SNIA DEVELOPER CONFERENCE
SDC 25
By Developers FOR Developers

Hyatt Regency Santa Clara, CA
September 15-17, 2025

# KV-Cache Storage Offloading for Efficient Inference in LLMs

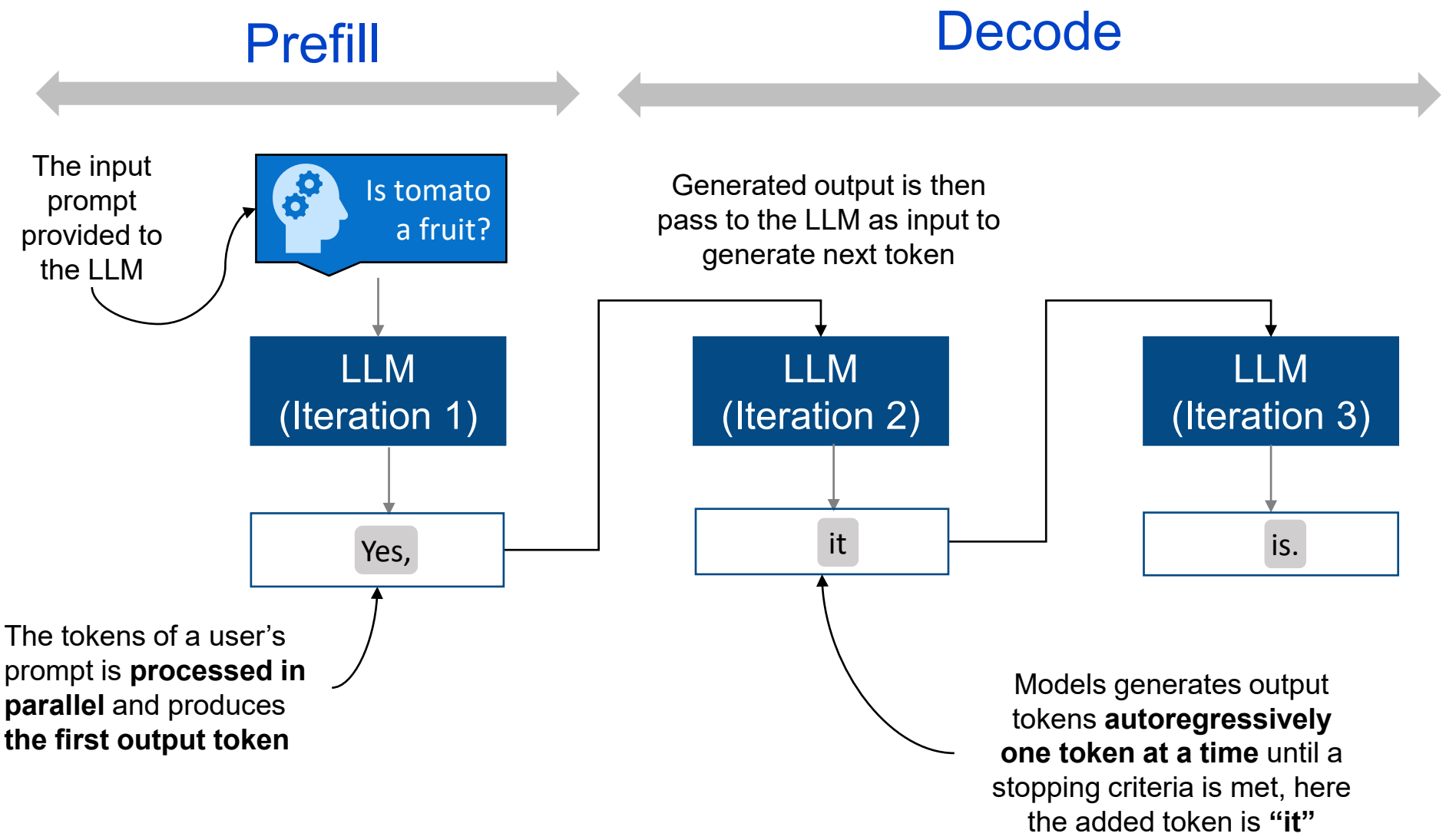
Ugur Kaynar
Dell Technologies
Technical Staff – Office of Storage CTO

# Outline

- How inference works
- What is KV-Cache and Why We Need It
- Storage and Performance Requirements of KV-Cache
- Current Software Stacks
- Remote Storage IO

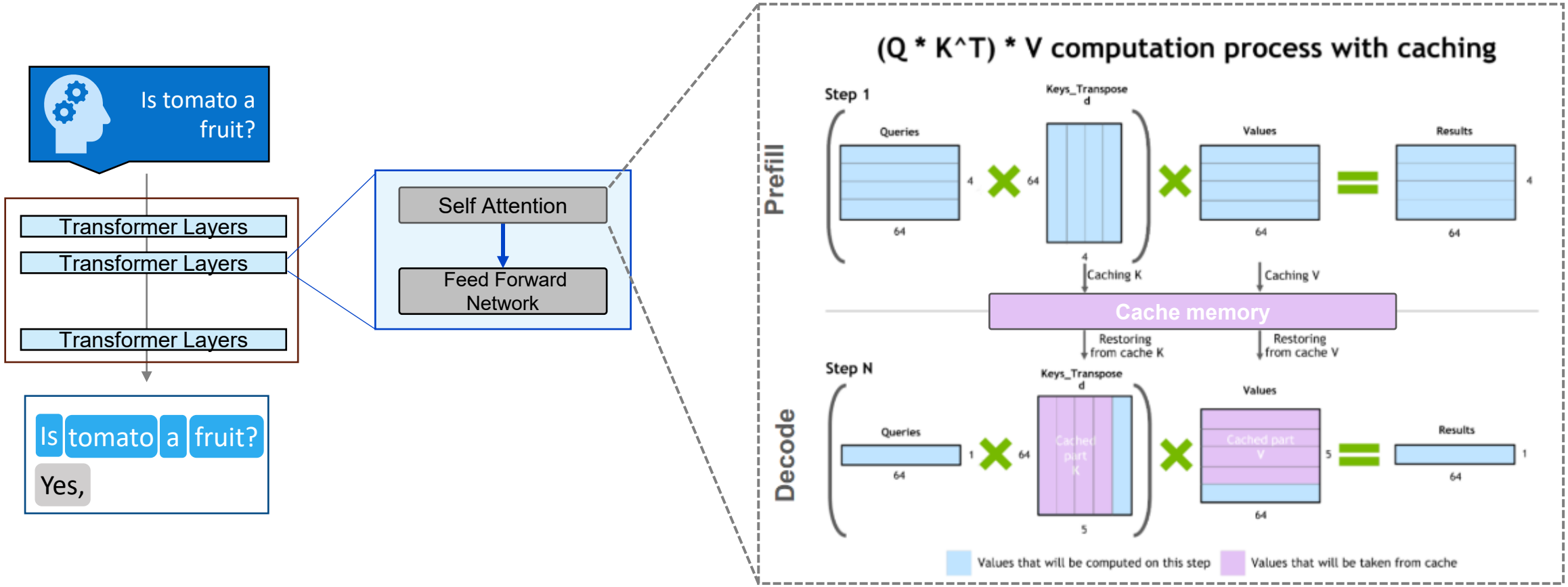
# LLM Inference Workflow



# Transformer and KV Cache

Instead of recomputing attention for all previous tokens at every step, **KV-Cache stores the key and value vectors** from earlier tokens. It only computes attention for the **new token**.

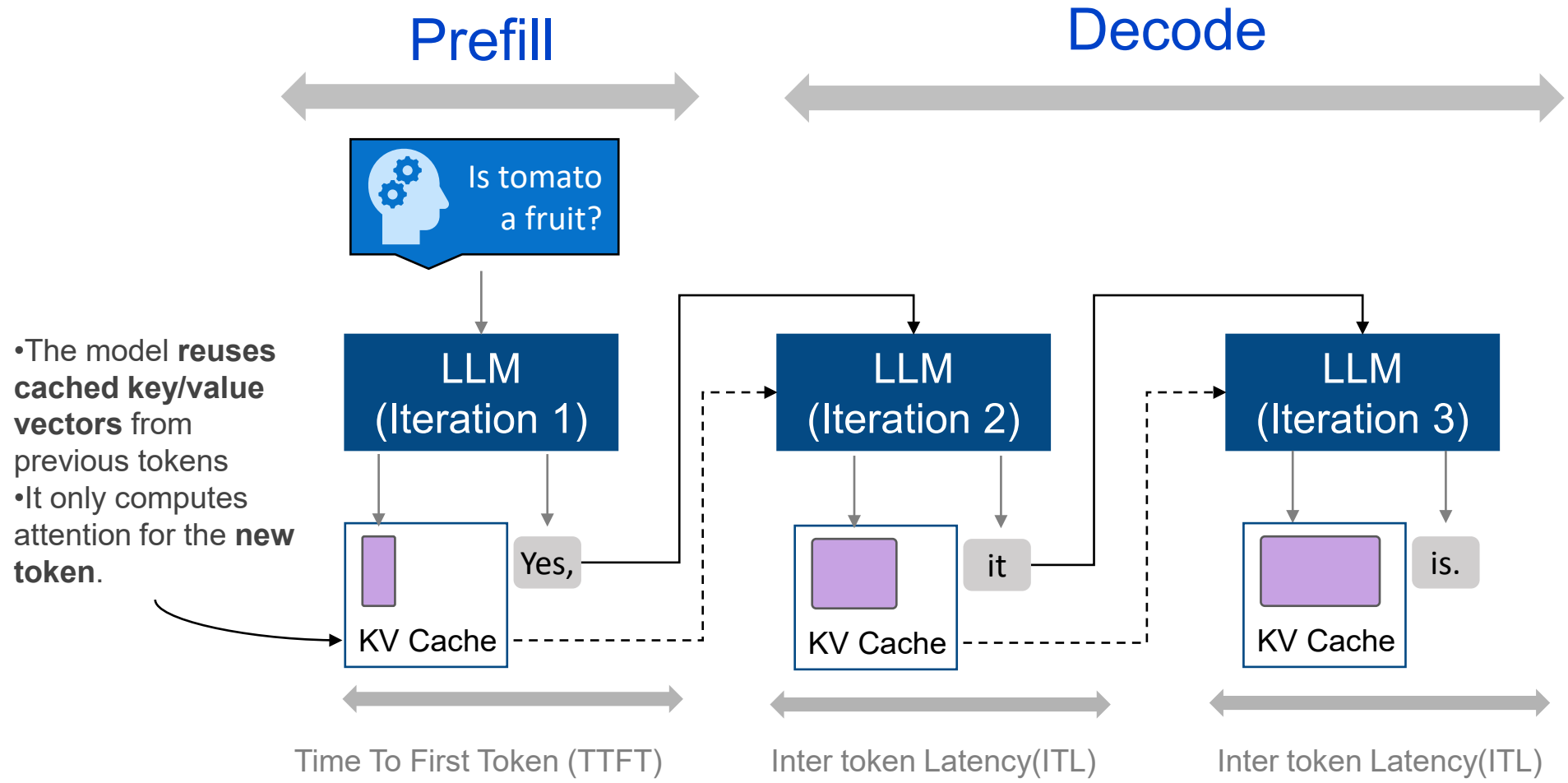



**Reference**

[1] LayerKV: Optimizing Large Language Model Serving

# KV Cache is an optimization for transformer based LLM inference



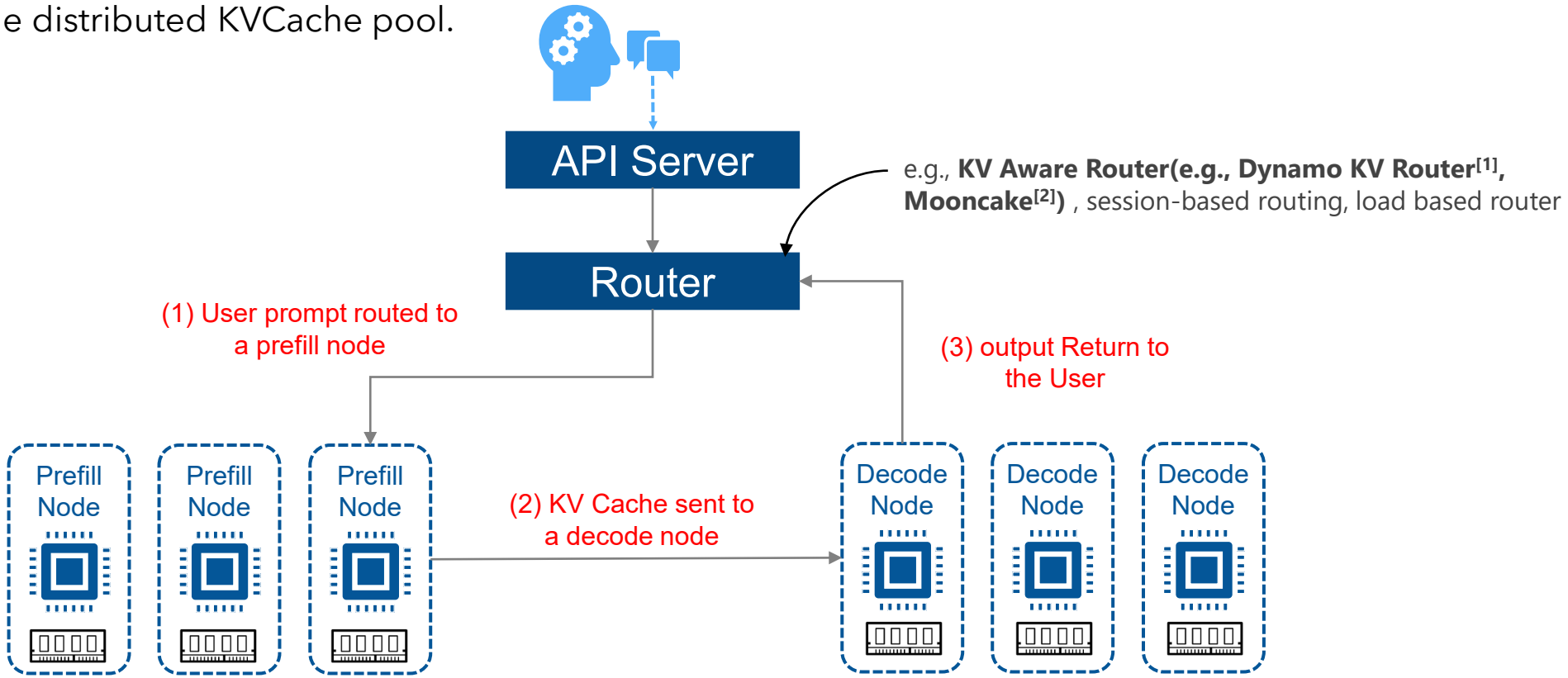

- The model **reuses cached key/value vectors** from previous tokens
- It only computes attention for the **new token**.

- *Prefill Phase*: A **compute-bound** task where input tokens (prompt and context) are processed to build KV cache.
- *Decode Phase:* A **memory-bound** task where tokens are generated one at a time using KV cache.

# Today, Large-scale Inference Deployments Increasingly Rely on a Disaggregated Architecture

**Disaggregated architecture** maximizes GPU utilization and helps to boost the performance. The architecture is typically centered around the distributed KVCache pool.




**Reference**


[1] KV Router — NVIDIA Dynamo Documentation
[2] Mooncake: Trading More Storage for Less

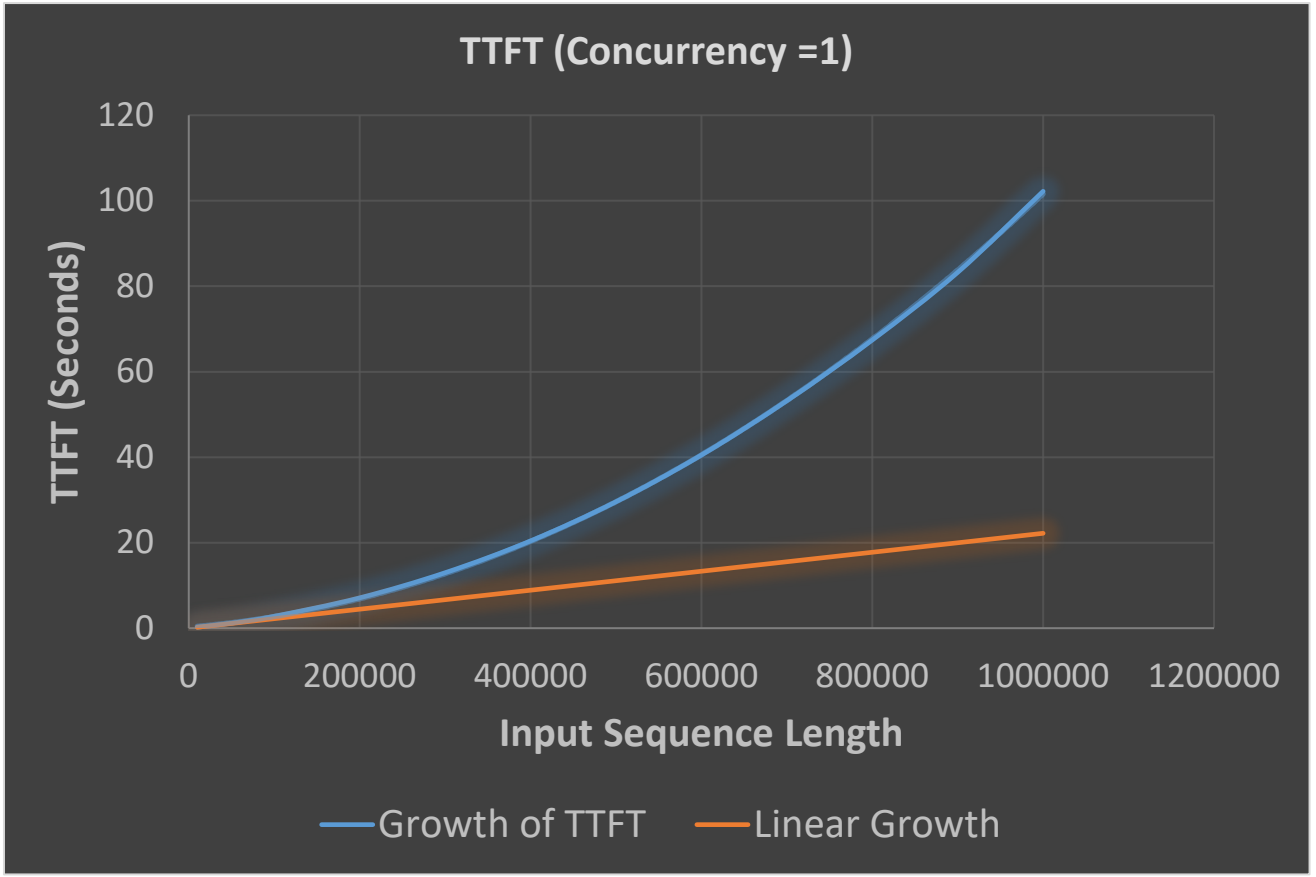
# Why Do We Need KV Cache in LLM Inference?

## KV Cache Computation is very expensive

- Llama4 Maverick (400B  parameters – Mixture of Expert)
  - 128 Experts - 17B (Active per expert)
  - Maximum context window of 1,000,000 tokens
- **The blue curve** shows the **growth** of TTFT as the context length scales up to 1 million tokens, based on measurements from our testbed.
- The **orange line** shows **linear growth.**

Compute complexity grow quadratically with context length[1].
Linear read of KV cache from storage can reduce the time complexity of token generation

- There is a strong focus on reducing KV computation complexity (e.g., Sliding Window Attention, Sparse Attention, Chunked Attention)




Note: These figures are preliminary; we're actively testing to better characterize the system


**Reference**
[1] LayerKV: Optimizing Large Language Model Serving with Layer-wise KV Cache Management

# Long Context Length and Real-World Use Cases


- Chatbot (Multi-turn conversation)
- Document Summarization
- Code Analysis
- Retrieval Augmented Generation (RAG)
- Multi Modal

**Increasing the content length directly increases prefill time** in LLMS, which can degrade the user response time and increase the cost.

Instead of generating the KV cache for every query, we should generate it once and then reuse it for successive queries.

# Capacity Requirement of KV Cache for LLM Inference

- Storage requirements of KV-cache depend on several model-specific factors: e.g, the number of layers, KV heads, precision, and whether full attention is used

**KV Cache Size Per Token =** 2 * number of layers * number of KV heads * dimension of head * precision

- The table shows memory requirement for KV cache with varying number of sequence lengths for single session[1].

| Model (FP 16) | 1 token | 1,000 tokens | 10,000 tokens | 100,000 tokens | 1,000,000 tokens |
|---|---|---|---|---|---|
| Qwen3-8B | 150 KB | 0.15 GB | 1.47 GB | 14.75GB | 147 GB |
| LLaMA-3.3-70B-Instruct | 330 KB | 0.33 GB | 3.28 GB | 32.77 GB | 327 GB |
| LLaMA-3.1-405B | 516 KB | 0.516 GB | 5.16 GB | 51.6 GB | 516 GB |

***One Token's KVCache (Tens of KB)***

***KVCache linearly increases with sequence length size for models with full attention [2,3].***


**Reference**

[1] KV cache Calculator - a Hugging Face Space by gaunernst

[2] InfiniGen: Efficient Generative Inference.pdf

[3] LayerKV: Optimizing Large Language Model Serving

# The storage capacity increases when inference serves concurrent requests.

| Model (FP 16) | 1 token | 1,000 tokens | 10,000 tokens | 100,000 tokens |
|---|---|---|---|---|
| LLaMA-3.3-70B-Instruct | 330 KB | 0.33 GB | 3.28 GB | 32.77 GB |

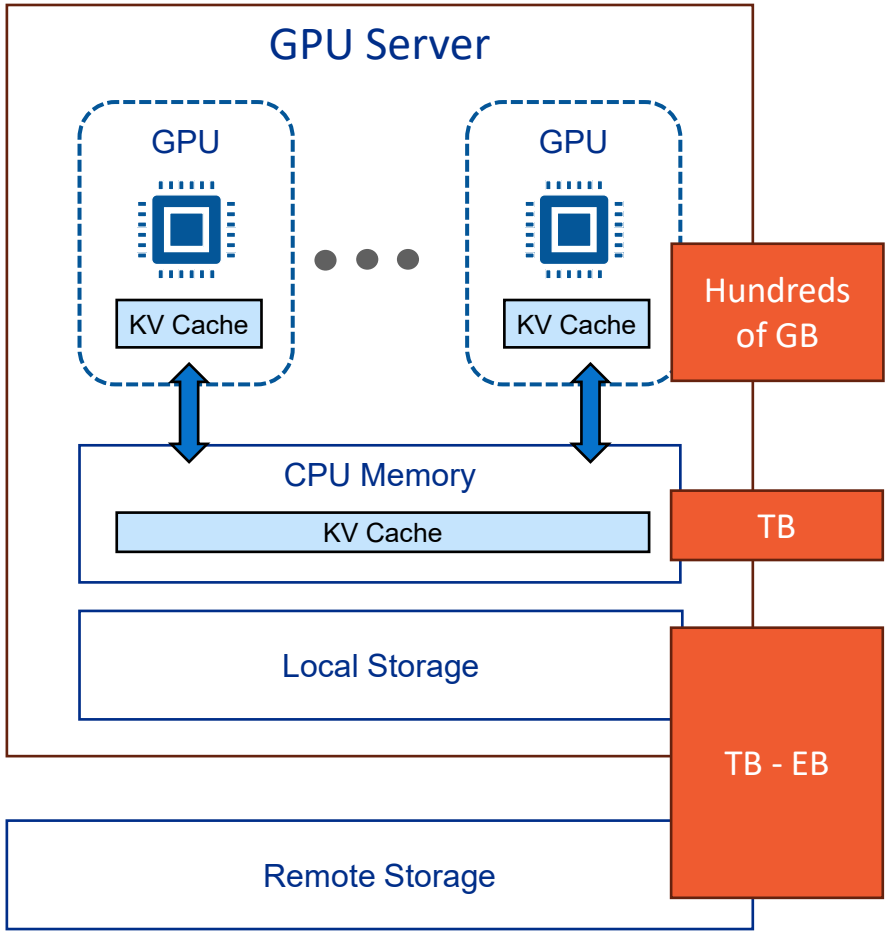
| Token Size | 1 user | 10 users | 100 users | 1,000 users | 10,000 users |
|---|---|---|---|---|---|
| 10,000 tokens | 3.28 GB | 32.7 GB | 327 GB | 3.19 TB | 31.9 TB |
| 100,000 tokens | 32.77 GB | 327 GB | 3.19 TB | 32 TB | 320 TB |

**Reference**

[1] KV cache Calculator - a Hugging Face Space by gaunernst
[2] InfiniGen: Efficient Generative Inference.pdf

# Existing 2-Tier KV Cache Design in LLM Inference




- GPU memory is used as 1st tier.
- Only a portion of GPU memory is allocated for KV-cache.
  - For example, with a 13B parameter model on an NVIDIA A100 (40GB), roughly **30% of the GPU memory** can be used for KV-cache [1].
- Evicted KV blocks from GPU can be saved in system memory.
- GPU memory is local to each GPU, but CPU memory is shared across all GPUs within the same host.

**Reference**
[1]Efficient Memory Management for Large Language Model Serving with PagedAttention

# Why Local in Memory Cache is Not Sufficient

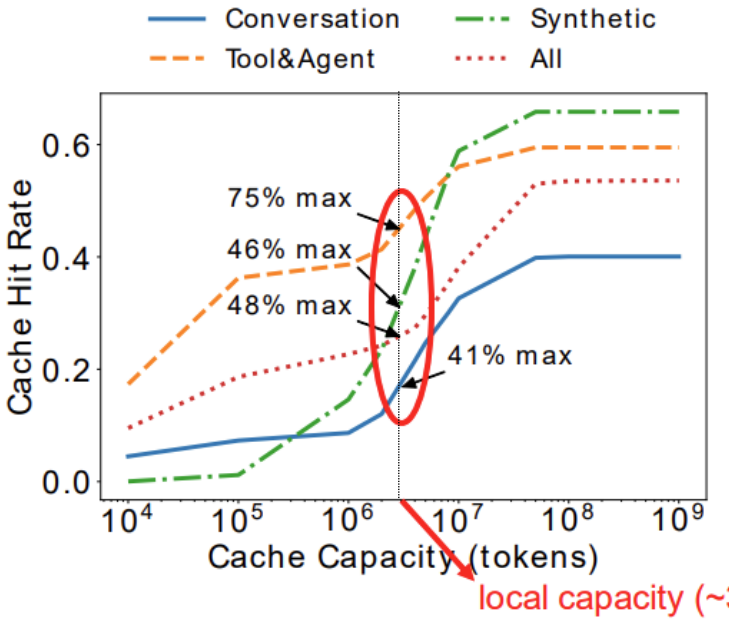
LLaMA3-70B model

Per token KVCache is 320KB

1 TB of DRAM (holds 3M tokens)

- 3M token capacity does not achieve 50% of the theoretical maximum hit rate[1]
- 50M tokens nearly reaches the theoretical maximum hit rate of 100% **(at least 20 DRAM required)**

## Trace Analysis: Is Prefix Caching a Simple Solution?




**Traces in our paper**
(open-sourced in https://github.com/kvcache-ai/Mooncake )

**Conversation**: collected from real-world online conversation requests

**Tool&Agent**: collected from real-world online requests that include tool use

**Synthetic**: synthesized from publicly available long context datasets

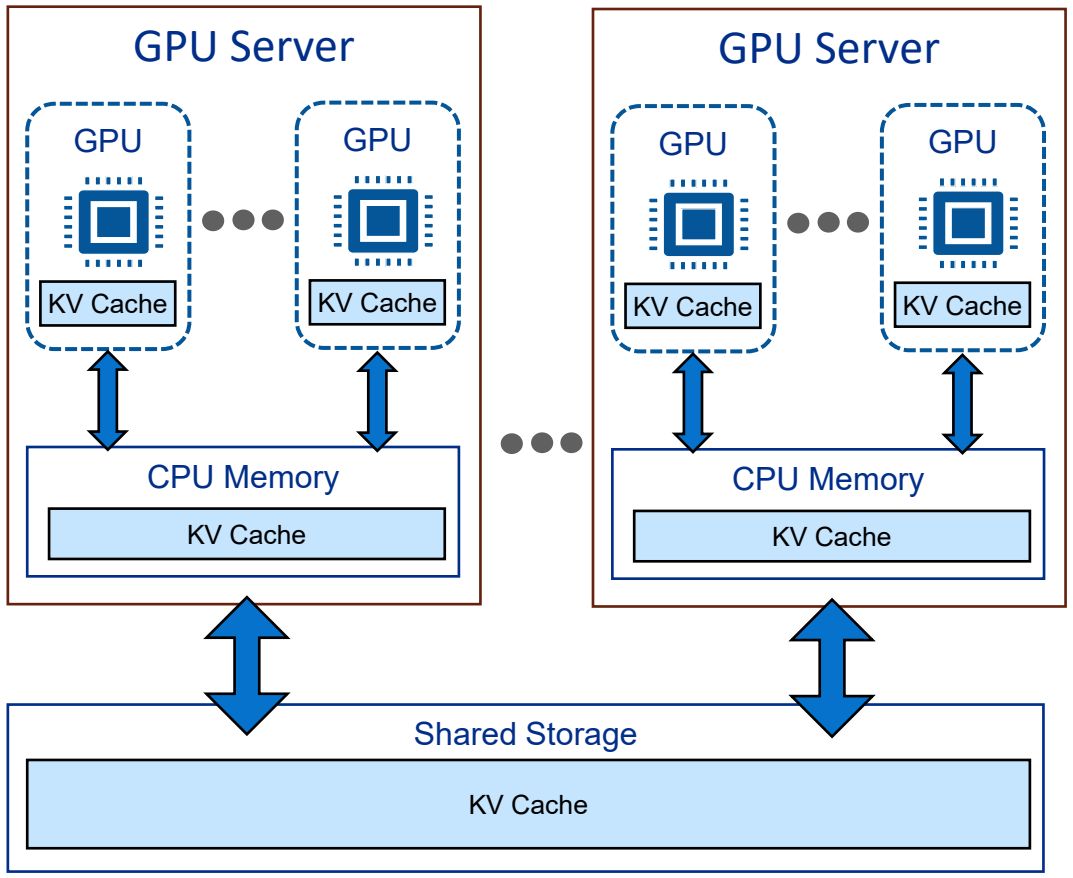
- Around **50%** of the tokens' KVCache in the real-world workloads can be reused
- **However**, the cache hit rate will significantly drop if only using the local cache

**Reference**

[1] Mooncake: Trading More Storage for Less Computation — A KVCache-centric Architecture for Serving LLM Chatbot

# Why do we need shared storage for KV cache?




- **Supports petabyte to exabyte-scale storage capacity**
  - Even if one session fits in memory, hundreds or thousands of concurrent sessions. (e.g., Deepseek repoted processing ~600B input tokens per day[1])
  - Offloading allows session isolation and scaling beyond memory limits.
- **Shared KV Cache Across LLM Instances**
  - Enables cross-node access and shared state
  - Eliminates redundant data duplication across nodes.
  - Local storage (e.g., NVMe on the same node) doesn't support cross-node access or shared state.
- **Session Persistence and Multi-Tenant Support**
  - In multi-user environments (e.g., chatbots, agents, RAG systems), sessions need to be paused, resumed, or migrated across nodes.
  - Remote storage enables session persistence, allowing KV-Cache to be offloaded and reloaded as needed.

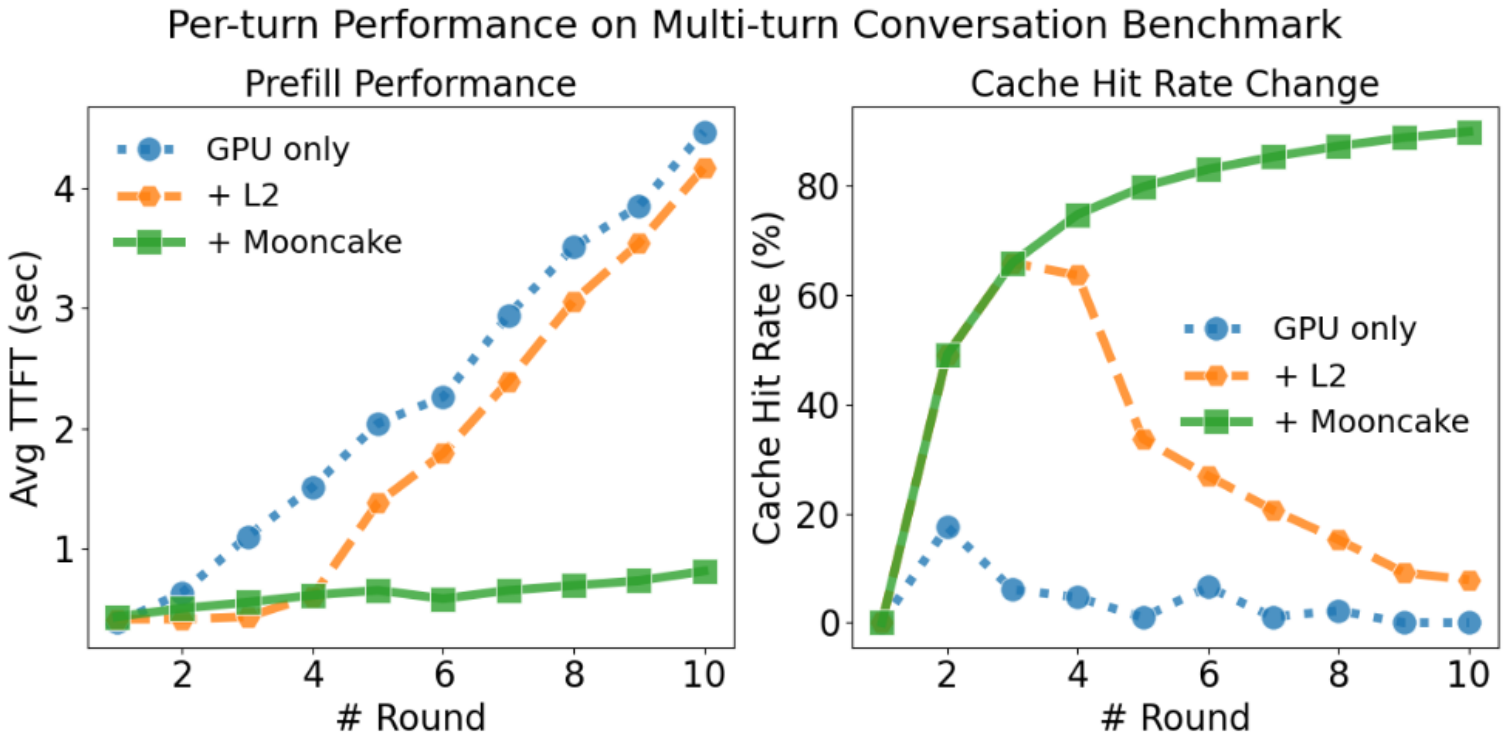
**Reference**

[1] Day 6: One More Thing, DeepSeek-V3/R1 Inference System Overview

# Recent benchmark numbers from SGLang highlight the advantages of hierarchical caching[1]

- HW: : 8 × H800 GPUs, 8 × mlx5 RDMA NICs, 2 NUMA nodes.
- Model: Qwen3-235B-A22B-Instruct-2507
- Benchmark: SGLang's multiturn benchmark
- **GPU only**
- **(HiCache L1) + L2**: KV cache stored across both GPU & CPU
- **(HiCache L1 + L2) + Mooncake backend**: KV cache stored across GPU, CPU, and Mooncake storage.

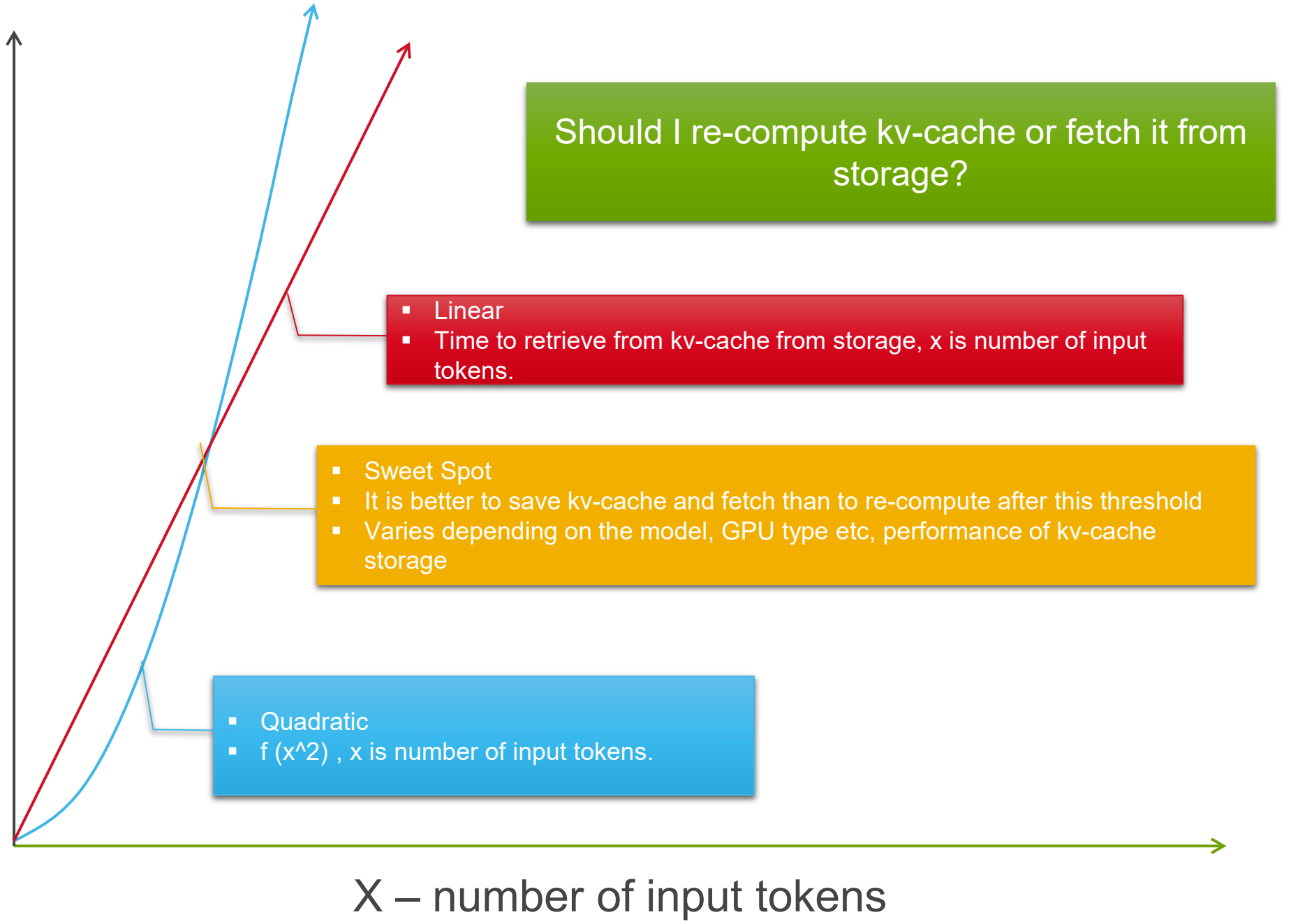



The KV cache hit rate plays a critical role in prefill performance. When the cache is hit, the TTFT is significantly lower compared to the cache-miss case

**Reference**

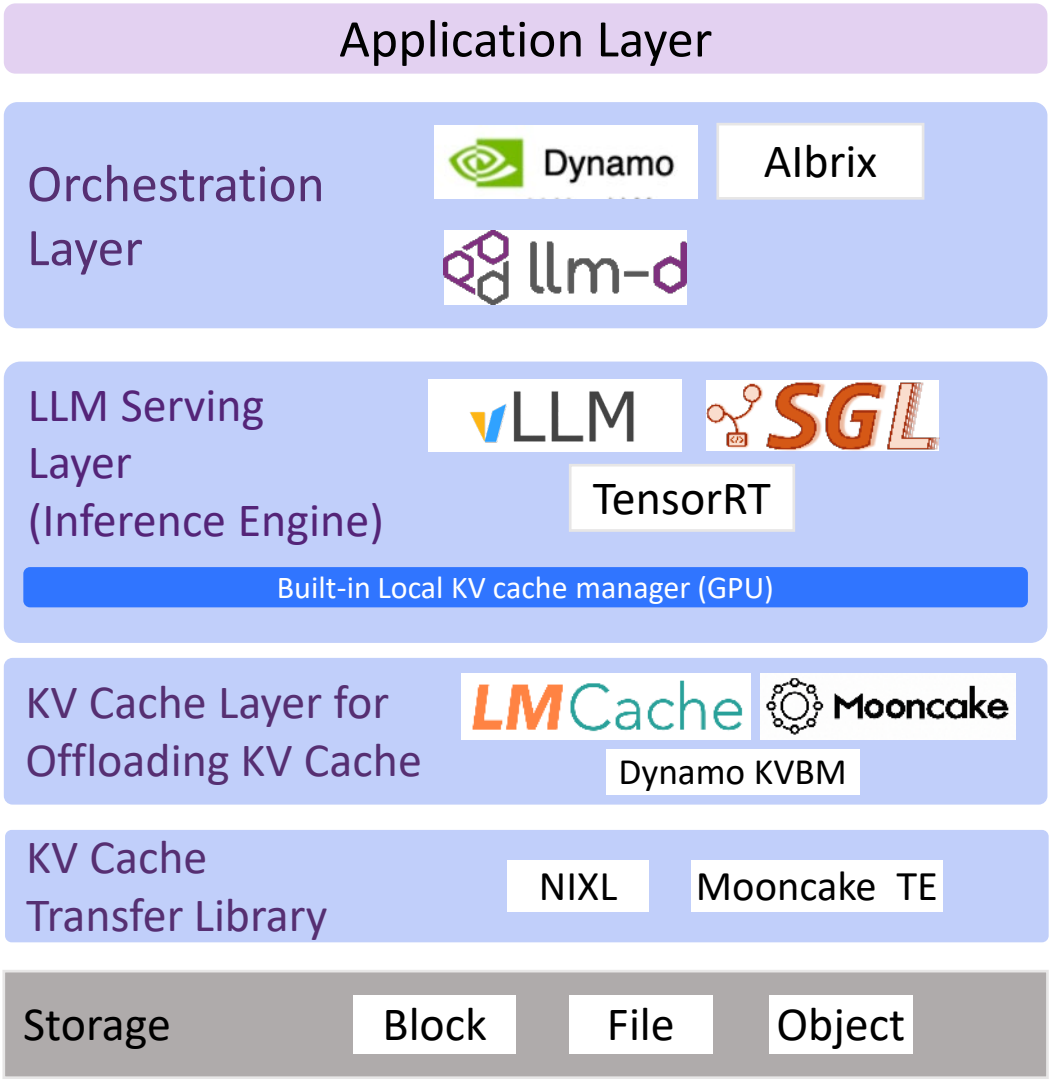
[1] SGLang HiCache: Fast Hierarchical KV Caching with Your Favorite Storage Backends



# AI Inference Stack: From Orchestration to Storage




- Deploy and manage LLM inference in multi-node distributed environments handles distributed inference, allocate GPU resources, and request routing.
- It shares state and scales horizontally—typically using Kubernetes.

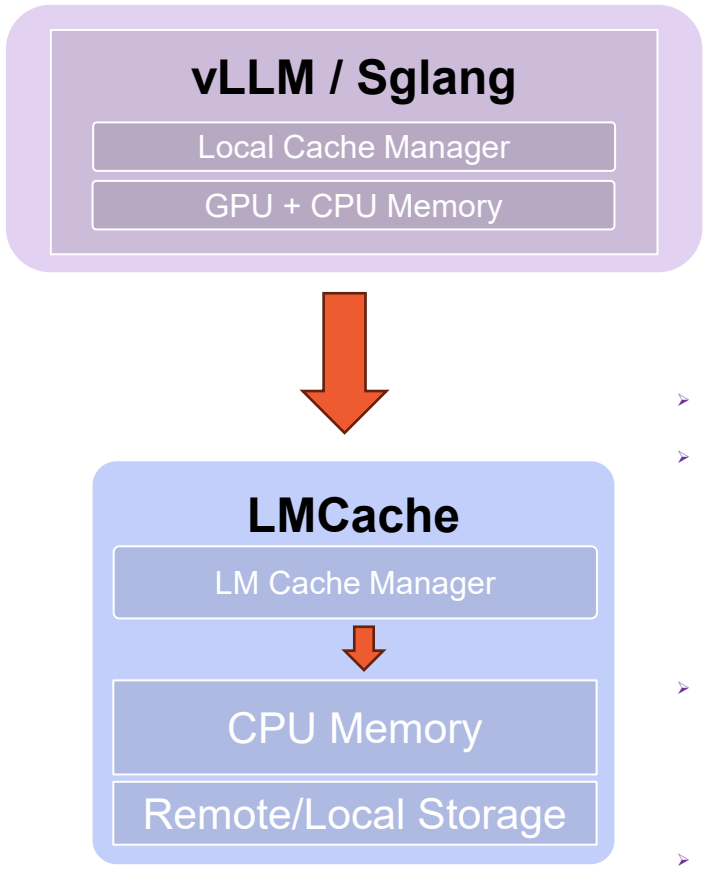
- Optimize inference for a single model instance, with built-in support for batching and memory management.
- A local (not global) KV cache manager optimized for GPU (some also support CPU offloading)
- **No support for local/remote storage offloading.**

- Provide globally multi-tier cache management across CPU and storage, with support for shared caches across nodes.

- Enables fast, low-latency movement of KV cache between GPUs, CPU memory, and storage systems.

# Cache Tiering and Management

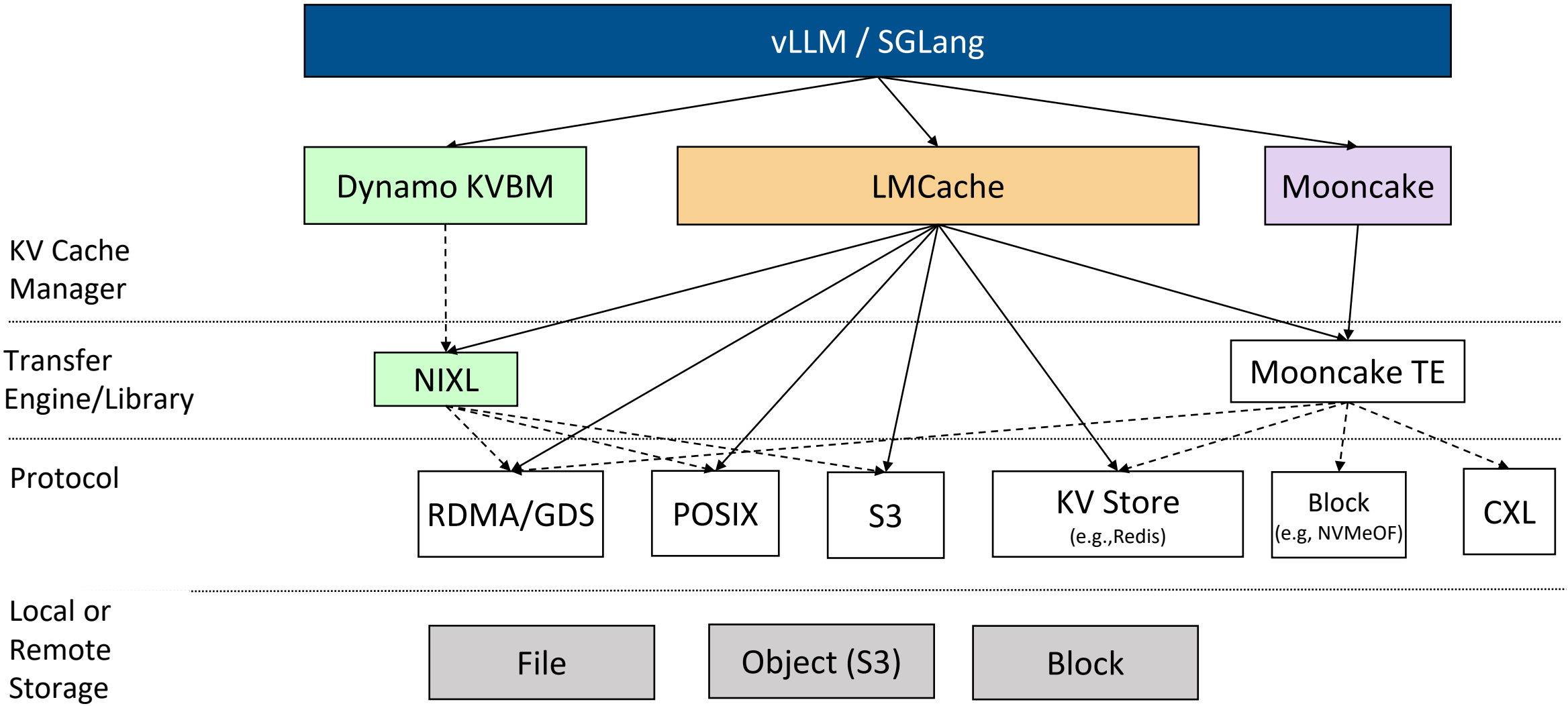



- Inference engines like vLLM and SGLang have built-in cache management, using **GPU memory as L1** for KV cache and **CPU memory as L2** via offloading.
- **Local Cache Management**: Cache management is local, with no global visibility into KV data on other GPUs or remote workers.
- These inference engines provide **abstraction classes** to support **remote storage offloading**, enabling integration with external memory layers beyond local HBM or CPU caches.

- **Cache Tiers (L2 + L3):** Tiered cache architecture, CPU memory and Local/Remote Storage
- **Cache Management:**
  - **vLLM:** Write-through - all data in HBM is offloaded to LMCache.
  - **Sglang:** Write-through, write-back, write-through-selective
  - **Eviction**: LRU, LFU, FIFO
- **Block-Level Granularity**: Operated on large blocks to optimize read/write performance.
  - Default is 256 tokens, which is independent of vLLMs block size.
  - One file per KV cache chunk (typically, in MB)
- Supports zero-copy KVCache transferring for different storage backends
- **Compression (CacheGen[1] ):**. Compressing KV cache into compact bitstreams, and decode phase loads KV cache into GPU memory and decompress them

# Multiple industry efforts are underway for KVCache acceleration



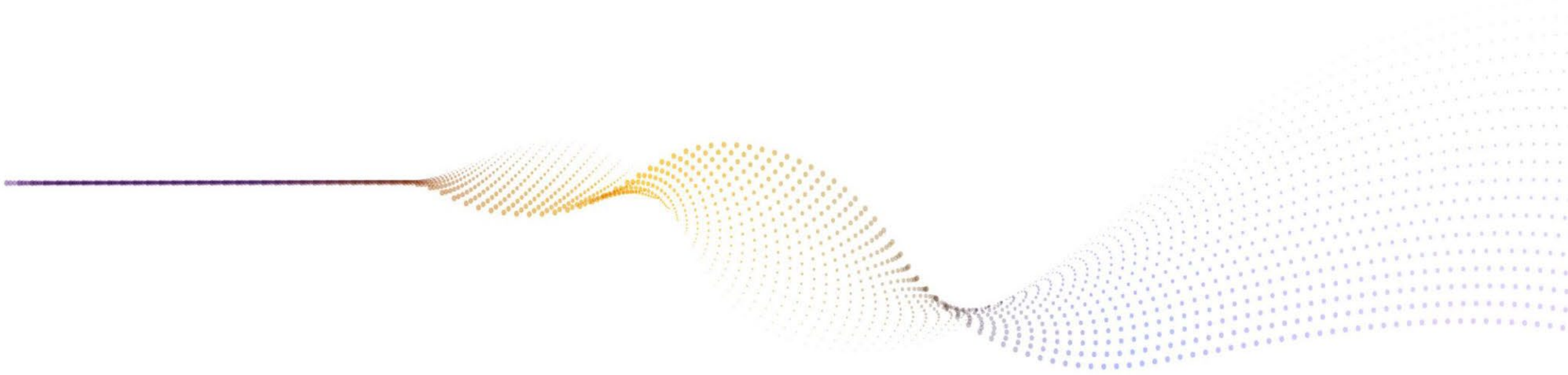

 

# Remote Storage I/O

- **Read-Dominant[1]:**
  - The workload is **heavily read-oriented**, since KV cache offloading mainly retrieves previously computed key-value pairs for reuse in attention.
- **Reads**: On a GPU/CPU cache miss, data is read from storage into GPU memory; an exact prefix match is required.
  - Data size: Fixed-size KV blocks (100 KB–100 MB+)
  - Read pattern: sequential and random read
- **Writes**: Shared storage is used as another tier, data in GPU/CPU memory replicated on shared storage tier.
  - Data size: Fixed-size KV blocks (100 KB to 100 MB+), with offloading libraries optimized for MB-scale I/O.
  - Write pattern: random write, write once, no modification
- **Access granularity:** Block-level, not individual key/values.
- **Latency-Sensitive**: Inference latency is directly impacted by KV retrieval time, so **low-latency access** is critical.
  - **RDMA is critical for reads,** since retrieving tokens and KVs are on the critical path.
  - **RDMA is *beneficial* for writes**, but **not as critical** as it is for reads in most inference workloads.
  - In current libraries, IO is still **CPU-initiated.**

**Reference**
[1] An I/O Characterizing Study of Offloading LLM Models
[2] InfiniGen: Efficient Generative Inference.pdf



# Thank you for attending!

Please remember to rate this session. You get access the presentations at
http://sniadeveloper.org/conference