

Serving DeepSeek-class MoE on AMD Instinct™ MI355X GPUs and AMD Pensando™ Pollara AI NICs, fully open source. MoRI-EP carries the expert all-to-all over RDMA, and MoRI-IO streams the KV cache layer by layer from prefill into decode, all on upstream vLLM and llm-d.

2× offline throughput

on average, experts over 16 GPUs (EP16) vs. tensor parallelism on 8 (TP8) ¹

up to 1.3× output throughput

experts over 16 GPUs vs. 8 (EP16 vs. EP8), within latency targets ¹

up to 2.5× goodput

MoRI-IO prefill/decode disaggregation vs. standard serving ²

Upstream

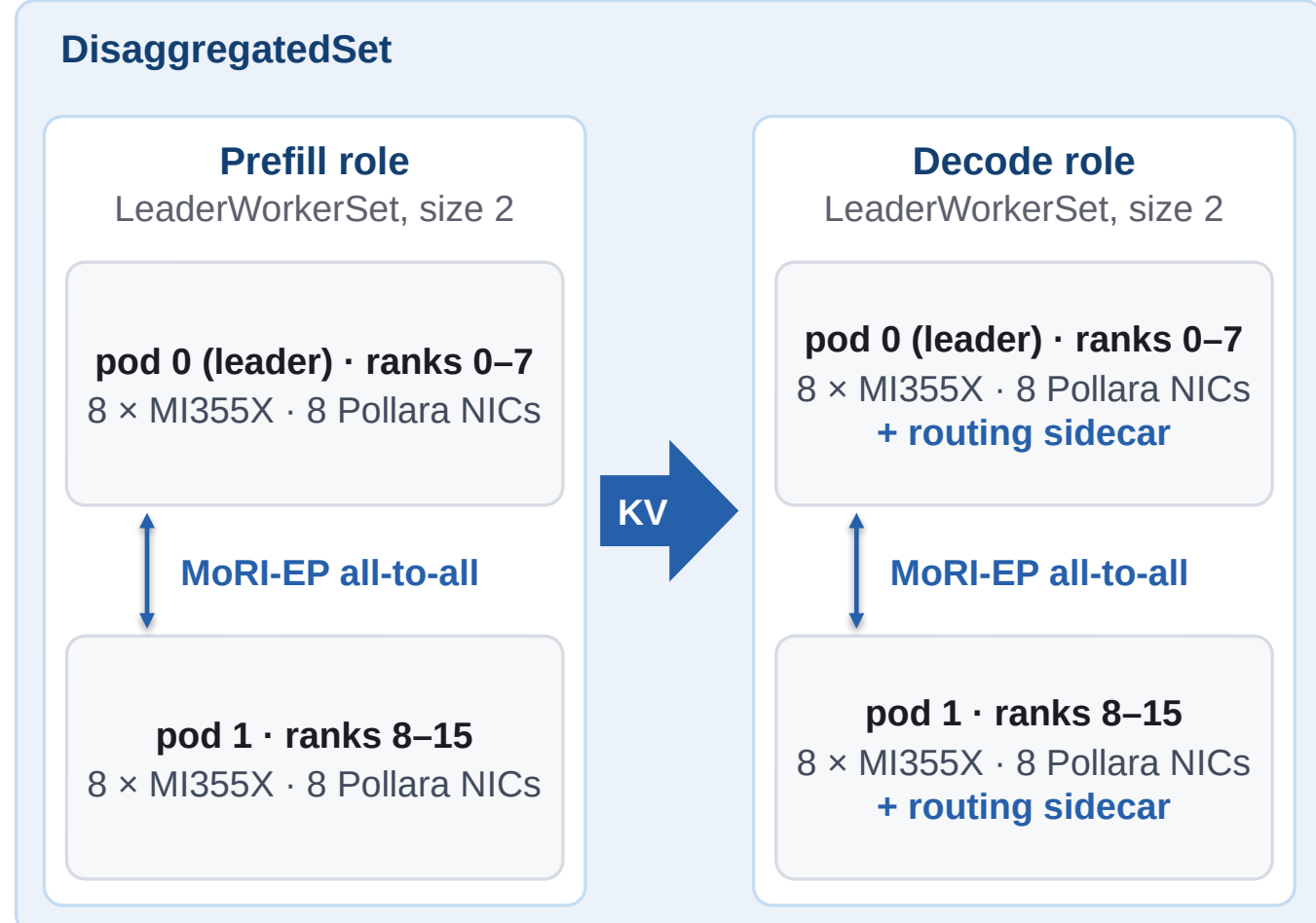
fully open source: every component is in vLLM, llm-d, and MoRI, tested nightly on 2P2D

Why MoE serving is hard

- Sparse MoE:** DeepSeek-V3 has 671B parameters, 256 routed experts, 8 active per token; experts crowd out KV cache
- Token routing:** every MoE layer sends each token to remote experts and back (dispatch and combine)
- Disaggregation:** each prompt's KV cache moves from prefill to decode, and throughput-bound prefill and latency-bound decode need different all-to-all kernels
- Addressing:** one data-parallel job spans pods, so 16 ranks must agree on addresses, rank numbers, and peers

2P2D on llm-d

2 prefill + 2 decode nodes (2P2D), deployed as one DisaggregatedSet requests from gateway + endpoint picker



MoRI-IO writes KV across roles; MoRI-EP links the pods in a role.

Rollouts replace all four pods as one unit, and each pod finds its peers through a shared set label and pod DNS.

Read vs. write mode

	Read mode	Write mode (ours)
Moves the KV	decode pulls	prefill pushes
Dispatch	serial: prefill, then decode	parallel: both at once
Transfer starts	after prefill ends	per layer, during prefill
Router relays block IDs	yes	no

Why write + parallel

- KV transfer overlaps prefill compute instead of following it, which matters most for long prompts
- The same MoRIIOConnector works behind llm-d's sidecar or vLLM's own proxy: one recipe for both

Wide-EP inside vLLM

One 16-GPU job per role; DP = EP = 16, no tensor parallelism



Every rank runs every layer. Each MoE layer does:



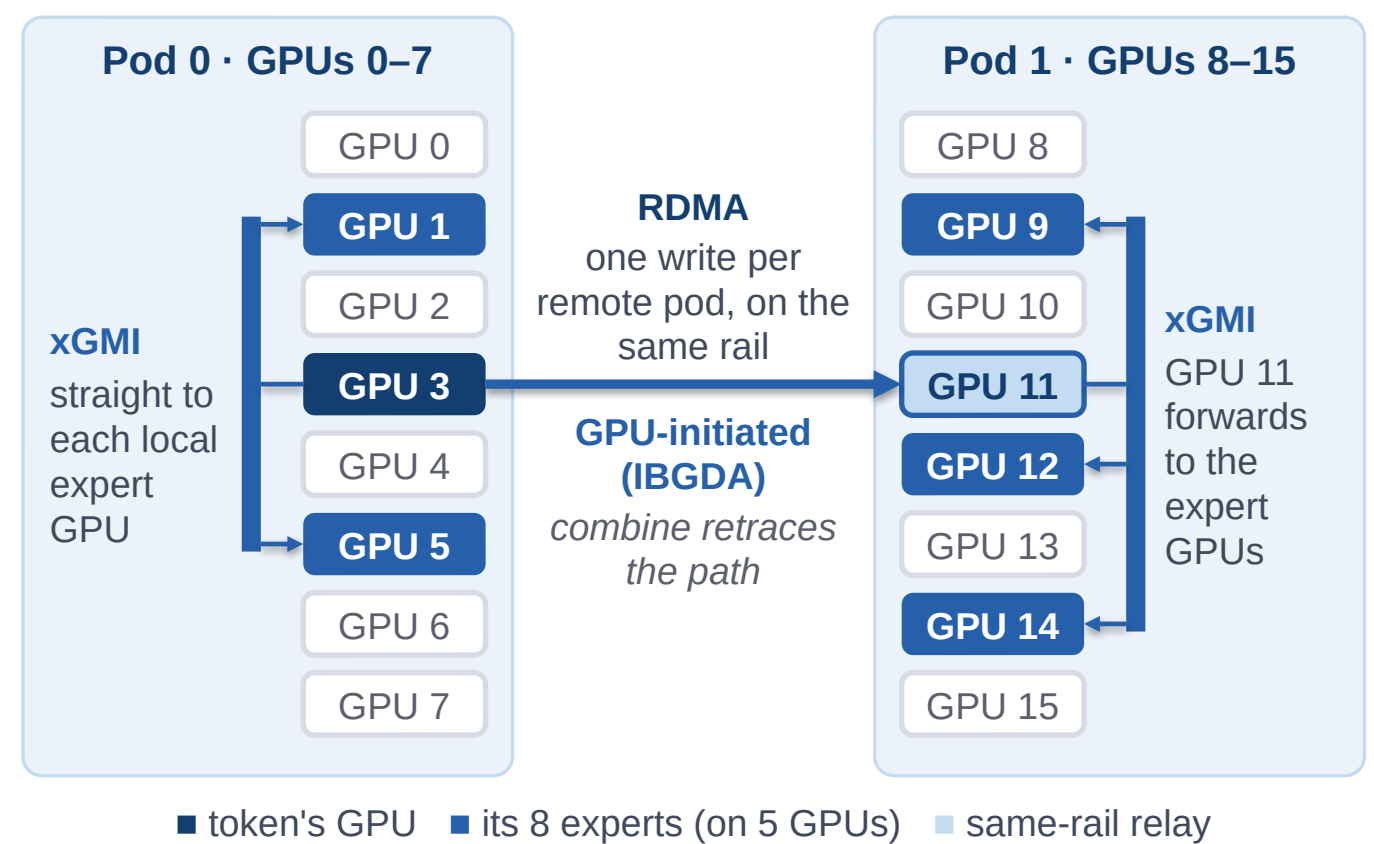
What Wide-EP changes

- Experts are spread, not copied:** each GPU holds 1/16 of the experts, freeing memory for KV cache and more concurrent requests
- One KV copy per token:** attention stays data-parallel (TP = 1); tensor parallelism would replicate MLA's KV cache
- Sparse traffic:** a token visits only the GPUs that own its 8 experts, which is far less data than an all-reduce, so RDMA between pods is enough
- Each phase gets its own kernel:** throughput all-to-all for prefill, low-latency all-to-all with graph capture for decode
- Failures stay small:** each prefill or decode group is self-contained, so a fault stays inside its group
- One recipe for the MoE family:** validated on DeepSeek-V3; the same many-expert, few-active design fits models such as the GLM, Kimi, and Qwen MoE families
- Published AMD results:** on average 2× offline throughput over TP8 and up to 1.3× over EP8 within latency targets, for DeepSeek on Instinct GPUs ¹

Moving tokens and KV on AMD

MoRI-EP for Wide-EP all-to-all, MoRI-IO for KV-cache transfer

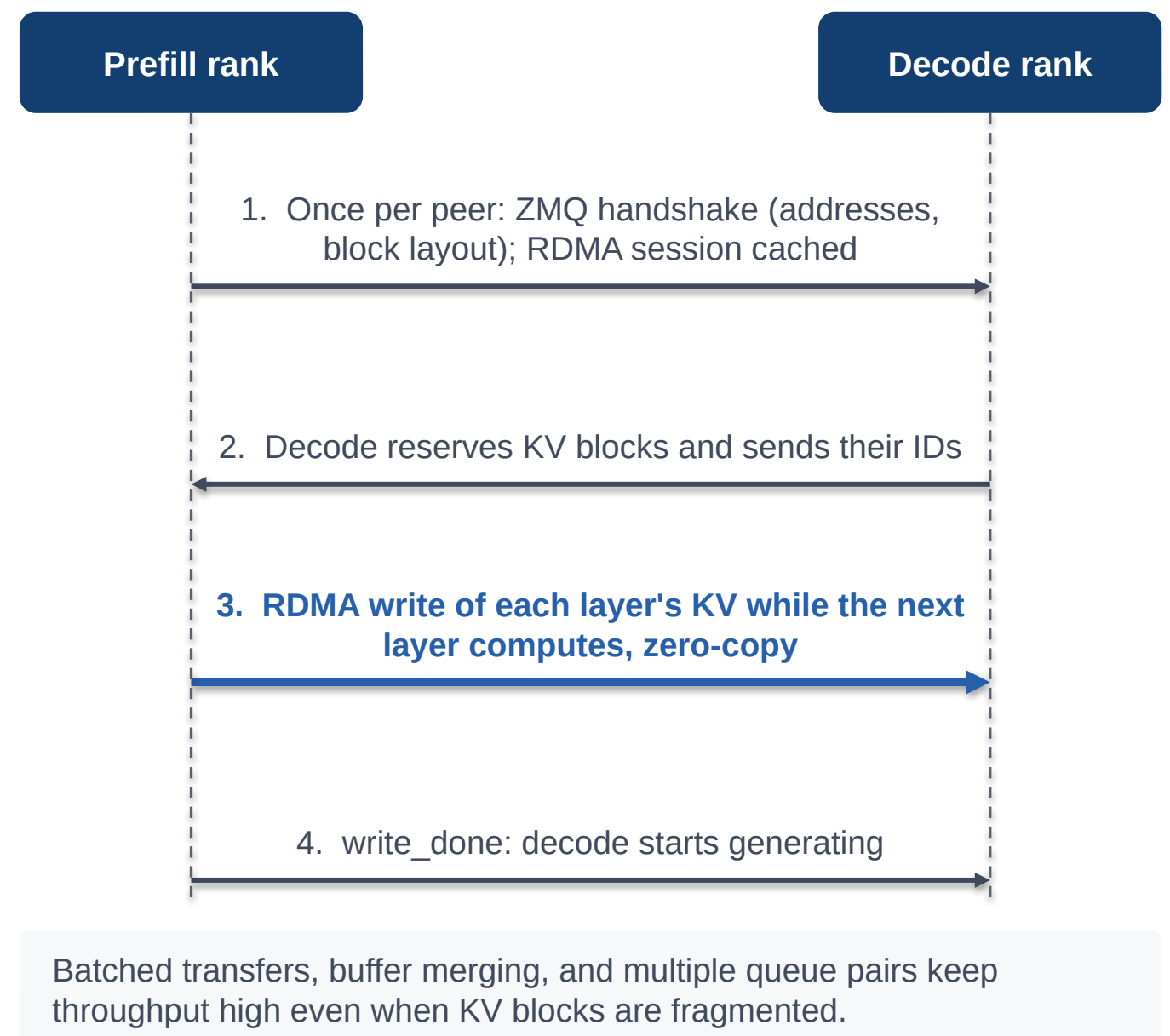
MoRI-EP · one token's trip across two pods



- A token crosses RDMA once per remote pod, not once per expert; xGMI fans it out
- Local and remote sends run at the same time on separate GPU thread blocks
- Prefill uses the throughput kernel (InterNodeV1); decode uses the low-latency, graph-capturable one (InterNodeV1LL)

MoRI-IO · KV-cache transfer, write mode

Plugged into vLLM as the MoRIIOConnector

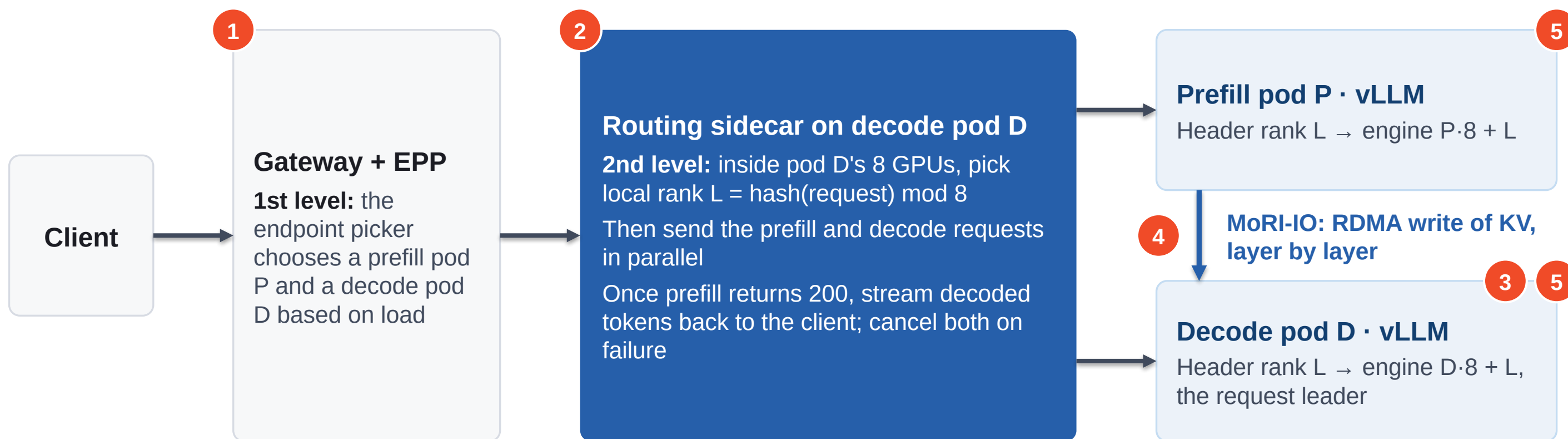


What's next

- More shapes:** wider EP, more instances, and smaller setups, too
- More models:** GLM, Kimi, Qwen, and other large MoE families on the same recipe
- Bigger workloads:** longer contexts and higher concurrency
- Next-gen AMD hardware:** AMD Helios rack-scale platform

llm-d router meets Wide-EP

How one request reaches one rank of a 16-rank job that spans two pods



Why it scales: global rank = pod × 8 + L, so the router only ever names a pod and a local rank. That addressing stays the same as the expert-parallel job grows.

What makes each step work (numbers match the diagram)

- Revision-scoped peers**
New pods pair with new pods; DNS re-resolves
In llm-d
- Pod-local ranks**
Router sends 0-7; vLLM adds pod × 8
In llm-d router
- Single request leader**
One of the 16 ranks signals KV ready
In vLLM
- Reply to requester**
write_done goes back to the engine that asked
In vLLM
- Rank-correct combine**
Each rank combines with its own top-k IDs
In vLLM

Resources

References

- llm-d repositories** github.com/orgs/llm-d/repositories
- MoRI** github.com/ROCm/mori
- MoRI docs** rocm.github.io/mori
- vLLM expert parallel** docs.vllm.ai/en/latest/serving/expert_parallel_deployment
- vLLM MoRIIOConnector** docs.vllm.ai/en/latest/features/morio_connector_usage
- ROCm Blogs** rocm.blogs.amd.com/software-tools-optimization/wide-ep-deepseek/README.html
- vLLM Blog** vllm.ai/blog/morioio-kv-connector

Key terms

2P2D – 2 prefill + 2 decode nodes
EP/DP/TP – expert/data/tensor parallelism
EP16 – experts over 16 GPUs
KV cache – attention keys and values
MLA – DeepSeek's latent attention
RDMA – memory access across nodes
xGMI – AMD GPU link inside a node
IBGDA – GPU-initiated RDMA
EPP – llm-d endpoint picker
AITER – AMD inference kernels



Scan to try it:
[llm-d Wide-EP guide](#)