Extend P/D disaggregation to include Encoding

At larger scale

Larger-scale systems

Support wide expert parallelism.

Improve cross-node transport and communication.

CLOUD NATIVE AI + INFERENCE DAY NORTH AMERICA

Resources

Explore the projects, then try KAITO + llm-d P/D guide.

llm-d | CNCF Sandbox

https://github.com/llm-d

KAITO | CNCF Sandbox

https://github.com/kaito-project/kaito

keda-kaito-scaler

https://github.com/kaito-project/keda-kaito-scaler

KAITO + llm-d: P/D guide

https://kaito-project.github.io/kaito/docs/prefill-decode-disaggregation/

Where can you find us

- llm-d.ai
- github.com/llm-d
- llm-d.ai/slack

- https://kaito-project.github.io/kaito/docs/
- https://github.com/kaito-project

https://llm-d.ai

Join Community Slack
llm-d.ai/slack

https://github.com/kaito-project

Join Community Slack
KAITO/slack

Thank you

Q & A

Andy Zhang • Linbo He
Microsoft | KAITO + llm-d