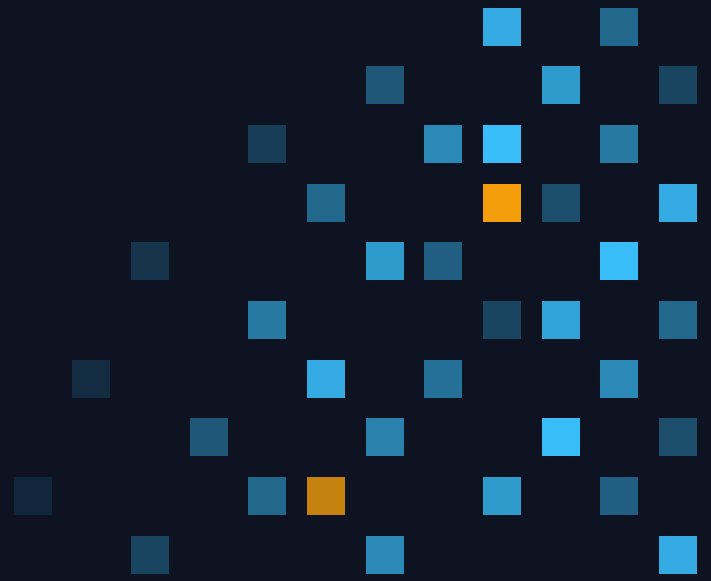
RUNTIME 2026 · TALK SUMMARY

Serving the Agentic Workload

When AI agents become the main workload, what has to change in inference



SPEAKER

Simon Mo

Co-founder & CEO, Inferact · vLLM

ENGLISH SUMMARY

AGICAFET

Condensed from the 18-minute talk video

TL;DR

The whole talk in 1 minute

01

Workloads have changed

Agent work has very long context (P50 ~142k tokens) but short replies (444 tokens), over dozens of turns

02

KV cache is the heart

vLLM answers with Hybrid KV Cache, split prefill/decode and a distributed KV cache pool (Mooncake: 3.8x throughput)

03

Year of diverse hardware

NVIDIA and AMD at production parity; Inferact's experimental TPU megakernel hits 709 tok/s per user

In the agent era, speed comes not just from faster GPUs but from “not recomputing”

CONTEXT

What is vLLM, and who uses it?

Open-source engine for LLM inference and serving

- Began as research at UC Berkeley Sky Computing Lab
- First prototype predates ChatGPT
- First API was Completions, not Chat
- More than 3,000 contributors
- Model labs ship day-0 support with the community

APPLICATION LAYER
Meta · Character.ai · Salesforce · Uber · Roblox · ByteDance · Databricks

MODEL LABS
Kimi · MiniMax · Thinking Machines · DeepSeek · Qwen · Mistral AI

INFERENCE CLOUD
Baseten · DigitalOcean · CoreWeave · Crusoe · Amazon Bedrock

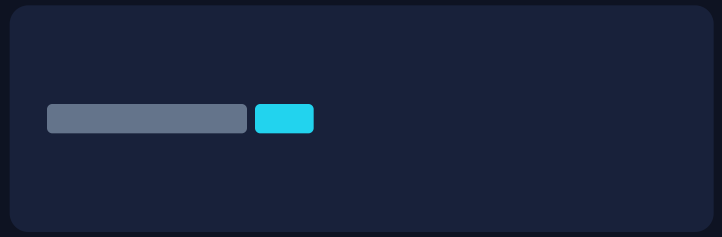
PARTNERS
NVIDIA · AMD · Google · Red Hat · PyTorch

01 · WORKLOAD SHIFT

Three eras of model workloads

2020-2022

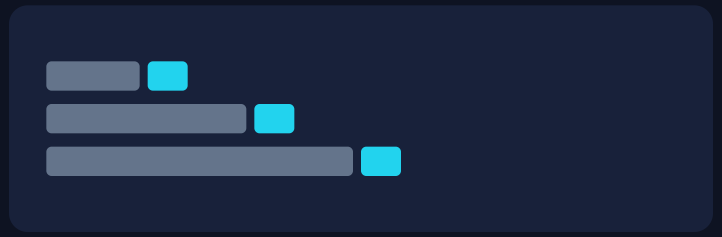
Single-turn completion



One prompt, one answer, e.g. code autocomplete

2022-2024

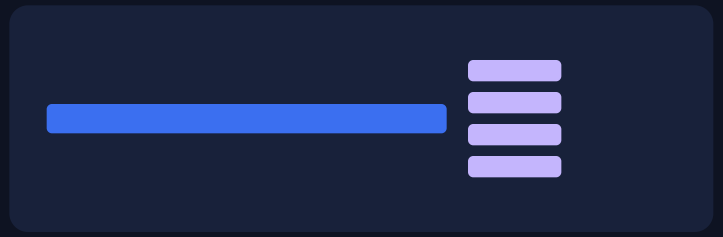
Multi-turn chat



Every turn carries the chat history, so vLLM added prefix caching, chunked prefill and P/D split

2024-2026

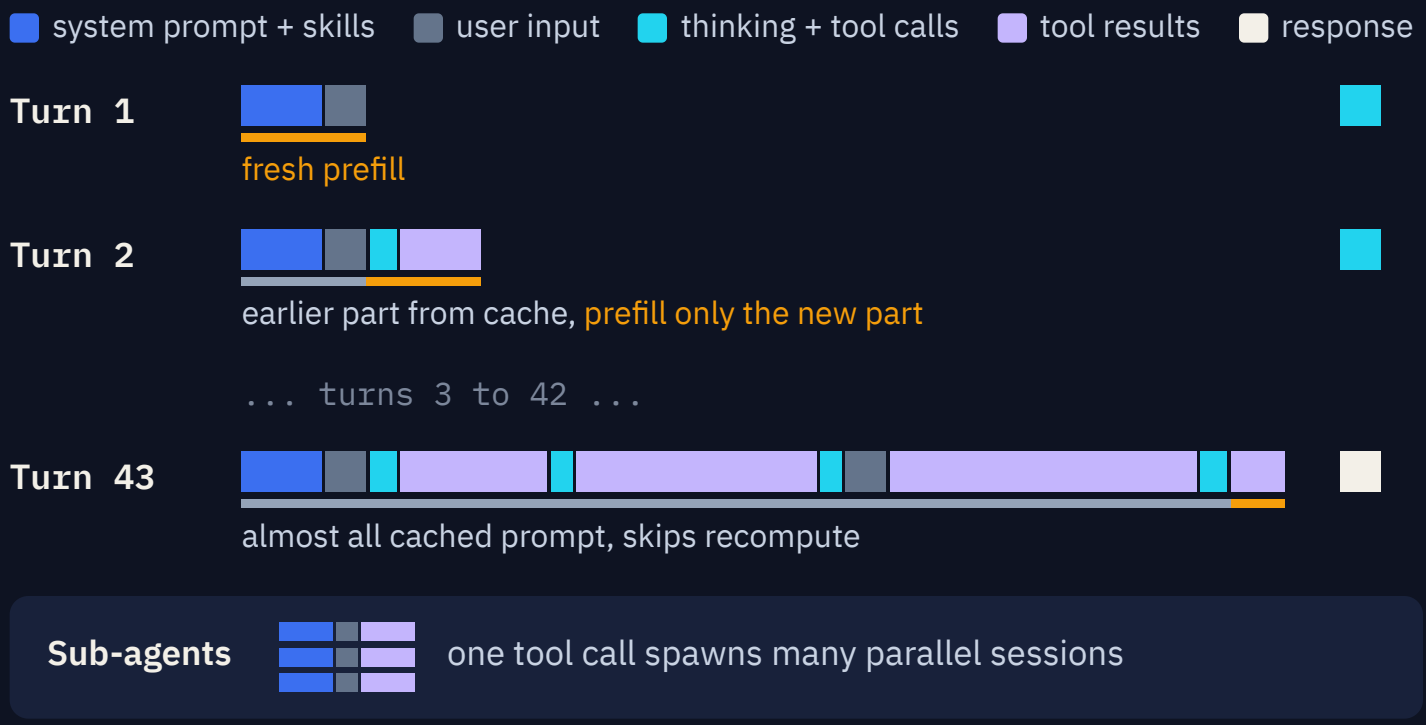
Agentic + real-time



Tool calls, sub-agents, real time: complexity blows up. vLLM's main focus this year

01 · WORKLOAD SHIFT

What one coding-agent session looks like



43
turns per session

142k
P50 input length (token)

444
P50 output length (token)

>95%
reusable prefix (cache hit)

44%
sessions with sub-agents (bursty)

Figures from the talk; same stats as SemiAnalysis AgentX coding-agent traces (vLLM × AgentX post)

THE NUMBER TO REMEMBER

Read 142,000 tokens to write only 444 tokens



Every turn the system “reads” a long context again but “writes” very little, so real speed depends on how much cache it can reuse

Ratio computed from the P50 values on the previous slide (142k ÷ 444)

THE BASICS FIRST

Prefill, Decode and KV Cache, made simple

Prefill

“Read the whole prompt”

Reads prompt, history and tool output in one go. Compute-heavy, sets TTFT (first-word wait)

Decode

“Write one token at a time”

Generates the thinking and next action. Memory-bandwidth-bound; sets smoothness (tok/s per user)

KV Cache

“Notes taken while reading”

Saves the computed state of tokens read so far. Same prefix, no recompute: a prefix cache hit

The catch: these “notes” eat GPU memory, so long contexts and many users need careful management

Background slide added by the editors

01 · WHERE IT GETS HARD

Where it gets hard: 3 serving challenges

The system serves thousands of users at once: some sessions run 10 turns, some 100, some tool results are 100k tokens long, and it must multiplex it all

01

Prefix cache pressure

Every call replays the full context, so KV cache reuse carries the load.

02

Execution efficiency

Each step processes more tokens while staying inside a tight latency budget and SLA.

03

Global routing

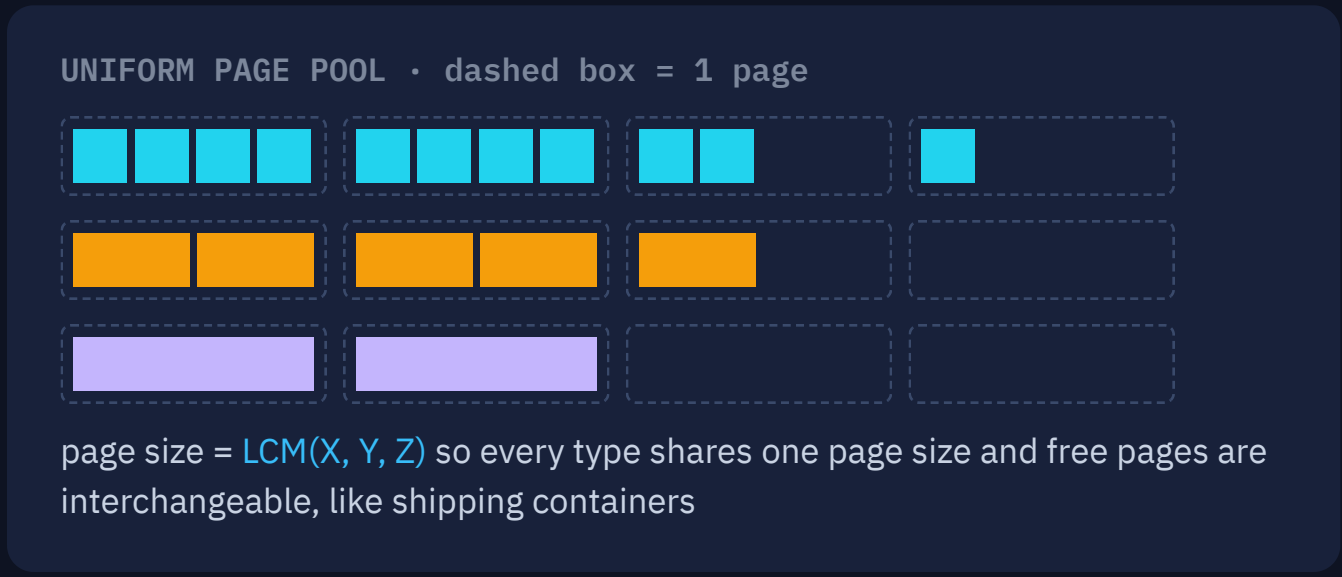
Routing sessions and bursty sub-agents means trading cache affinity against load balance.

02 · SOLUTION · KV CACHE ON ONE MACHINE

Hybrid KV Cache: one page size for all attention

New models mix attention types across layers, and each type needs a different KV cache

- **Full attention**
  cache grows with every token
- **Sliding window**
  recent window only, fixed size per request
- **Linear attention**
  fixed-size state, but a large one



2-2.5x more KV cache space freed in production: more users per chip, better throughput and cost

02 · SOLUTION · ACROSS MACHINES

Why production splits Prefill and Decode

Aggregated one server does both



- ⚠ Heavy prefill: streamed thinking stalls midway
- ⚠ Heavy decode: prefill can't get in, TTFT suffers

Disaggregated split roles

Prefill pool
reads context

→

Decode pool
writes the answer

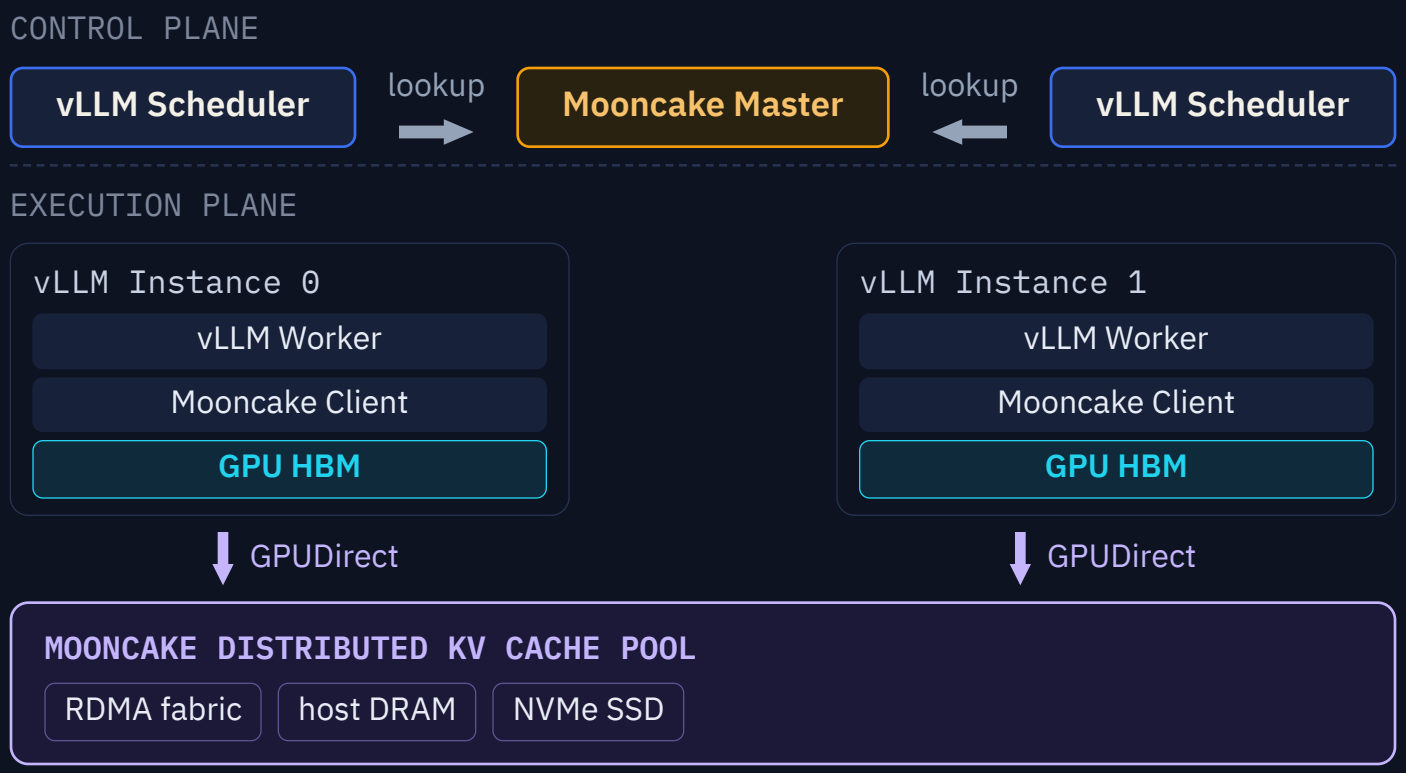
KV Cache Pool shared by all machines

- ✓ Steady latency, no fighting over machines
- ✓ Scale each side independently per workload

“Nobody is serving production agentic workload on an aggregated inference setup.”

02 · SOLUTION · ACROSS MACHINES

Distributed KV Cache Pool with Mooncake



Simply put

KV cache is no longer stuck on one GPU: a prefix computed on one machine can be reused by another

3.8x

higher throughput

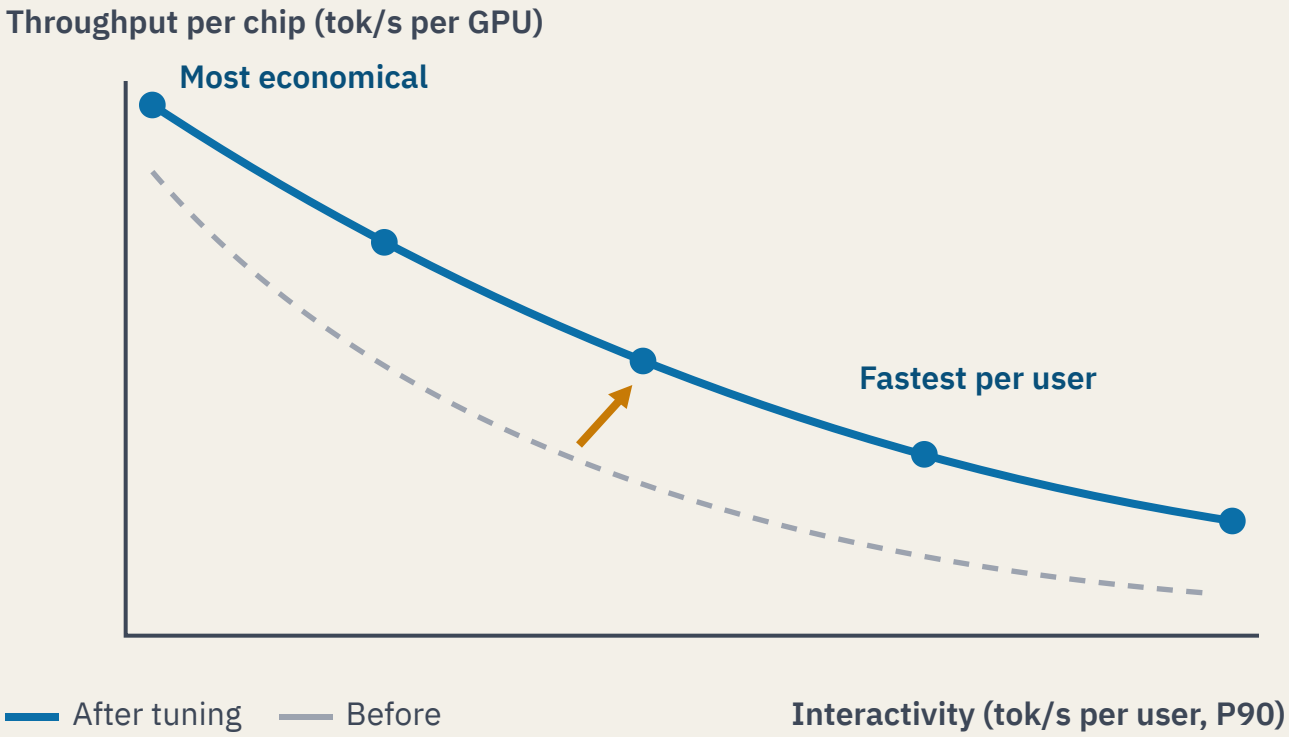
46x

lower P50 TTFT on agentic traces

vLLM integrates natively with several frameworks · this one is Mooncake · numbers: vLLM × Mooncake post (Codex traces, 1P1D on 12× GB200)

02 · RESULTS

How it’s measured: Throughput vs Interactivity



Every point is one configuration (parallelism, batch, kernel)

Pareto frontier means you can’t be cheapest and fastest at once; you choose

vLLM’s job is to push the curve up and right, and let you pick any point

REAL BENCHMARK

SemiAnalysis InferenceX · Kimi K3 2.8T · NVIDIA GB300, GB200, B200, B300 and AMD MI355X

Left chart is a simplified illustration of the idea

Original chart: Kimi K3 2.8T Agentic Token Throughput per Chip vs P90 Interactivity (updated 21 Sep 2026)

03 · WHAT'S NEXT

Next Steps for vLLM Agentic Serving

Agentic-first

Agent/session hints, programmable KV caches: set TTL, priority, where cache lives

E.g. a session you won't touch for an hour: pin it, leave it to LRU, or evict it?

Agentic-efficient

Ultra-low latency (spec decode, engine efficiency); instant scaling and recovery

Goal: pick any point on the Pareto curve and it runs in production automatically

Agent-friendly

Interfaces and infra for agents developing vLLM

The team uses models served by vLLM to optimize vLLM: a closed loop with agentic evaluation

03 · HARDWARE HETEROGENEITY

This is the year of hardware heterogeneity

Chips from several vendors now match on specs and availability (parity of chips), each with its own strengths

NVIDIA

BLACKWELL · B200 / GB200 / GB300

General purpose, well designed and battle-tested

AMD

INSTINCT MI355X

Leads on HBM size (288GB). More large vLLM users now deploy production on AMD

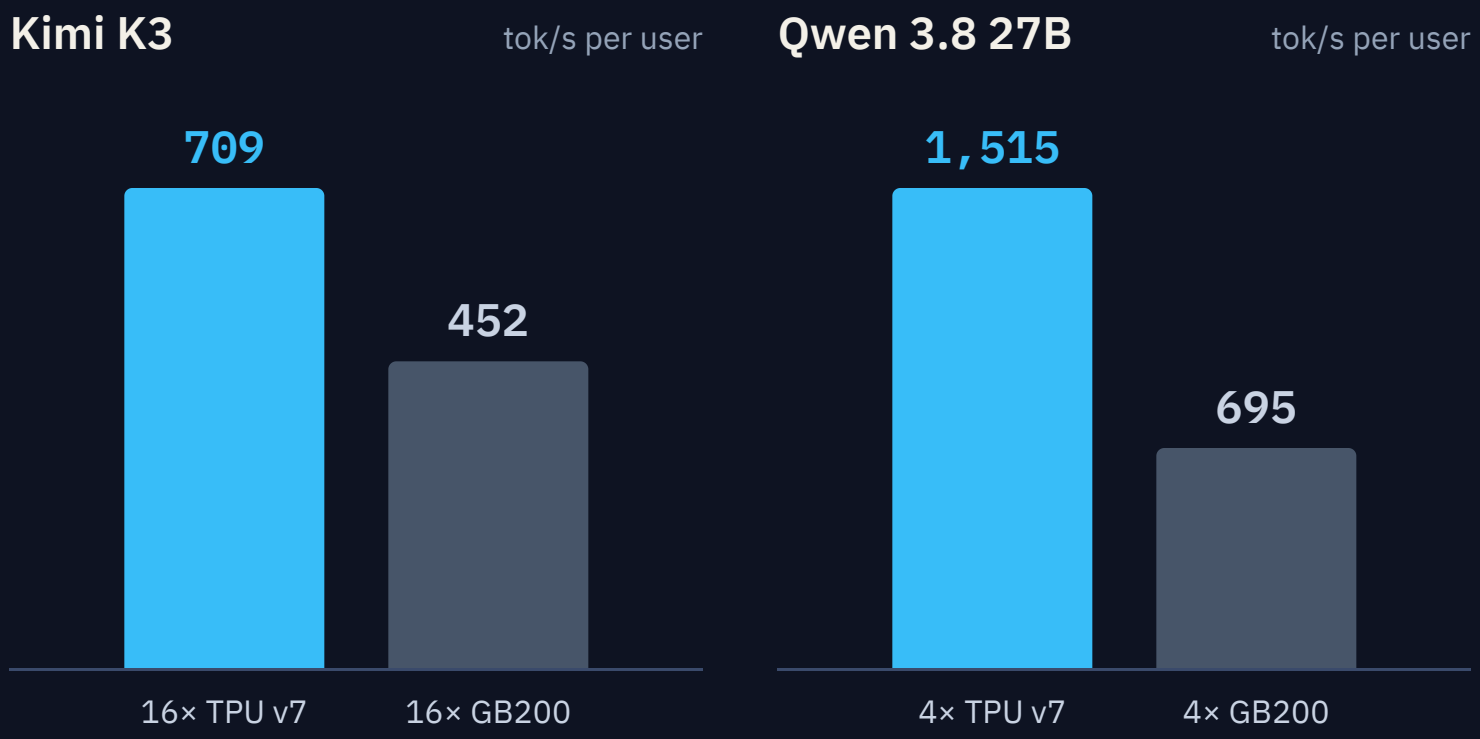
Google TPU

TPU v7 · IRONWOOD

Giant systolic arrays plus large, programmable VMEM. Powers models like Gemini

03 · HARDWARE HETEROGENEITY

First TPU Megakernel: 709 tok/s/user



TPU megakernel

vLLM baseline on GPU

1.6x / 2.2x

faster than baseline (Kimi / Qwen)

Not bench-maxing, but to understand how hardware differs

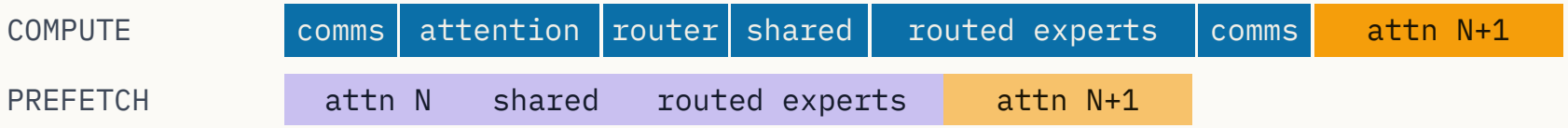
Inferact's figures, speculative decoding on both sides (acceptance length 6) · each chart uses its own bar scale · multipliers computed from the chart

Inside the Megakernel: overlap, no waiting

Before: separate kernels, each step waits on the last



Megakernel: all 92 MoE layers run in a single Pallas call



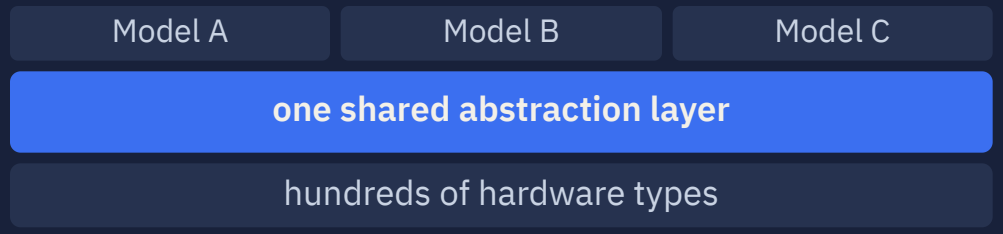
TPU’s VMEM (like GPU shared memory, but bigger and more programmable) lets it prefetch the next layer’s weights onto the chip while the current layer computes, like a chef prepping the next dish while this one cooks

Diagram redrawn from Inferact’s “Inside the Megakernel” slide

03 · LESSON FROM THE vLLM TEAM

“Premature abstraction is bad.”

2024-2025 · unify



Abstract kernel and model interfaces to unify and simplify across thousands of architectures

2026 · split for speed



Each model gets its own file, kernels and almost its own inference engine, for extreme performance

Alongside: agents write more kernels, adopt the latest kernels from hardware teams, keep model structure flat

03 · ROADMAP

vLLM Q4 Roadmap: Agentic Production Serving

Area	Focus	What it means
PRODUCTION	Agentic workloads · Large-scale serving	Serve agent workloads in production at large scale
PERFORMANCE	Model performance · Speculative decoding	Agents help write kernels; cut latency at low concurrency
REFINEMENT	KV cache refactoring · CI & release	Restructure KV cache; keep vLLM reliable and secure
ECOSYSTEM	Hardware support · RL & Omni	New hardware, RL workloads and omni-modal models

Every area has a working group open for the community to join and contribute

TAKEAWAYS

Applying this to your own AI stack

Measure your real workload

Check the input/output ratio, turn count and prefix cache hit rate of your agent traffic

Keep the prefix stable

Put the system prompt, tool definitions and static docs first in context for high cache hits

Pick the Pareto point that fits

Batch jobs favor throughput per chip, interactive ones tok/s per user: tune serving differently

Plan to scale across machines

Beyond one node, consider prefill/decode disaggregation and a KV cache connector like Mooncake

Editors' notes, not part of the original talk

JOIN THE vLLM COMMUNITY

Let’s make open-weight models run faster

Join 3,000+ contributors around the world

BUILD

docs.vllm.ai

LEARN

vllm.ai/blog

CONTRIBUTE

github.com/vllm-project/vllm

Source: “Serving the Agentic Workload” by Simon Mo, Co-founder & CEO, Inferact · Runtime 2026 · English summary by AGICAFET