VLLM BANGKOK DAY · 6 SEP 2026

Supercharging vLLM with LMCache

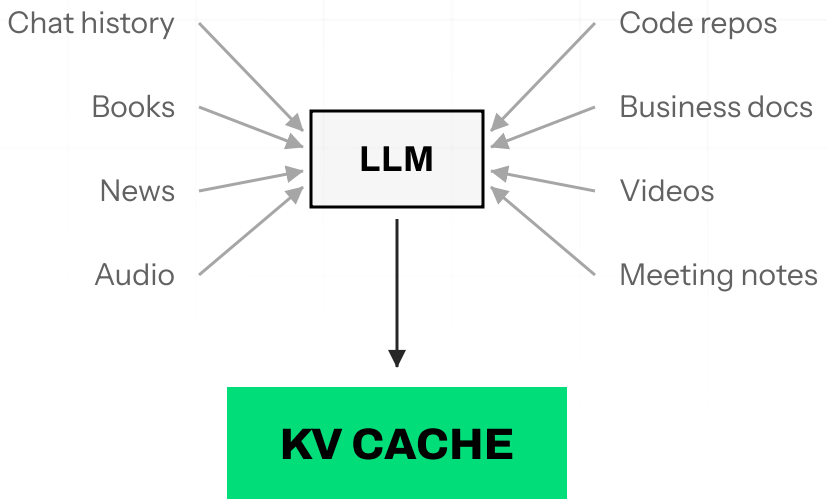
KV-CACHE REUSE IN PRACTICE SHAW NGUYEN · TENSORMESH #VLLMBANGKOKDAY

tensormesh

Long-context inference

"In the next year, you're going to see very large context windows… When they are delivered at scale, it's going to impact the world at a scale no one understands yet."

Eric Schmidt · former CEO, Google



Context keeps growing. **All of it becomes KV cache before the model answers anything.**

WHY IT MATTERS · THE COST OF STATE

Your GPU's most expensive bytes

PER TOKEN

320 KB

of KV cache

LLAMA-3.1-70B · BF16 · ALL LAYERS

PER REQUEST

~40 GB

of state at 128K context

HALF AN H100 - FOR ONE REQUEST

LIFETIME

1 request

and then it's freed

KV CACHE IS PER-REQUEST STATE BY DEFAULT

The math: 2 (K+V) × 80 layers × 8 KV heads × 128 head-dim × 2 bytes ≈ 320 KB / token

> To produce one token the model re-reads everything before it. The KV cache **is** the conversation state - and most clusters pay to rebuild it from scratch, all day long.

WHY IT MATTERS · WHO'S BEHIND THIS

The team that turned KV-cache reuse into a field

01

The researchers

Founded by the people who pioneered KV-cache reuse, and who still publish the ideas the field builds on.

UCHICAGO · UC BERKELEY · CMU

02

LMCache · open source

We build and maintain the de-facto open-source KV-caching layer - now a project of the PyTorch Foundation.

10K+ STARS · 250+ CONTRIBUTORS · 800K+ DOWNLOADS

03

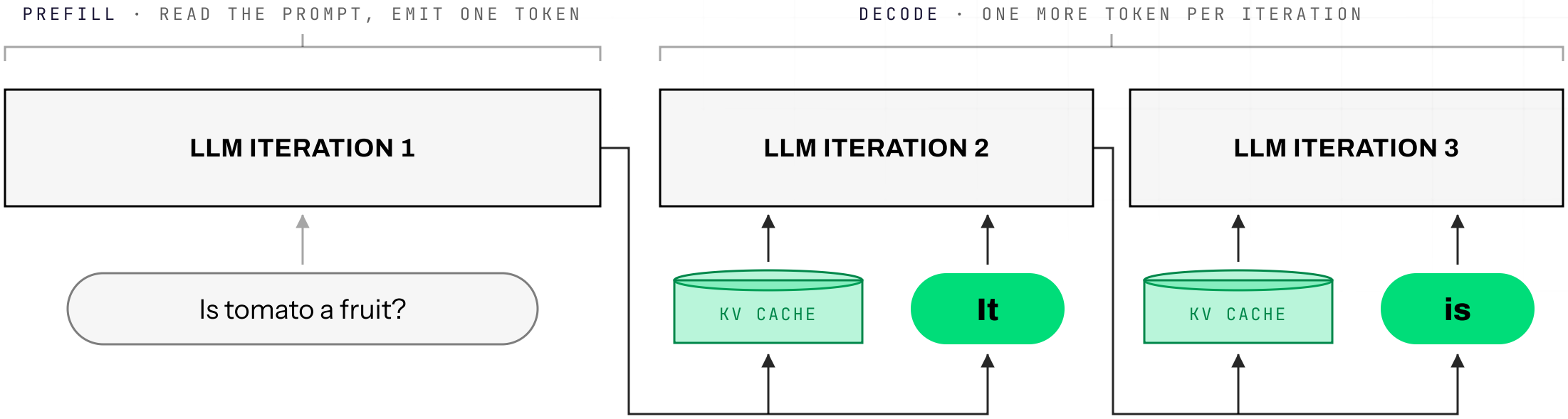
You already run our code

vLLM's own tree ships the LMCache connectors. Every mainstream orchestrator has an LMCache path.

NVIDIA DYNAMO · RED HAT LLM-D · BYTEDANCE AIBRIX · VLLM PRODUCTION-STACK

Everything under **LMCache** in this talk is open source - clone it and run it tonight.

Prefill once. Then decode, one token at a time.



Prefill is compute-heavy; decode is bandwidth-heavy. Everything after this slide is about **not paying for prefill twice.**

PagedAttention + automatic prefix caching

PagedAttention

- KV cache in fixed-size blocks
  A page table, like OS paging - no fragmentation.
- Blocks shared and reused
  Across requests - the seam every cache layer plugs into.

Automatic prefix caching

- Byte-identical prefix → skip that prefill
  Block hashes match. On by default in vLLM V1.
- Hits are partial and block-granular
  The longest cached prefix wins - never all-or-nothing.

> Excellent engineering, and the engine internals are well covered elsewhere. Our question is a different one: **what happens to these expensive blocks over time, and across pods?**

FUNDAMENTALS · WHERE IT STOPS

The cache dies young

Evicted	HBM pressure - LRU eviction and preemption drop blocks mid-flight.
Killed	Restart, upgrade, node swap: in-process GPU memory dies with the engine.
Orphaned	The router sends turn 2 to a replica that never saw turn 1, and a scale-up starts cold.
Blind spot	Only a byte-identical **prefix** can hit - mid-prompt reuse, like a RAG document, can't.

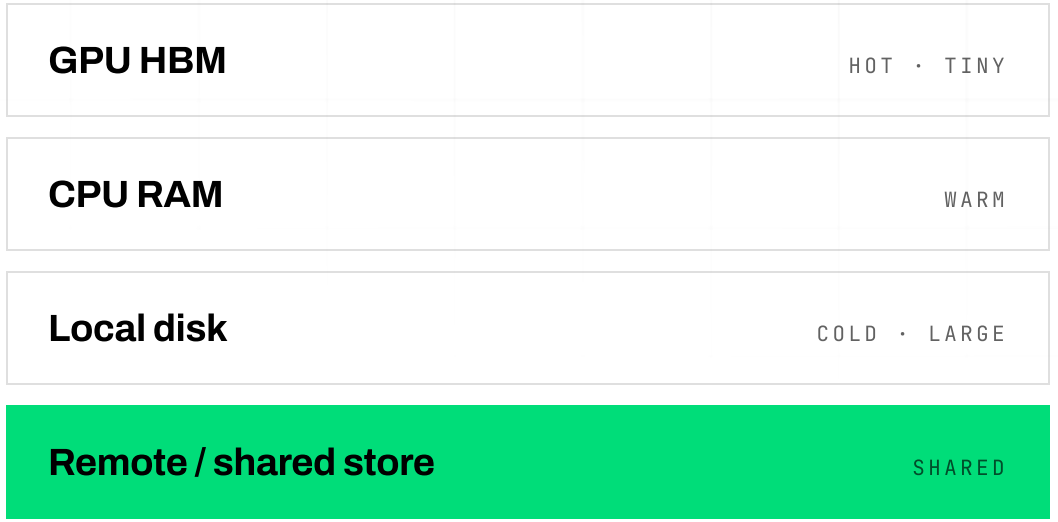
> Born at great expense, paid out once or twice, dead young. And the metric nobody graphs: **the KV-cache hit rate.**

LMCACHE · THE FIX, OPEN SOURCE

LMCache - stop deleting it

Step one is simply: **stop deleting it.** LMCache gives the KV cache a memory hierarchy that survives requests, engine instances, even restarts.

- **Offload past the GPU**
  CPU RAM, then local disk, then a remote shared store.
- **Decoupled from the engine**
  Its own process, so the cache outlives the engine. vLLM, SGLang, TRT-LLM.
- **Honest about the overlap**
  vLLM ships connector and offloading blocks natively; LMCache is the mature multi-tier layer on that seam.



LMCACHE · MOVING KV CHEAPLY

CacheGen - make the cache cheap to move

At 320 KB per token, shipping raw KV across tiers - or across the network - can cost more than recomputing it.

- Compress and stream the KV

  A purpose-built encoder, not a general-purpose codec. SIGCOMM '24.

- Keeps the lower tiers economical

  CPU, disk and network stay worth using, even without NVLink-class links.

FORESHADOW

Whether a fetch beats a recompute is pure economics.

Reuse share × bandwidth × model size. We'll come back to this.

LMCACHE · ONE REQUEST, END TO END

How a cached request actually flows

> A cache hit is **partial, not all-or-nothing** - reuse what's cached, prefill only what's new. And with **CacheBlend**, "what's cached" starts to include the middle.

LMCACHE · REUSE THE MIDDLE

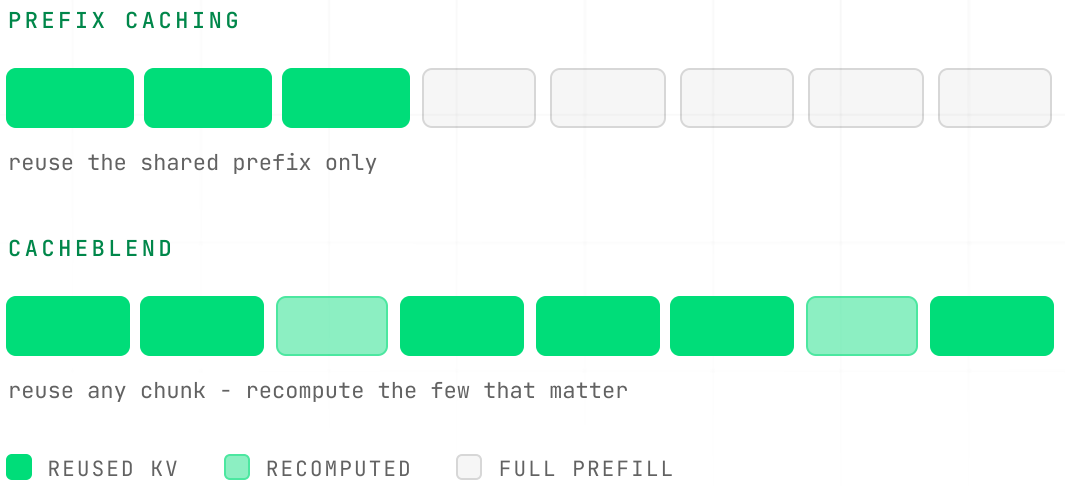
CacheBlend: beyond the prefix

Prefix caching stops at the first differing token. But real reuse - a RAG document, a reordered agent context - lands **mid-prompt**.

CacheBlend reuses those chunks anyway, recomputing only a small fraction of tokens to stitch the cross-attention at the seams.

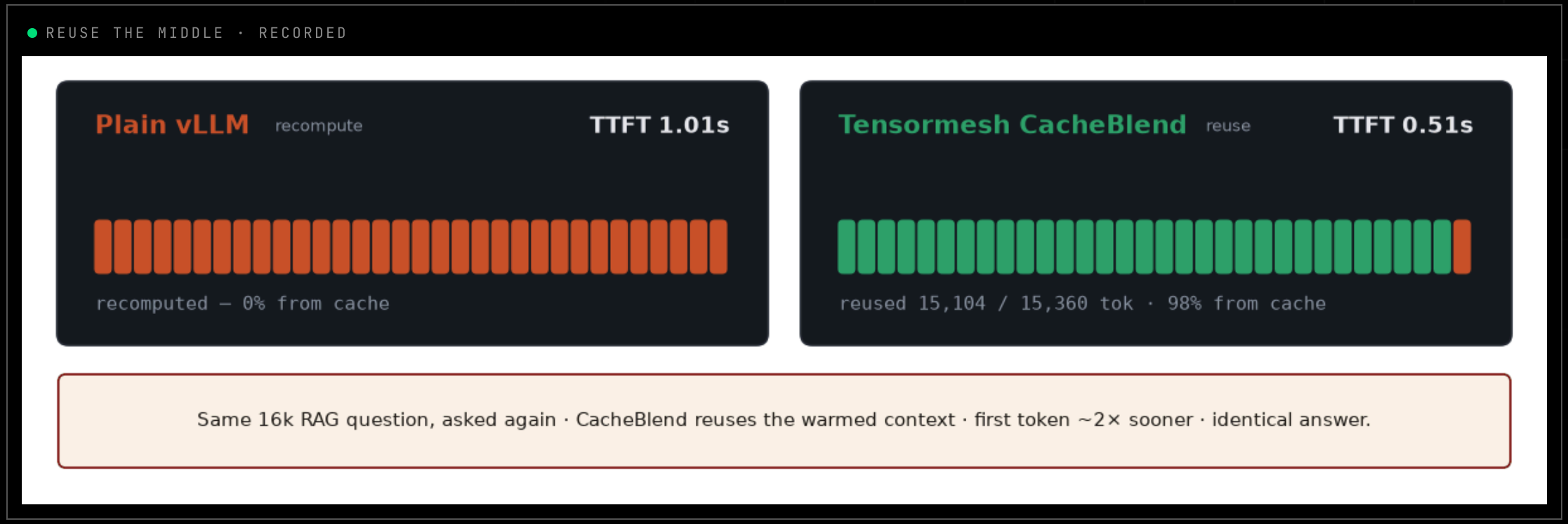
OPEN SOURCE · EUROSYS '25

RECOMPUTE 15% BY DEFAULT



2.2–3.3× faster TTFT at answer quality that **matches** full prefill · EuroSys '25, 2WikiMQA, Llama-70B / Yi-34B, 2×A40.

LMCACHE · DEMO



Same 16K-token RAG question, asked twice · left: plain vLLM recomputes · right: 98% of tokens from cache, first token in half the time · same answer

WORKLOADS · WHERE REUSE PAYS

Long documents and RAG re-read the same context

The same corpus is re-read across requests - and once it outgrows GPU memory, **every miss re-prefills.**

OFFLOAD OFF → ON

5.9×

faster first token

TTFT P50 1,626 → 274 MS

OFFLOAD OFF → ON

4.57×

the throughput

INPUT 13.1K → 60.0K TOK/S

OFFLOAD OFF → ON

7.6×

better at the tail

TTFT P99 8,400 → 1,098 MS

A 120B model · long-document QA · offload off vs on · working set 3.4× GPU KV · conc 16 · TP=2 · n=3

One operating point, not a constant - **offload wins most exactly when the working set outgrows GPU memory.**

What 4.57× actually buys

SAME HARDWARE, SAME SLO	LONG-DOC & RAG	UNIQUE BATCH
reuse share	the same corpus, read again and again	≈ none
measured, offload off → on	**4.57× input throughput**	no change
so: same fleet	serves ≈ 4.57× the traffic	-
or: same traffic	**≈ 78% fewer GPU-hours**	-

> **Working set 3.4× GPU KV, n=3.** This converts that one operating point, not a rule - the arithmetic holds here and nowhere else.

> Savings scale with **reuse share × bandwidth × model size**. The 4.57× is a capacity number - **spend it as fleet or as budget, not both.**

WORKLOADS · ONE TRUE STORY

The 0% hit-rate mystery

`Deployed ✓`	Caching configured correctly, capacity fine, everything green.
`Telemetry ✗`	Hit rate ≈ **zero** - through six rounds of debugging.
`Root cause`	The **chat template** rendered history slightly differently each round, so no two turns shared a byte-identical prefix. Every request a full miss.
`It generalises`	Timestamps, request IDs, shuffled few-shots, non-deterministic serialisation - all silently zero your hit rate.

> Reuse requires **byte-identical tokens**. Never assume your hit rate - **measure it.**

Is this workload worth caching?

- Chat, agents, RAG – **yes**

  History or documents repeat. Prefix-shaped or mid-prompt, there is something to hit.

- Unique one-shot prompts – **no**

  Nothing repeats, nothing to hit. No cache helps a workload that never repeats itself.

- Tiny prompts – **probably not**

  Prefill is already cheap, so lookup and transfer overhead can exceed the saving.

Measure before you build

```
rate(lmcache_mp_lookup_hit_tokens_total[5m])
/
rate(lmcache_mp_lookup_requested_tokens_total[5m])
```

No universal target. Compare it to the share of tokens you **know** repeat - sitting far below that is a bug, not underperformance.

Hit rate is a property of the workload, not the engine.

PRODUCTION · PICK YOUR RUNG

Three rungs, not one leap

01

LMCache, one node

Offload past the GPU to CPU RAM and local disk. Survives eviction and engine restarts.

PIP INSTALL · ANSWERS EVICTED + KILLED

02

Add a shared tier

An RWX volume or a RESP store every node can read. Redeploys stop starting cold.

ANSWERS ORPHANED · NO OPERATOR REQUIRED

03

Add TMO

A reconciled cache fleet with tenants, quotas, per-tenant metrics and matched releases.

HELM OCI · GITOPS · OPENSHIFT · AIR-GAPPED

> Rungs one and two are **entirely open source**. Start there - you'll know you've outgrown them.

PRODUCTION · THE OPERATOR

TMO - day-2 ops for the KV cache

Rung three. TMO doesn't make the cache faster - LMCache does that. It makes it **durable, shareable, diagnosable and upgradeable**: the four things a library can't own. Upgradeable first.

`One CR → a fleet`	The operator reconciles a per-GPU-node cache DaemonSet; vLLM attaches over **same-node shared-memory IPC**, with no network hop.
`Status is an API`	Continuously reconciled, with PHASE and READY written back to the CR - machine-checkable and CI/CD-gateable.
`Validated resilience`	Engine, coordinator, operator and vLLM-pod kill scenarios, **validated per release**.
`Pinned, matched sets`	Engine, operator, chart and serving image ship as one validated unit - never `:latest`.

TMO - BETA | TMI (HOSTED) - GA

PRODUCTION · TIERED CACHE

L1 host RAM plus a durable L2

- **L1 - host memory**

  The hot working set, and the fastest warm hits.

- **L2 - durable, and shared**

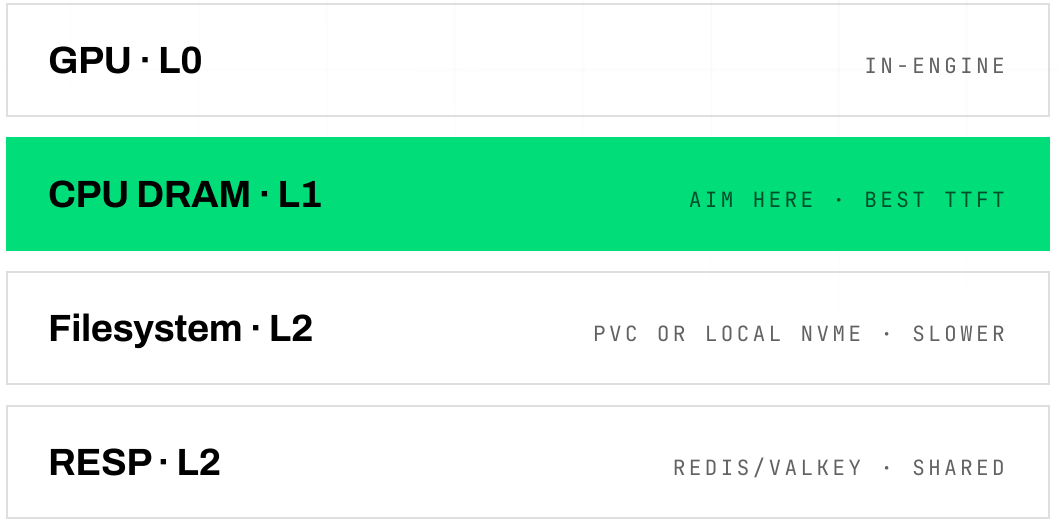
  An RWX PVC every node can read, node-local NVMe, or RESP (Redis/Valkey). It outlives the pod, and a node that never saw turn 1 can still serve turn 2.

- **Sizing signal**

  The reuse-gap histogram - time between reuses of the same content - tells you how big L1 needs to be.

- **Placement is the signal**

  Mostly L1 is best TTFT; mostly L2 is useful but slower; mostly miss means the workload, not the cache, is the problem.



PRODUCTION · MULTI-TENANCY

A governed cache, not just a faster one

Who may share	`cache_salt` per request: same model, prefix and salt → reuse. Different salt → **different cache keys that never collide**.
vLLM-native	The salt folds into **vLLM's own prefix hash** at token 0; TMO holds that isolation across L1 and L2. It also closes cross-tenant cache-timing probes.
No noisy neighbours	IsolatedLRU plus per-tenant quotas via a **runtime admin API**, no redeploy. Eviction victims come only from the offending tenant.
Chargeback	Per-tenant hit-rate and usage metrics - QoS verification and billing evidence.

> vLLM gives the hook - **TMO adds the governance.**

PRODUCTION · OBSERVABILITY

Hit rate becomes an SLO you can hold

The pipeline

- **OpenTelemetry-native**
  Engine → OTLP → Collector → Prometheus and Tempo. A ServiceMonitor ships with it.
- **Per-model and per-tenant dimensions**
  The same cut you need for QoS checks and chargeback evidence.

Watch these first

- **Lookup hit rate**
  Alert on it. A drop is a regression, the same way latency is.
- **L1/L2 activity and the reuse-gap histogram**
  The sizing signal from earlier, now a panel you can watch - not a one-off study.

The 0% mystery was found on exactly these graphs - **in an afternoon.**

CLOSE · BE HONEST

Open source vs. the product

OPEN SOURCE

- vLLM · LMCache · CacheGen · the production-stack
- CacheBlend - published research, EuroSys '25
- The operators and CRDs

WHAT TENSORMESH ADDS

- CacheBlend, productized - non-prefix reuse in production
- Multi-tenancy: salts, quotas, per-tenant metrics
- Validated matched releases · OpenShift and air-gap · support and SLA · TMO (Beta)

> The moat is operations - running the open science reliably, not a secret cache trick.

CLOSE · TAKE HOME

Five things

01 **The KV cache is state** - the most expensive bytes your GPU makes. Treat it as data, not scratch.

02 **Hit rate is a workload property.** Measure it, never assume it. Remember the 0% mystery.

03 **Reuse isn't only prefixes.** For RAG the money is mid-prompt - that's CacheBlend.

04 **Caching is economics.** It pays when reuse × bandwidth × model size say so, and not otherwise.

05 **Production caching is an ops problem** - tenancy, quotas, observability, lifecycle. That's TMO.

CLOSE · GO DEEPER

Resources & links

LMCache	github.com/LMCache/LMCache
production-stack	github.com/vllm-project/production-stack
vLLM engine	github.com/vllm-project/vllm
Contact	shaw@tensormesh.ai

START TONIGHT

vLLM + LMCache - all open source.

Talk to us when the cache becomes **infrastructure**.

#VLLMBANGKOKDAY · #LMCACHE

Thank you.

Questions?

SHAW NGUYEN · TENSORMESH SHAW@TENSORMESH.AI START WITH VLLM + LMCACHE - ALL OPEN SOURCE

tensormesh