Towards a Solution to the Management Scaling Paradox in Distributed LLM Inference

Amir Noohi, Bita Asoodeh, Antonio Barbalace

University of Edinburgh

EuroMLSys '26, Edinburgh, April 27, 2026

Why Disaggregated LLM Serving?

LLM inference has two phases with opposite resource profiles:

Prefill

Processes the whole prompt in one pass

Compute-bound (GPU FLOPs)

Output: **KV cache**

Decode

Generates tokens one at a time

Memory-bandwidth-bound

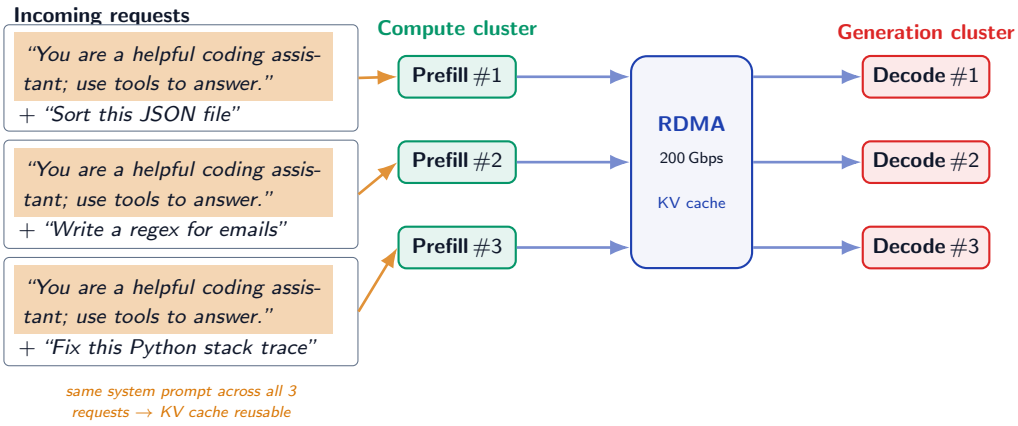
Reads: full KV cache per step

When colocated, prefill and decode *interfere* — tail latency explodes.

Solution: place them on **separate nodes**.

Zhong et al., DistServe (OSDI 2024); Patel et al., Splitwise (ISCA 2024); Qin et al., Mooncake (FAST 2025).

The Cross-Node KV Cache Problem



Prefix caching reuses KV across requests

Shared system prompts, few-shot, RAG context

Production hit rate: **45–71%**

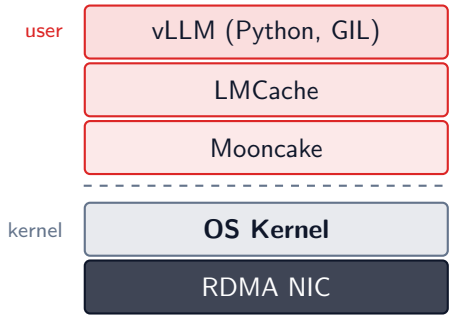
Current Stack: LMCache + Mooncake atop vLLM

LMCache

Chunk identification · multi-tier storage
(GPU → CPU → disk → remote)

Mooncake

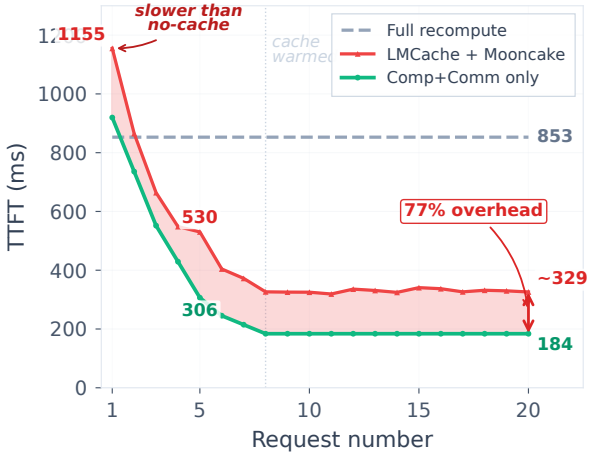
Centralized metadata (Conductor)
RDMA transfer engine (NIXL)
Deployed in production (Kimi)



Everything runs in **user space**

LMCache (github.com/LMCache); Qin et al., Mooncake (FAST 2025); Kwon et al., vLLM PagedAttention (SOSP 2023).

Reality Check: We measured this



Llama-3.1-8B · 7,680 tokens
80% hit · 200 Gbps RDMA

- Full recompute **853 ms**
- Ideal (comp + comm) **184 ms**
- LMCache + Mooncake **329 ms**

Paradox: 4.6× possible but stuck at **2.6**×

Where does the overhead come from?

Multiple causes, all rooted in user-space design choices:

Redundant layering

Every op hits the kernel anyway (RDMA MR, page allocator, CPU buffers).
User-space just *adds* work on top.

Centralized coordinator

Mooncake's Conductor is a *single* RPC bottleneck — 2 HTTP + 1 ZMQ round-trips *per transfer*.

Synchronization & locking

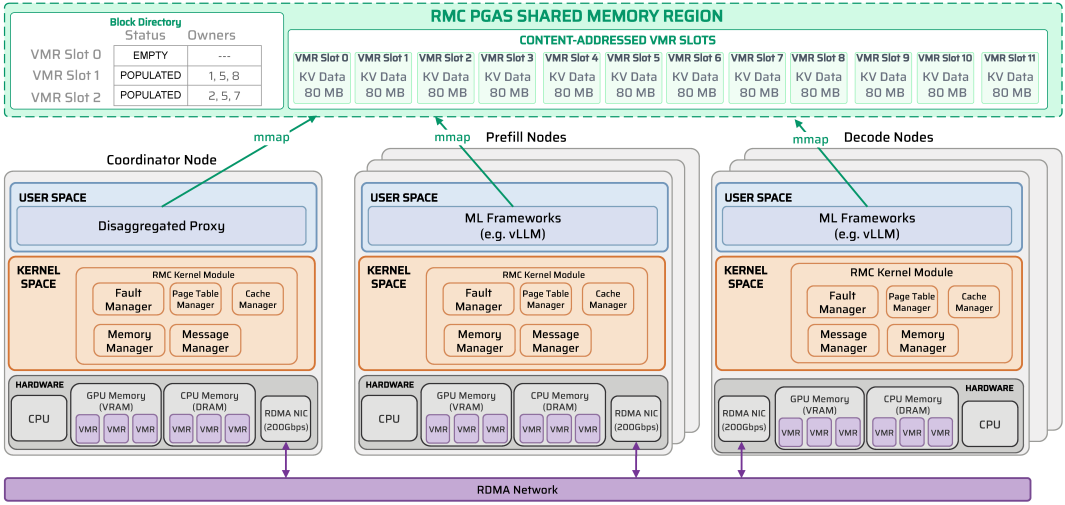
SHA-256 hashing serialized under Python's GIL. $4N$ mutexes per retrieve. `cudaStreamSync` per chunk.

Contention at scale

Every node hits the same directory service; hot prefixes create request storms on the coordinator.

vLLM already uses `/dev/shm` locally — why not extend it cluster-wide?

RMC: content-addressed PGAS in the kernel



Evaluation Setup

Infrastructure

Nodes	1 prefill + 1 decode + 1 coordinator
GPU	2× NVIDIA RTX 4090 (24 GB)
CPU	Intel i7-9700K
RAM	32 GB DDR4
NIC	BlueField-2, 200 Gbps RDMA
Model	Llama-3.1-8B-Instruct (BF16)
Layout	32 layers, 8 KV heads, 128-d
Chunk	256 tokens, 32 MB
Serving	vLLM 0.15.1
Baseline	LMCache 0.3.14 + Mooncake 0.3.9
Metric	TTFT (ms)

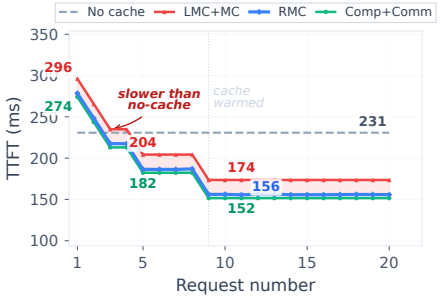
Workloads (production-representative)

ID	Tokens	Chunks	Hit
W1 Multi-turn Chat	2,048	8	50%
W2 RAG Retrieval	4,096	16	44%
W3 Code / Agent	6,144	24	54%
W4 Long Context	7,680	30	63%

Compared systems

- Full recompute – no caching (colocated)
- LMCache + Mooncake – user-space SOTA
- **RMC** – in-kernel PGAS (ours)
- Comp + Comm – lower bound (instrumented)

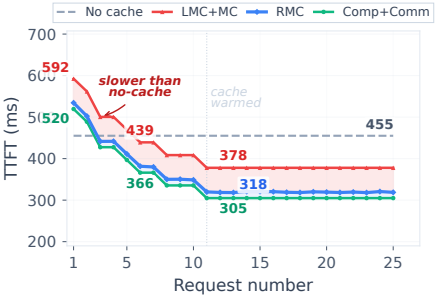
Evaluation: short-context workloads



W1 – Multi-turn Chat

2,048 tokens · 50% hit

10% improvement vs LMC+MC



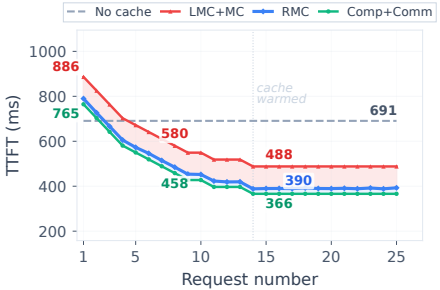
W2 – RAG Retrieval

4,096 tokens · 44% hit

16% improvement vs LMC+MC

RMC stays close to the **compute+comm lower bound**;
LMC+MC pushes it up to the **upper bound**.

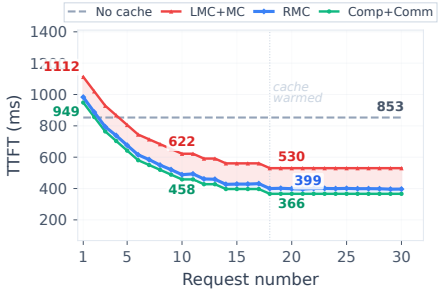
Evaluation: long-context workloads



W3 – Code/Agent

6,144 tokens · 54% hit

20% improvement vs LMC+MC



W4 – Long Context

7,680 tokens · 63% hit

24% improvement vs LMC+MC

Gap **widens** at longer context: more cached chunks
→ more per-chunk overhead for LMC+MC → bigger RMC advantage.

Thank you

Questions?

amir.noohi@ed.ac.uk