Disaggregated Architecture for LLM Inference

Mingxing Zhang @ KVCache.AI

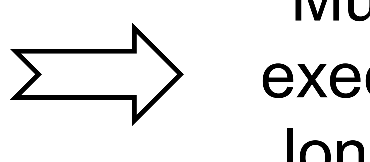
https://github.com/kvcache-ai

Background: Large Language Models (LLMs) Become Agents

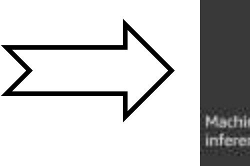
LLM
Tool Result
Tool Call
Workflow & Schedule
manus
Agent (Manus, LangChain)
Execution
Tool A: Web Browser
Tool B: Code Exec
Tool C: Search Engine

LLM
Self-Decision
Tool Using
Web Browser
Search Engine
......
Stop
Output
Result Synthesis

Single-turn, short inputs/outputs



Multi-turn, complex execution topologies, long inputs/outputs.

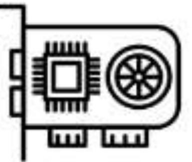


Inference dominates the cost

Challenge of Online Model as a Service System

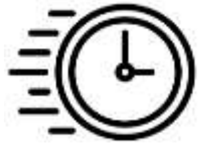
More Data + Larger Model + Longer Context = ☺ Higher Intelligence

BUT



Lack of GPU Supply

Higer Inference Cost



Longer Response Time

How to improve inference MFU? How to reduce the needed TFLOPS?

How to reach the above goals **without** sacrificing user experience?

Different Hardware are Good at Different Dimension

	H800	H20	Xeon SPR + 8 * DDR5-4800
Hardware Spec	80GB VRAM， 3.3 TBps ~ 1 PFLOPS > $ 10,000	96GB VRAM， 4 TBps ~ 200 TFLOPS ~ $50,000	8*64GB DRAM， 8*40GB/s < 20 TFLOPS ~ ¥60,000
Best for	Allround, especially for TFLOPS/$	Bandwidth/$	Capacity/$

!!! The price numbers are not accurate, just a demonstration!

LLM Inference 101: Prefill & Decode

Prefill

Decode

LLM iteration 1

What day is it today?

LLM iteration 2

LLM iteration 3

LLM iteration 4

KV Cache

Today

KV Cache

is

KV Cache

Friday

END

Time To First Token (TTFT)

Time Between Tokens (TBT)

LLM Inference 101: Prefix Caching

- KVCache can be shared across requests with the same prefix, reducing computation

"What day is it today"

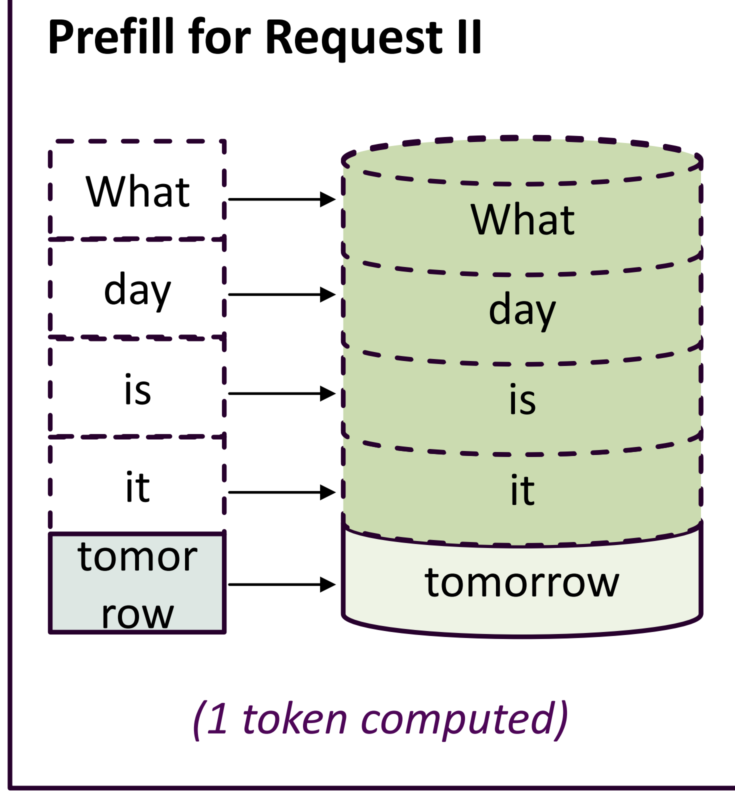
Prefill for Request I

What → What
day → day
is → is
it → it
today → today

(5 tokens computed)

KVCache Reuse

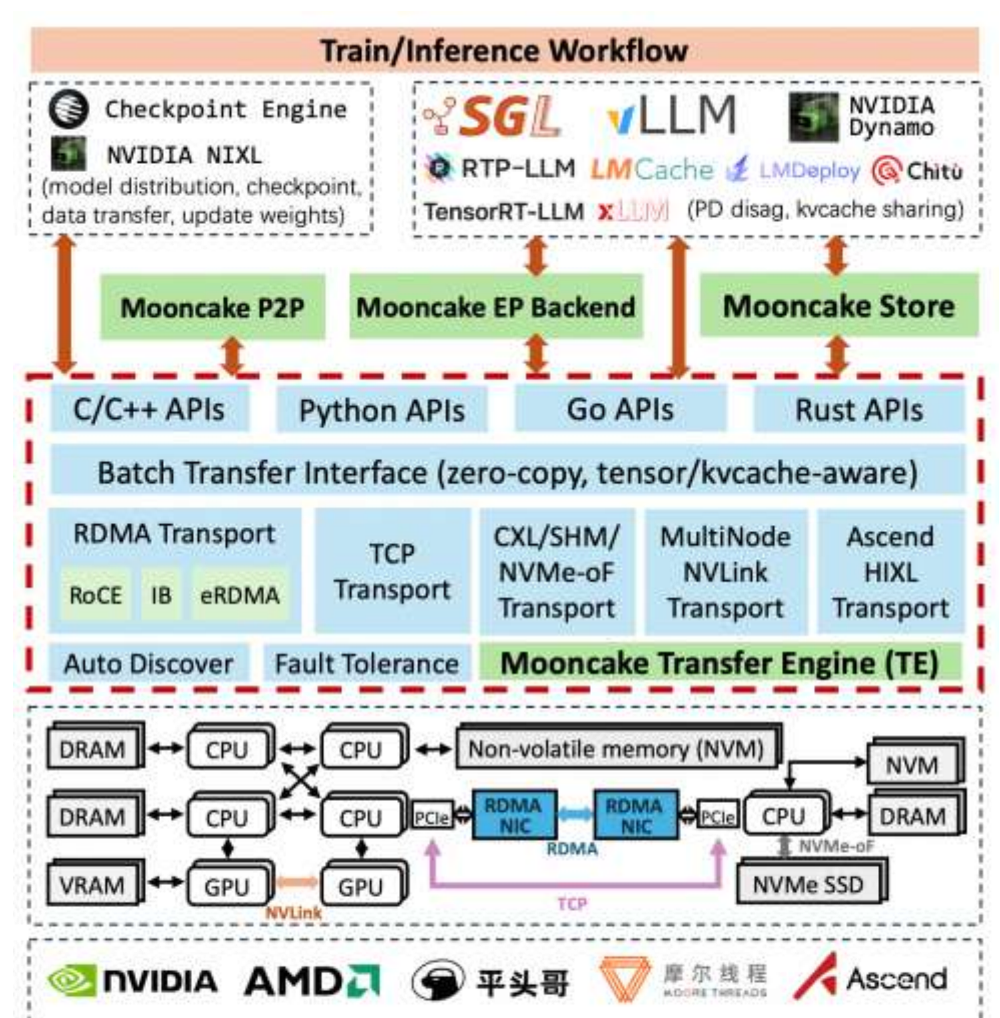
"What day is it tomorrow"



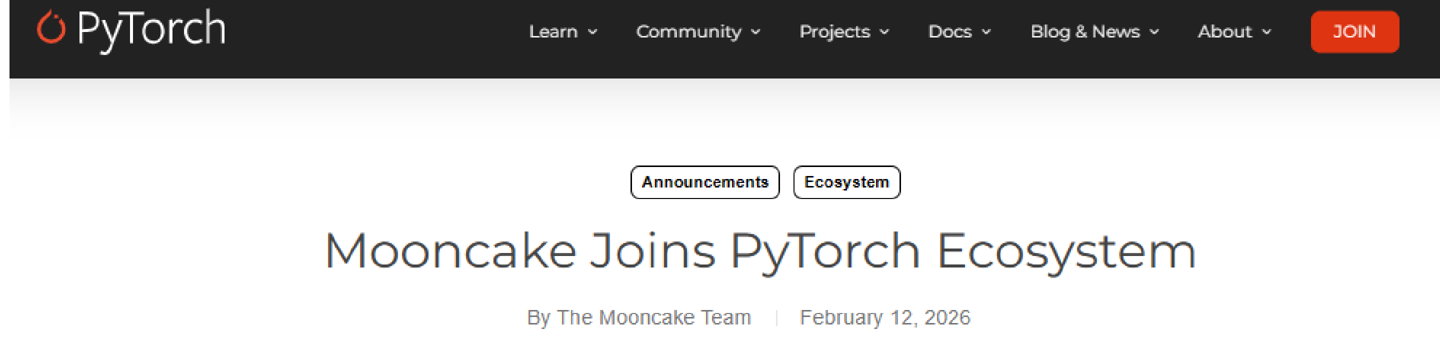
Different Hardware are Good at Different Dimension

	H800	H20	Xeon SPR + 8 * DDR5-4800
Hardware Spec	80GB VRAM，3.3 TBps ~ 1 PFLOPS > $ 10,000	96GB VRAM，4 TBps ~ 200 TFLOPS ~ $50,000	8*64GB DRAM，8*40GB/s < 20 TFLOPS ~ ¥60,000
	For Prefill!	**For Decode!**	**For KVCache!**
Best for	Allround, especially for TFLOPS/$	Bandwidth/$	Capacity/$

!!! The price numbers are not accurate, just a demonstration!



A KVCache-centric Disaggregated Architecture for LLM Serving



Core Technologies of Mooncake

Kimi @ Moonshot AI

Download APP

Log In

KIMI

Ask Anything...

Researcher

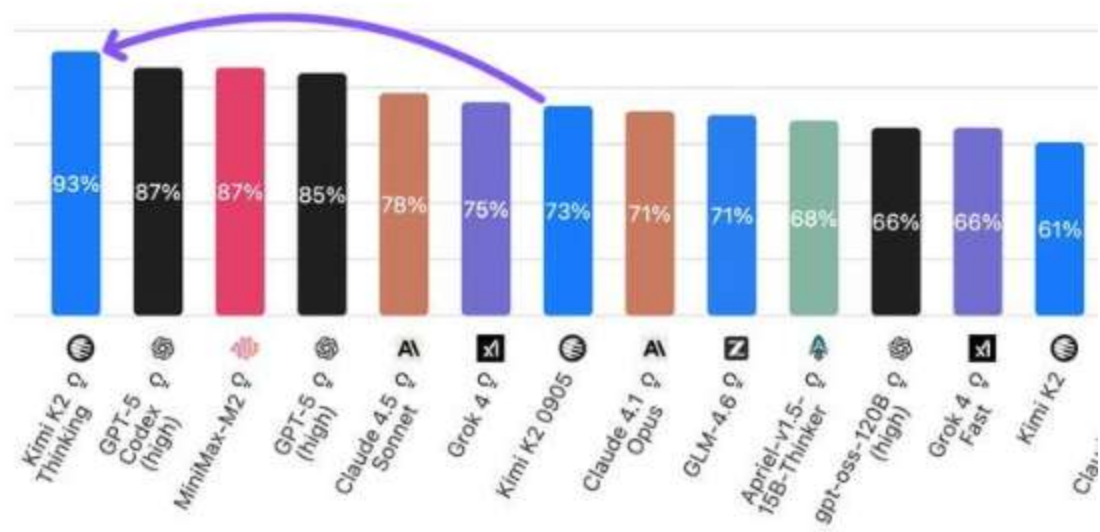
K2

'Another DeepSeek moment': Chinese AI model Kimi K2

作者：E Gibney · 2025 — As with DeepSeek's models, Kimi K2 is open-weight, m

downloaded and built on by researchers for free. It can be accessed through ...

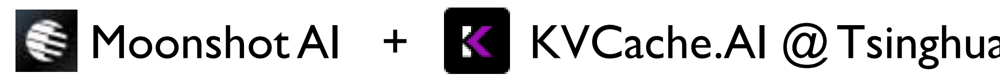
τ^2-Bench Telecom (Agentic Tool Use)



Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving

- The serving platform of Kimi

1. P/D disaggregation architecture centered around the distributed KVCache pool
2. Trading more storage of less compute! Increase the throughput of Kimi by 75%
3. Meet SLO guarantee



More: https://github.com/kvcache-ai/Mooncake

Mooncake (I): 在月之暗面做月饼，Kimi 以 KVCache 为中心的分离式推理架构

KVCache-centric Conductor

Cache-aware Prefill Scheduler

KVCache Balance Scheduler

Load-balance Decoding Scheduler

Prefill Pool

KVCache Pool

Decoding Pool

Prefill Instance

GPU/VRAM

Local Chunked Prefill Scheduler

Paged KVCache

CPU/DRAM/SSD

Distributed KVCache Pool

PP/SP

Prefill Instance

GPU/VRAM

Local Chunked Prefill Scheduler

Paged KVCache

CPU/DRAM/SSD

Distributed KVCache Pool

RDMA

Inter-node KVCache Transfer

CPU/DRAM/SSD

Distributed KVCache Pool

GPU/VRAM

Paged KVCache

Local Scheduler

Decoding Instance

CPU/DRAM/SSD

Distributed KVCache Pool

GPU/VRAM

Paged KVCache

Local Scheduler

Decoding Instance

P&D Disaggregation Becomes a Necessary

- Prefill and decode needs different parallelism strategy, e.g., DeepSeek V3/R1

API Server

Prefill Load Balancer

Decode Load Balancer

Expert-Parallel Load Balancer

Prefill Service

Expert-Parallel Load Balancer

Decode Service

External KVCache Storage (Optional)

知乎 @DeepSeek

KVCache Cache introduces High Challenges to Storage System

- Each 1 token -> 2 * layers * hidden dimension = tens of KB KVCache
- Not only the size of KVCache is large, it also requires high transfer bandwidth to avoid stall of GPU

10B+ Model (GB)

100B+ Model (Hundreds of GB)

TB Model (TB)

KVCache of TB Model (tens of TB)

Reusable KVCache

(Hundreds of TB ~ PB)

Mooncake is Used in Various Scenarios of LLM

EPD Disaggregation
Mooncake TE

Elastic Large EP
Mooncake EP

Distributed KVCache Pool
Mooncake Store

Model Weight Loading
Mooncake TE

PD Disaggregation
Mooncake TE

RL Disaggregation
Mooncake Store
Mooncake TE

Mooncake – Open Sourced and Build with the Community

From flagship applications → To industry-wide adoption

- 2024.3 Kimi went viral for its long-context capabilities, using Mooncake to handle surging traffic
- 2024.6 Mooncake tech report sparked wide industry discussion
- 2024.11 Mooncake open-sourced; adopted by Alibaba and Ant Finance
- 2025.2 USENIX FAST Best Paper Award
- Used in Dynamo, the distributed inference system highlighted at GTC 2025 Keynote

ERIK RIEDEL BEST PAPER AWARD

presented to

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu

for

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

Amy Rich, President — 2/25/2025, Date

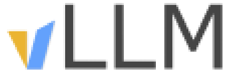
USENIX FAST2025 Best Paper

kvcache-ai/Mooncake -- An open-source initiative co-launched by Moonshot AI and Tsinghua University, with collaboration from various large model and infrastructure providers

Moonshot AI

and more …

Mooncake – Adopted/Collaborated with Other Famous Communities



- One of the most widely used inference engines, adopted by major cloud providers
- Its distributed inference is built on Mooncake

41.5K Stars

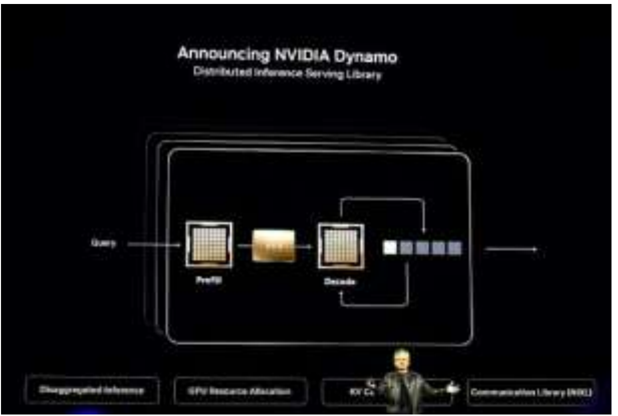
Official release!

Cluster Scale Serving

- Support XpYd disaggregated prefill with MooncakeStore (#12957)

NVIDIA Dynamo

- Spotlighted by Jensen Huang at GTC 2025 Keynote
- Its architecture is inspired by Mooncake, with explicit acknowledgments



Acknowledgement

We would like to acknowledge several open source software stacks for motivating us to create Dynamo.

- vLLM and vLLM-project
- SGLang
- DistServe
- Mooncake

- *Memory bottlenecks*: Large-scale inference workloads demand extensive capacity. KV cache offloading across memory hierarchies (HBM, DDR, N memory limits and speeds up latency. (Mooncake, AIBrix, LMCache)

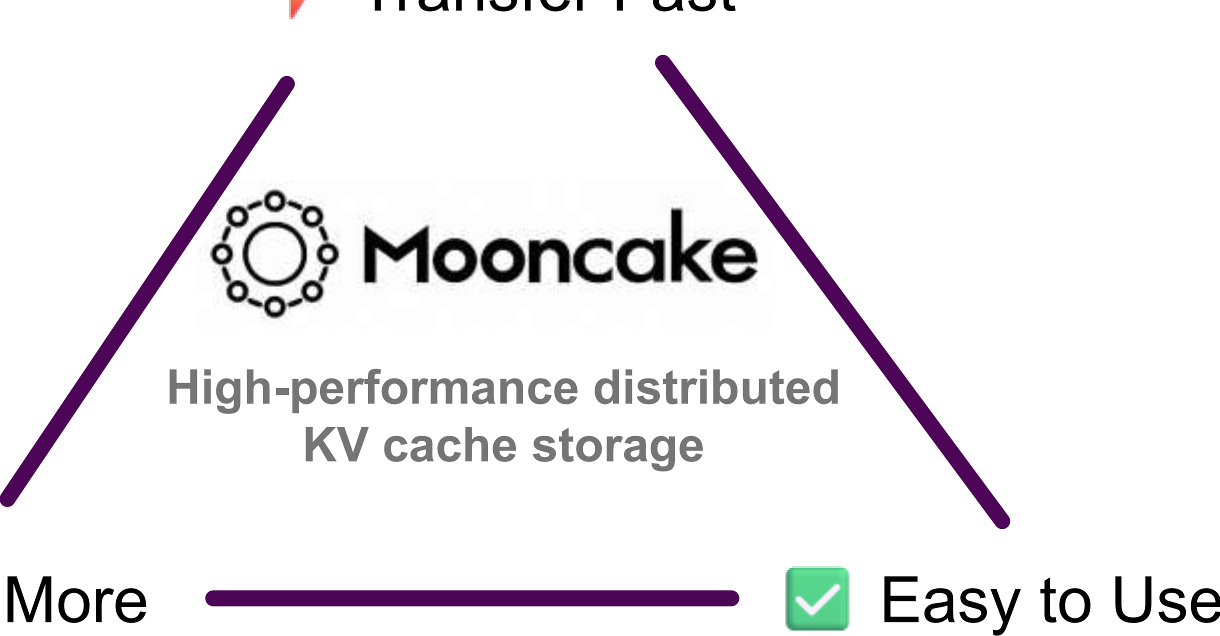
- Inference engine of xAI, widely used in DeepSeek inference
- Distributed architecture was co-developed with Mooncake

The SGLang Team is honored to announce that the following well-known companies and teams, among others, have adopted SGLang for running DeepSeek V3 and R1. @AMD @nvidia @Azure @basetenco @novita_ai_labs @BytedanceTalk @DataCrunch_io @hyperbolic_labs @Vultr @runPod

SGLang has achieved a milestone with full support for PD Disaggregation, thanks to the MoonCake team. Teng Ma, Shangming Cai, Xuchun Shang and Yuan Luo were instrumental in this achievement. Special thanks to Atlas Cloud for their support with the H100s cluster. Let's go! 🚀

Key to KVCache

- **Mooncake Transfer Engine**
- **End-to-end zero-copy**



Store More

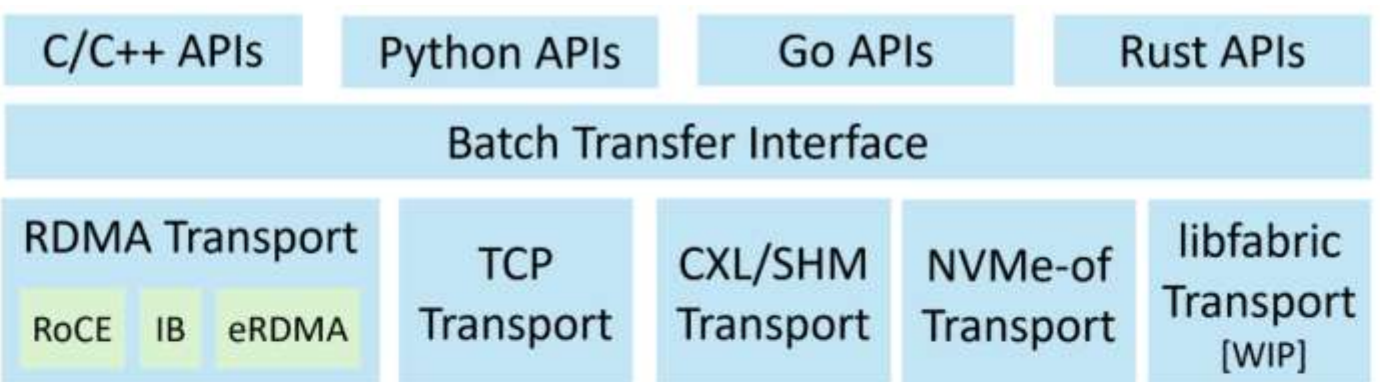
Easy to Use

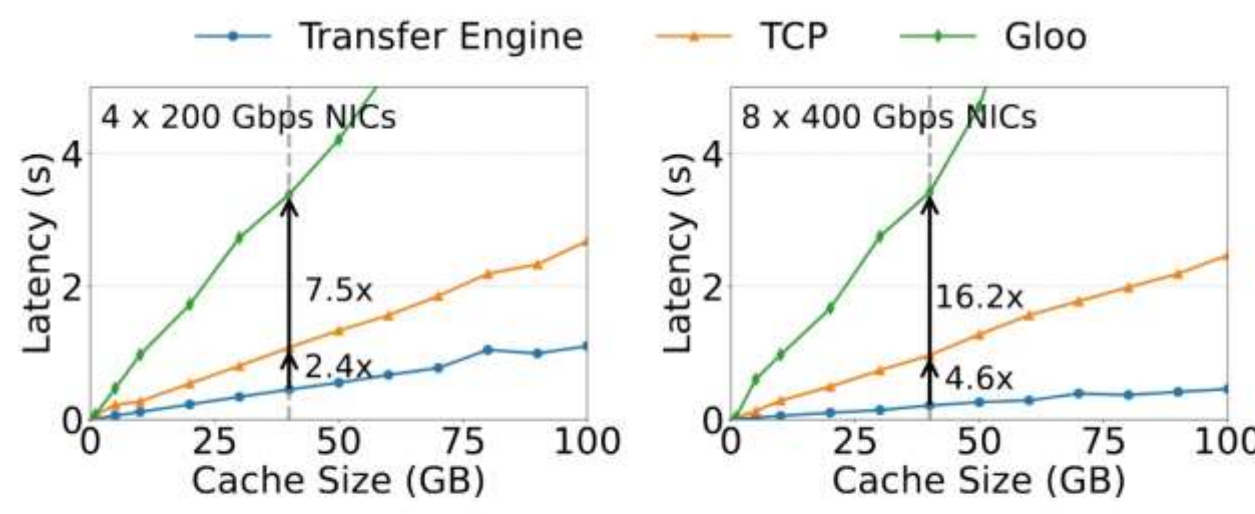
- **Elastic, Shared, and Multi-layer KV Cache**
- **Memory Allocator Optimized for LLM Inference**

- **Extensive and user-friendly APIs**

Transfer Fast: Mooncake Transfer Engine

- **Key features**
  - Topology-aware path selection
  - Multi-NIC pooling
  - Supports multiple protocols and provides unified interfaces.
  - Multi-language APIs



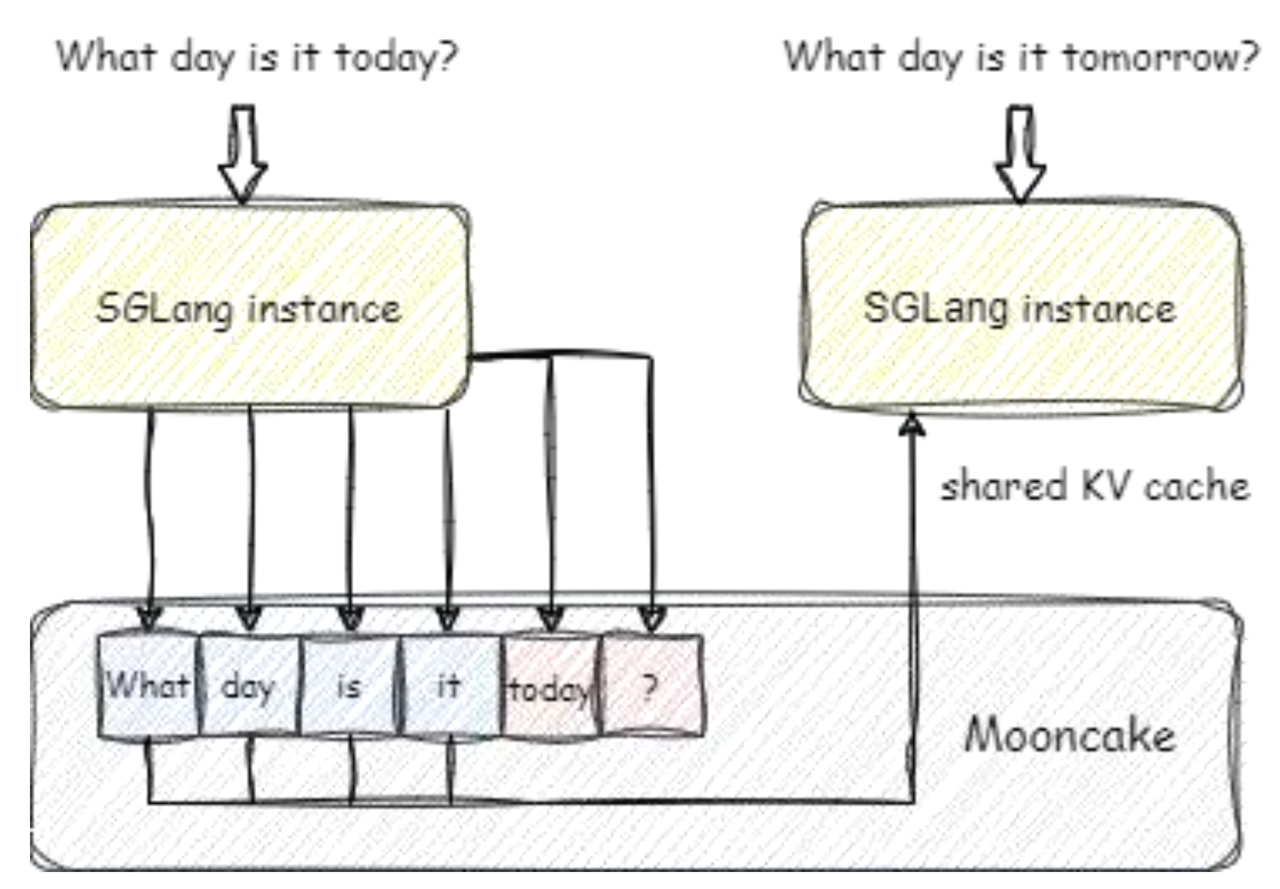


Lightening fast over RDMA

- 40 GB KVCache (128k tokens, LLaMA3-70B)
- **87 GB/s @ 4×200 Gbps**, RoCE
- **190 GB/s @ 8×400 Gbps**, RoCE

Store More: Elastic Shared Multi-layer KV Cache

- **Key features**
  - **Distributed KV cache sharing:** storing one and usable by all
  - **Dynamic resource scaling:** dynamically adding and removing store nodes (startup in <80s for 500GB memory and 8 RDMA NICs)
  - **Multi-layer storage (WIP):** offloading cached data from RAM to SSD



Extensive APIs, Easy to Use

Put/Get APIs

- Put/Get single object
- Batch Put/Get
- **(Batch) Zero-copy Put/Get: recommended**
- (Batch and zero-copy) Put/Get from/into multi-parts

Configurable KV cache placement

- Replica number
- With soft pin
- Preferred segment

Hello world example

```
from mooncake.store import MooncakeDistributedStore

# 1. Create store instance
store = MooncakeDistributedStore()

# 2. Setup with all required parameters
store.setup(
    "localhost",            # Your node's address
    "http://localhost:8080/metadata",    # HTTP metadata se
    512*1024*1024,          # 512MB segment size
    128*1024*1024,          # 128MB local buffer
    "tcp",                              # Use TCP (RDMA for
    "",                             # Leave empty; Mooncake
    "localhost:50051"        # Master service
)

# 3. Store data
store.put("hello_key", b"Hello, Mooncake Store!")

# 4. Retrieve data
data = store.get("hello_key")
print(data.decode())  # Output: Hello, Mooncake Store!

# 5. Clean up
store.close()
```

Evaluation: Effective Request Capacity

- Effective request capacity: Number of requests that meet the latency requirements
- Achieve up to a **498%** increase in effective request capacity compared to vLLM, vLLM with prefix caching and with chunked prefill

Mooncake vLLM vLLM Prefix Caching vLLM Chunked Prefill

(a) Conversation
Better
+498% +157% +59%
Threshold I Threshold II Threshold III

(b) Tool&Agent
Better
+64% +42% +22%
Threshold I Threshold II Threshold III

(c) Synthetic
Better
+62% +40% +28%
Threshold I Threshold II Threshold III

Request Capacity Ratio (%)
Time between Tokens (ms)

Heterogeneous GPU Interconnects

- Multiple paths coexist within the same cluster

Node 1
CPU
PCIe Switches
GPU
NIC

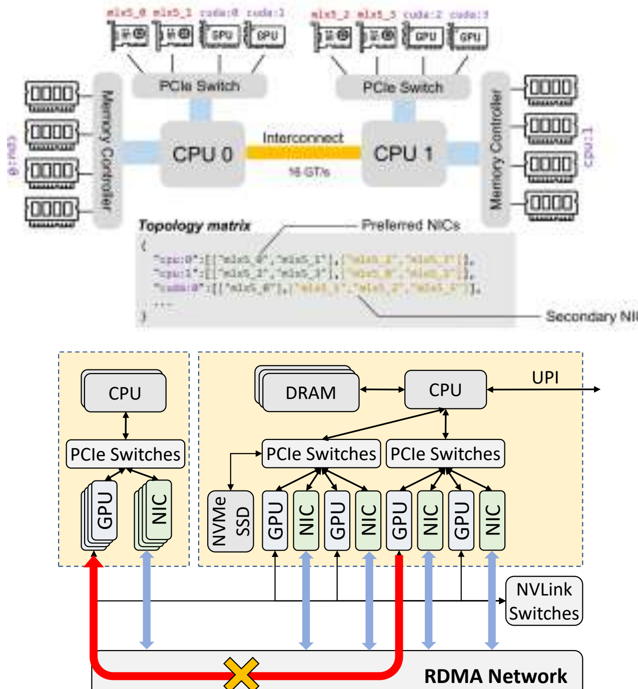
Node 2
DRAM
CPU
UPI
CPU
DRAM
PCIe Switches
PCIe Switches
PCIe Switches
PCIe Switches
NVMe SSD
GPU
NIC
GPU
NIC
GPU
NIC
GPU
NIC
NIC
GPU
NIC
GPU
NIC
GPU
NIC
GPU

NVLink
Switches
GPU-Direct Storage
NVMe-of Attached SSD
GPU-Direct RDMA
RDMA Network

Hidden Risks of Mooncake TE

```
auto engine = new TransferEngine();
engine.installTransport("rdma", args);

auto id = engine.allocateBatch();
engine.submitTransfer(id, reqs);
while (true) engine.getStatus(id, st);
engine.freeBatch(id);
```



- The Imperative Path Selection Paradigm

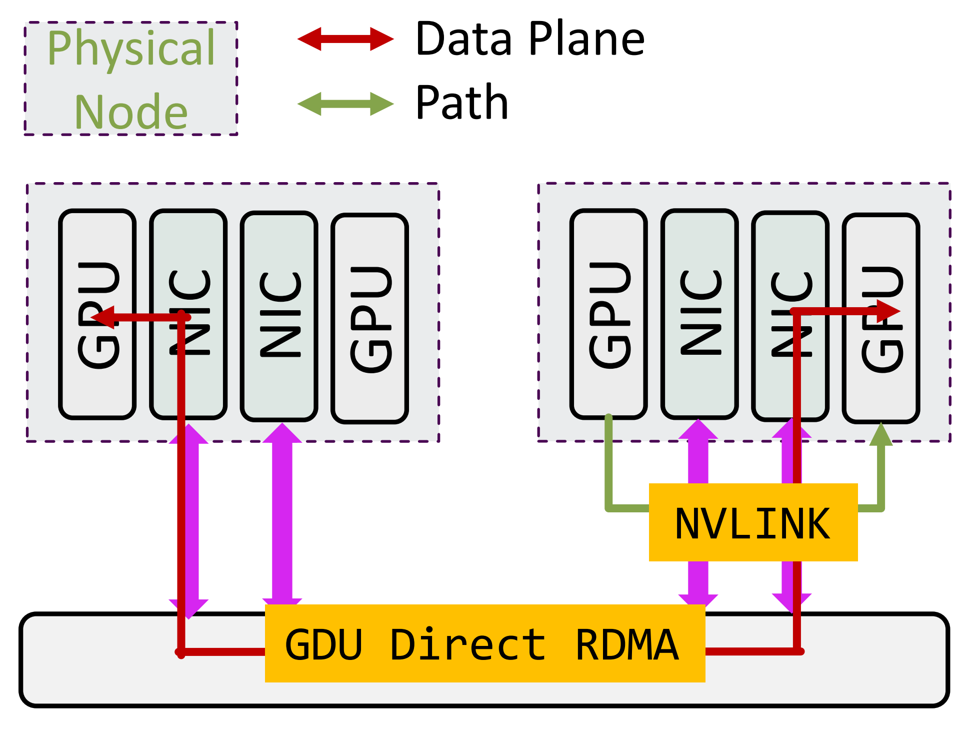
◎ made static binding decisions once **at startup**

◎ executed a **fixed, state-blind** path scheduling policy

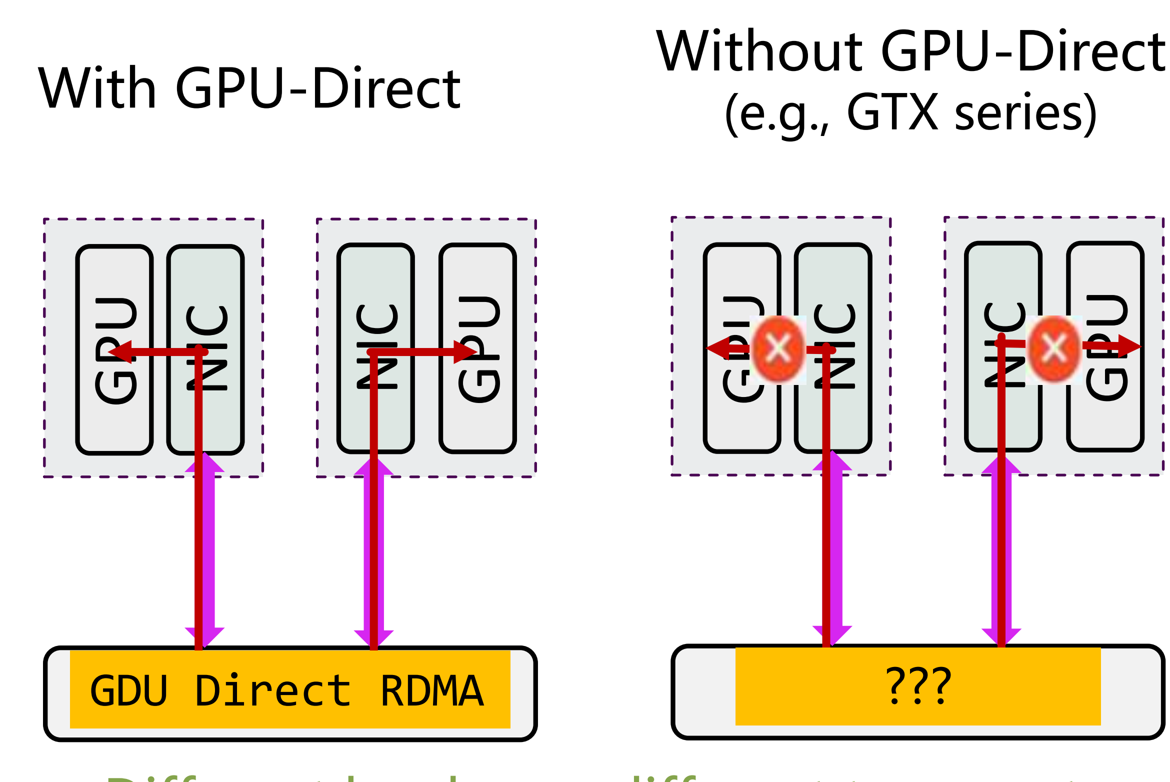
◎ **executed fragilely**, and lacks mechanisms to detect & bypass unavailable paths

Challenges from Imperative Path Selection

- Static Binding
  - Creates communication silos



Different workloads, different transports



Different hardware, different transports

Challenges from Imperative Path Selection

- State-Blind Scheduling
  - Increases latency and wastes bandwidth

Slices from this request
Slices from concurrent requests

A transfer request with 2560 KB
40 slices in total (each 64KB)

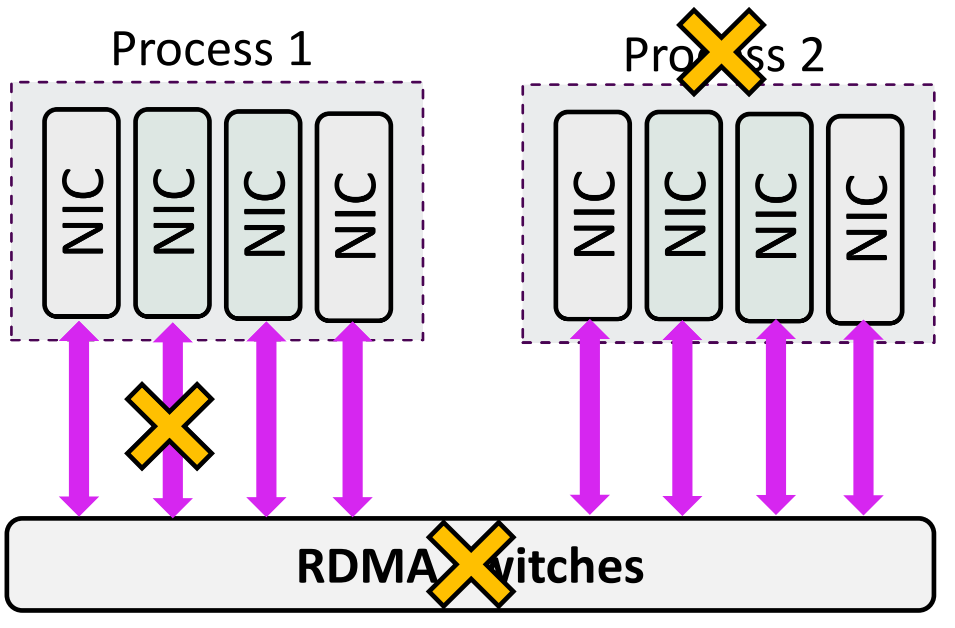
Topology matrix snippets:
"cpu:0": {[NIC0, NIC1, NIC2, NIC3], [...]}

NIC0
NIC1
NIC2
NIC3

Transfer Latency

Challenges from Imperative Path Selection

- Fragile Execution
  - Requires manual intervention and heavy troubleshooting



- What if a single RDMA fabric failed?
- What if a single process crashed?
- What if the RDMA switch failed?

TENT: Transfer Engine NT (New Technology)

- Goal: Make all transports first-class citizens

vLLM | SGLang | Dynamo | LMCache | Checkpoint Engine | NIXL | Mooncake Store | ...

Transfer Engine APIs

Orchestration
Discover
Routing
Staging
Plugin Loaders
Error Handling

Unified Segment Abstraction
Buffers
Devices
Topology

Transport Plugins (load all if possible)
RDMA (with GDR/eRDMA)
NVLink
MNNVL
TCP
RoCM
SHM/CXL
GDS
io_uring
HIXL
...

Control Plane Plugins
P2P
etcd
Redis
HTTP

Heterogeneous GPU Interconnects

Features of Mooncake TENT

Dynamic Orchestration

- ◎ Unified Segment Abstraction
- ◎ Application-Oblivious Topology Discovery
- ◎ Dynamic Per-Request Orchestration

Adaptive Slice Spraying

- ▪ Latency Prediction based NIC Selection
- ◎ Cross-Process Fairness

Resilient Self-Healing

- ◎ Link-Level Resilience
- ◎ Transport-Level Resilience

Unified Segment Abstraction

(applicable to all transports)

0 UINT64_MAX

Logical View

cpu:0 *cpu:1* *cuda:0*

Metadata View

name: *node050:12345*
type: *memory*
topology
buffers
devices

This segment can be referred as the engine's host and port

Topology

location: *cpu:0*
tier 1 devices: *[mlx5_0, …]*
tier 2 devices: *[]*
tier 3 devices: *[mlx5_4, …]*

Devices

name: *mlx5_0*	
type: *rdma*	
transport specific data	rdma/{lid,gid,props}
	…

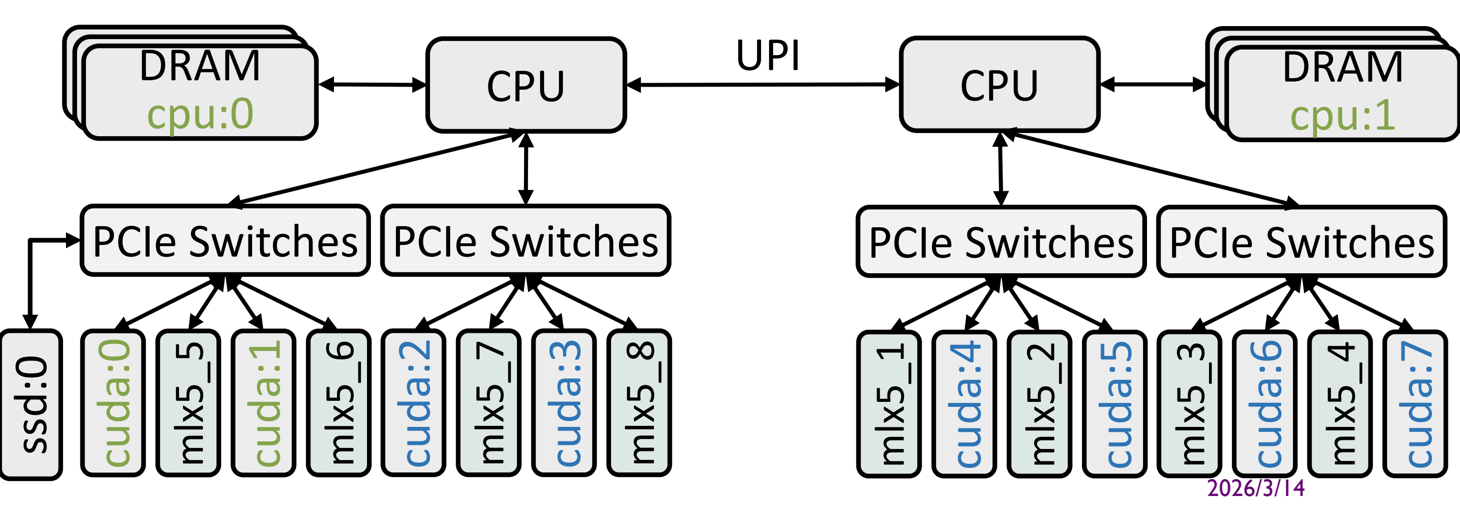
Buffers

address	
length	
location: *cpu:0*	
transports: *[nvlink, rdma, tcp]*	
transport specific data	nvlink/handle
	rdma/rkeys
	…

Application-Oblivious Topology Discovery

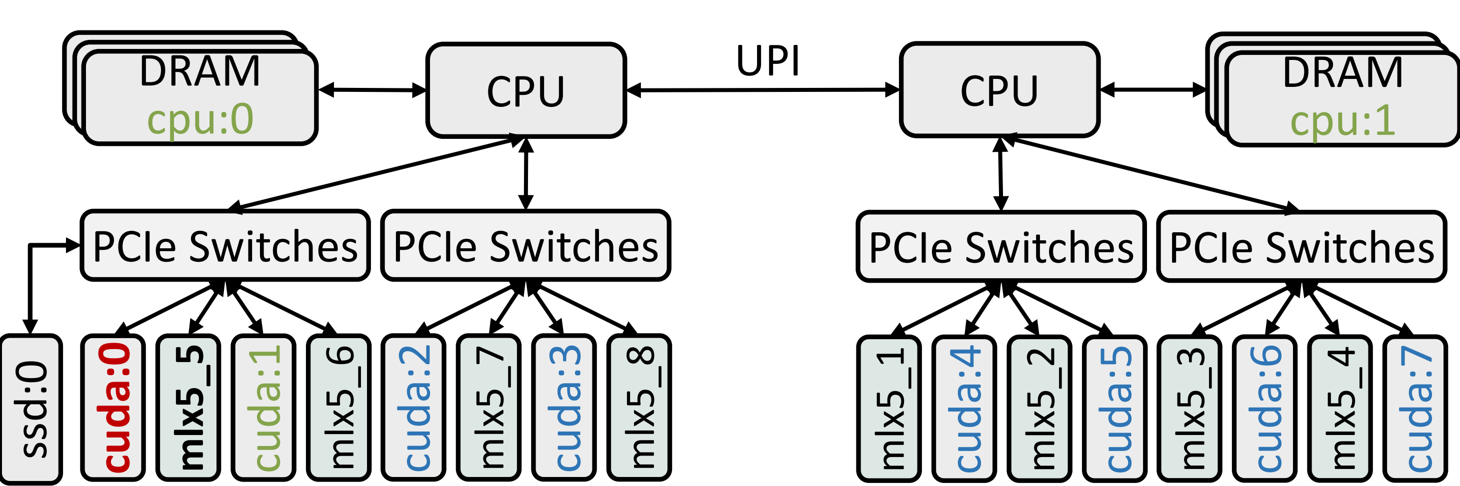
- Step 1: Probe hardware information
  - List of memory/NIC devices
  - Their NUMA affinity, PCIe Bus ID
  - Capabilities: bandwidth, direct-access, etc.

```
MEM:
cpu:[0-1],
cuda:[0-7]
NIC:
mlx5_[1-8]
```



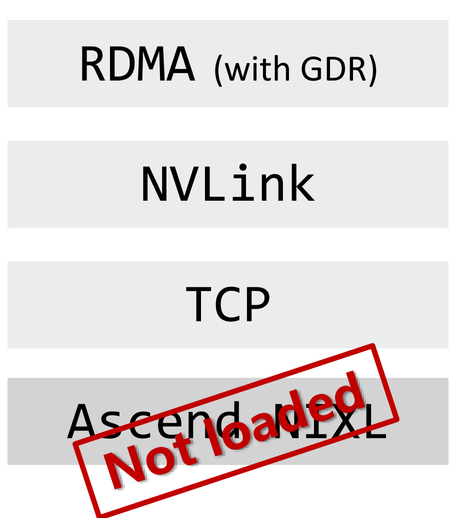
Application-Oblivious Topology Discovery

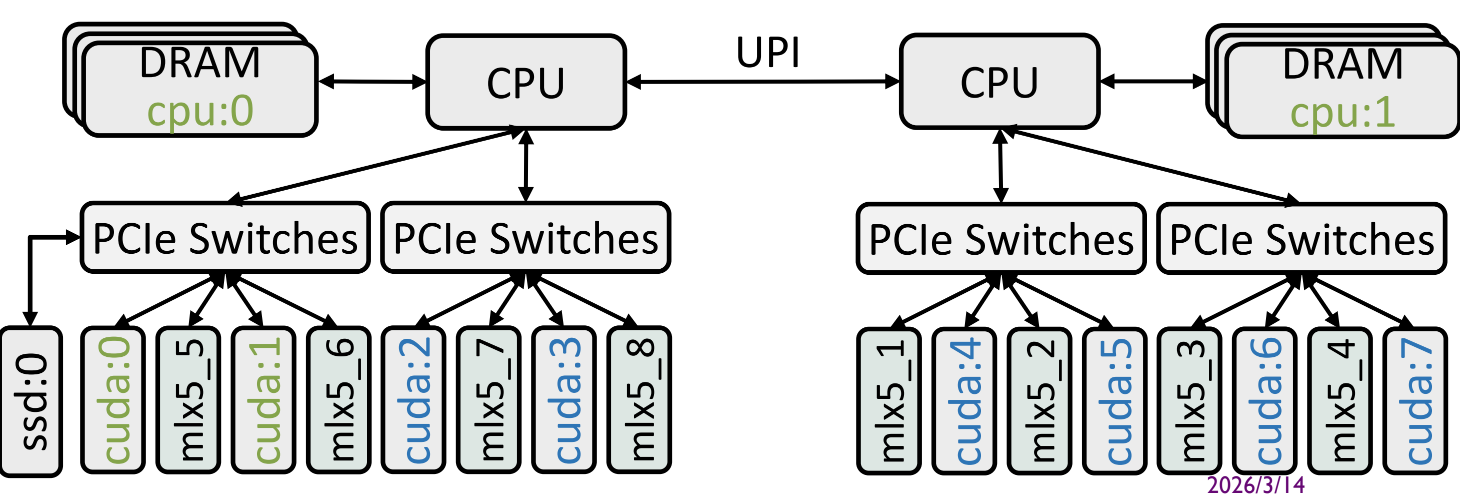
- Step 1: Probe hardware information
- Step 2: Maps NICs for each MEM
  - Tier 1: NIC(s) with the shortest PCIe hop
  - Tier 2: same NUMA but not in tier 1
  - Tier 3: cross NUMA



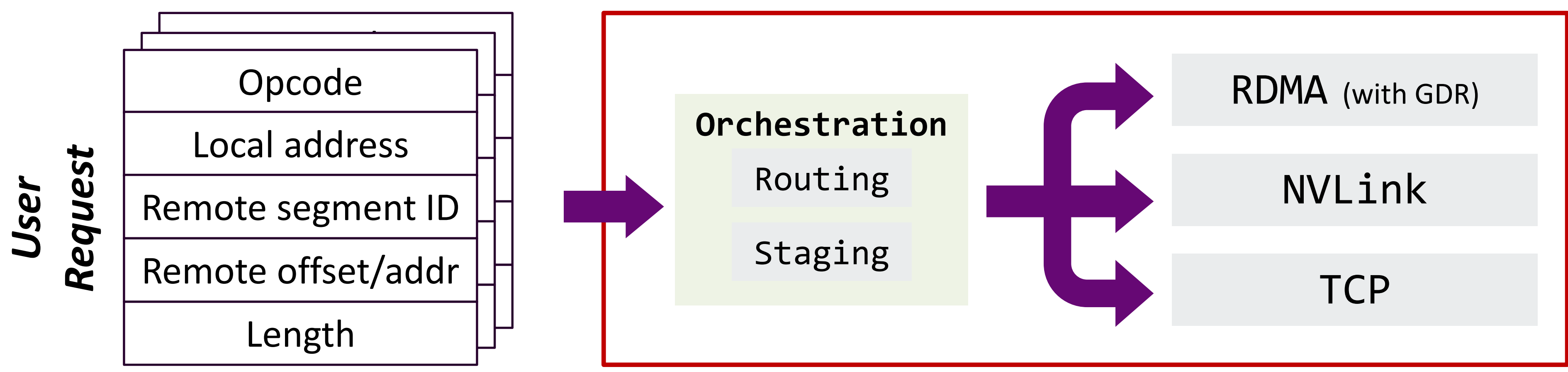
Application-Oblivious Topology Discovery

- Step 1: Probe hardware information
- Step 2: Maps NICs for each MEM
- Step 3: Load transports
  - Runtime support and transports can be dynamic libraries
  - They can be loaded on runtime (e.g., if CUDA is enabled)





Dynamic Per-Request Orchestration



- Decide transports for each request using the Unified Segment
  - Local address

    find local MEM type

    find local installed transports

◎ Remote segment ID & offset/address

find remote MEM type

find remote installed transports

Prefer to use a transport with the highest speed, and supported by both sides

Features of Mooncake TENT

Dynamic Orchestration

- ◎ Unified Segment Abstraction
- ◎ Application-Oblivious Topology Discovery
- ◎ Dynamic Per-Request Orchestration

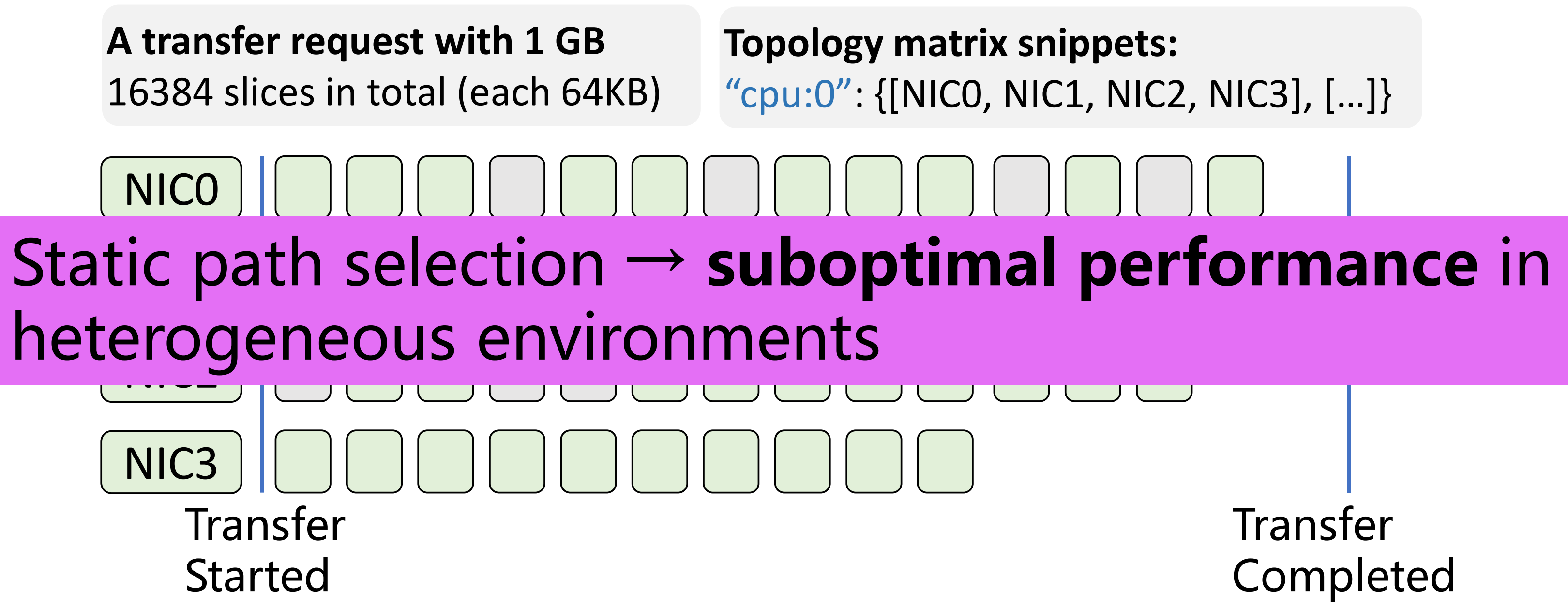
Adaptive Slice Spraying

- ▪ Latency Prediction based NIC Selection
- ◎ Cross-Process Fairness

Resilient Self-Healing

- ◎ Link-Level Resilience
- ◎ Transport-Level Resilience

Telemetry-Driven Adaptive Scheduling



- How to reduce transfer latency and avoid bandwidth waste
  - **Predict**: select local-remote NIC pair based on historical telemetry
  - **Feedback**: use measured latency to update prediction parameters

Local NIC Selection

- Latency estimation
- $L_{pred}(NIC_k) = \beta_{1,k} \frac{IF_k + P_k * S}{BW_k} + \beta_{0,k}$
  - IF_k : NIC inflight bytes
  - S : Slice length
  - P_k : Penalty factor (e.g., higher for cross-NUMA)
  - BW_k : NIC bandwidth
  - $\beta_{0,k}, \beta_{1,k}$: Prediction parameters

Pre-transfer

◎ Calculate L_{pred}

◎ Find the best NIC

◎ Reserve inflight bytes IF_k

Post-transfer

◎ Return inflight bytes IF_k

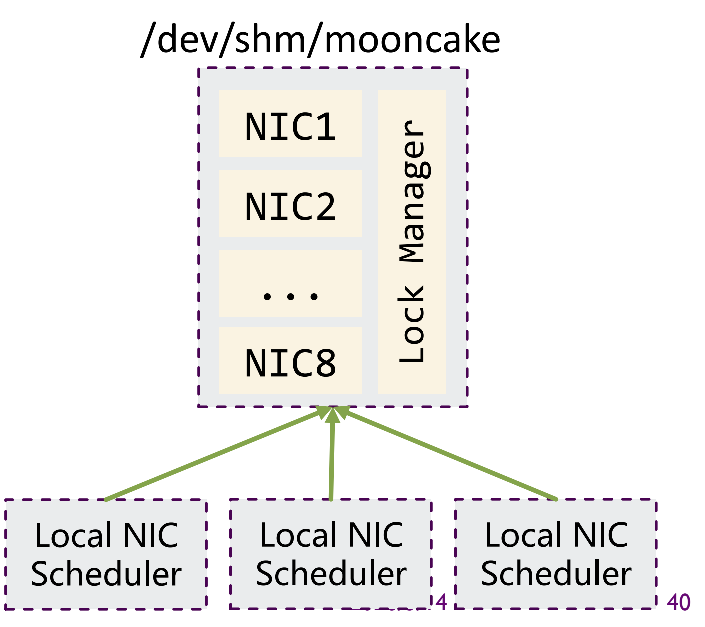
◎ Measure latency

◎ Update $\beta_{0,k}, \beta_{1,k}$ using EWMA

(make estimation more accurate)

Cross-Process Fairness

- How to avoid any single process from saturating NICs?
  - Coarse-grained quota allocation
- Decentralized, shared-memory implementation
  - Not every scheduling task enters the global level
    Interval: ~tens of milliseconds
  - Tolerant shutdown/crashes of any process



Features of Mooncake TENT

Dynamic Orchestration

- ◎ Unified Segment Abstraction
- ◎ Application-Oblivious Topology Discovery
- ◎ Dynamic Per-Request Orchestration

Adaptive Slice Spraying

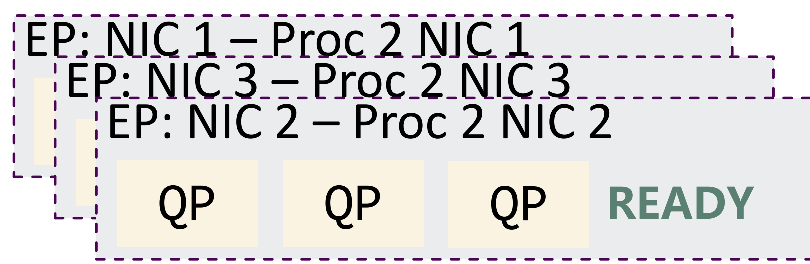
- Latency Prediction based NIC Selection
- ◎ Cross-Process Fairness

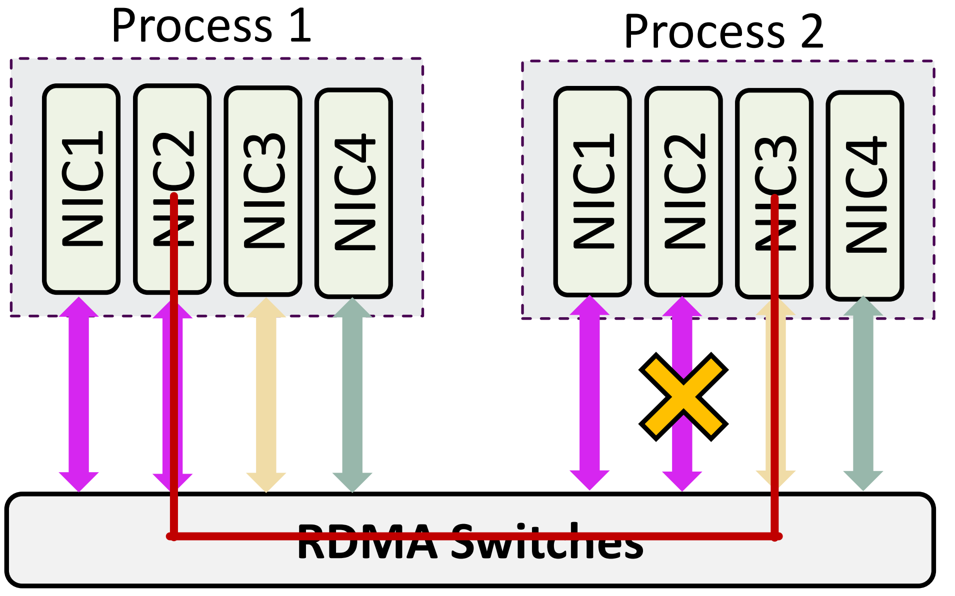
Resilient Self-Healing

- ◎ Link-Level Resilience
- ◎ Transport-Level Resilience

Proactive Dual-Layer Resilience

- RDMA Resource Lifecycle
  - Endpoint == NIC-to-NIC connection





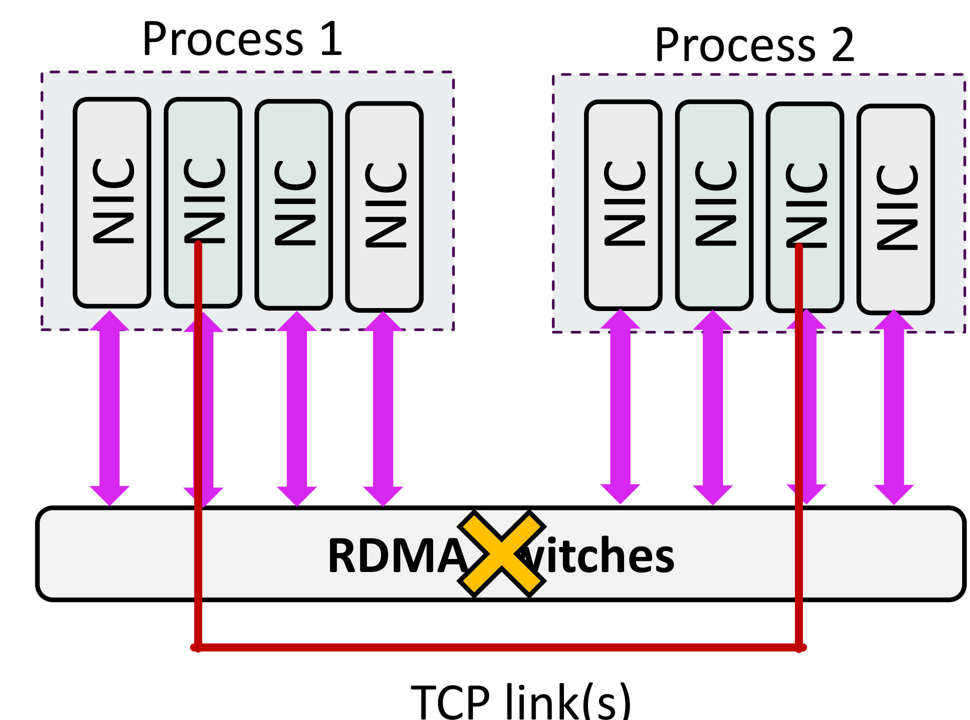
Link-Level Resilience

◎ Detect failed/unstable link: PAUSE, CLOSE or TERMINATE

◎ Allow suboptimal path

Proactive Dual-Layer Resilience

- Transport-Level Resilience
  - Transparent fallback to other transports
    (e.g., RDMA→TCP)
  - Driven by Dynamic Per-Request Orchestration



- Recovery
  - Transport/path will be reused after link recovery

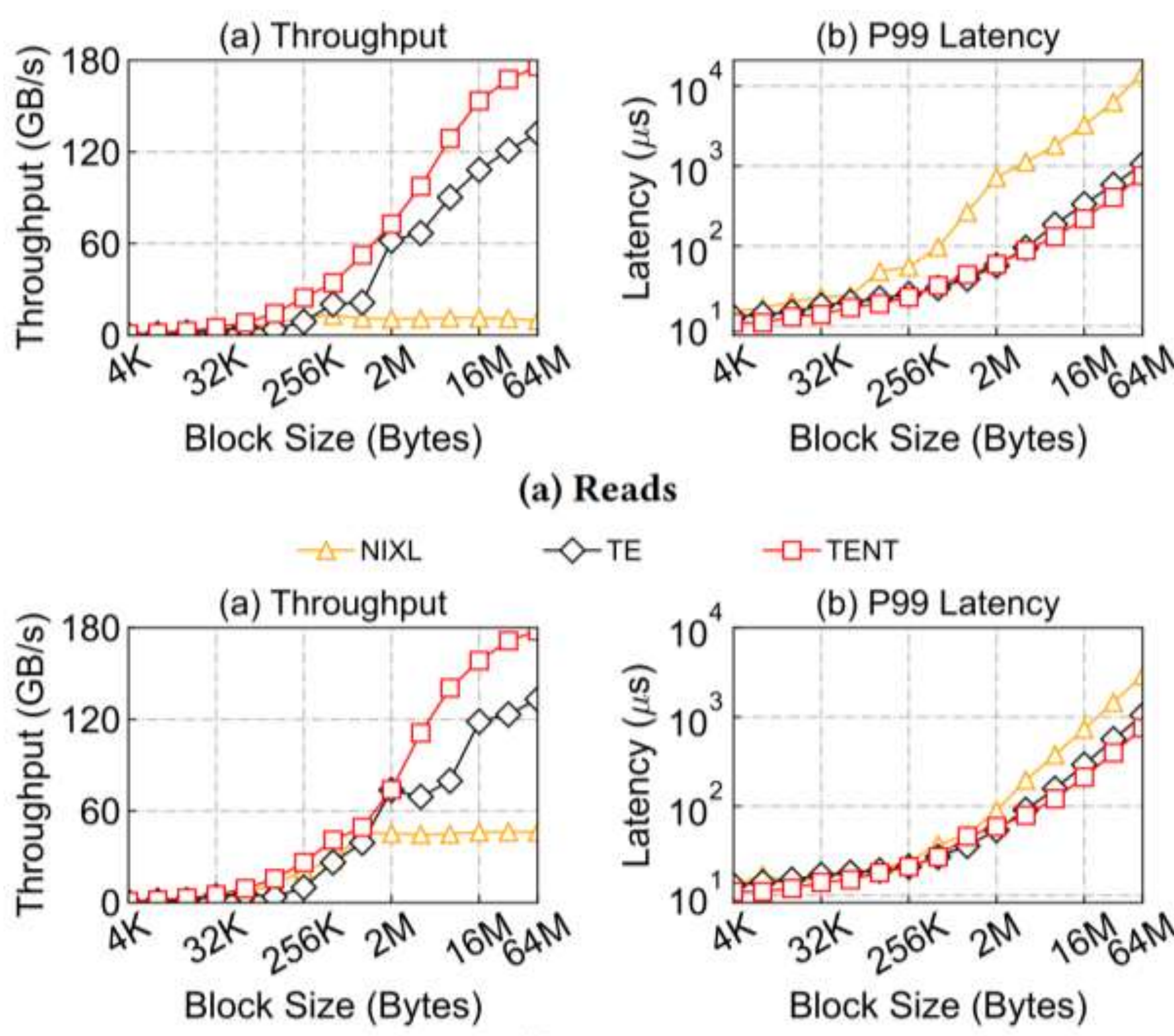
Evaluation

- Test Cluster: NVIDIA H800 Platform
  - Each node is equipped with two Intel Xeon Platinum 8468V CPUs and eight NVIDIA H800 GPUs
  - NVLink & 200 Gbps $\times$ 8 RoCE interconnect

- **Competitors**
  - Mooncake TENT
  - Mooncake TE (mainstream version)
  - NVIDIA NIXL (UCX backend)

Evaluation

- Synthetic Workload
  - Block size range from 4KB to 64MB
  - Two concurrent threads, 8 NICs are fully utilized

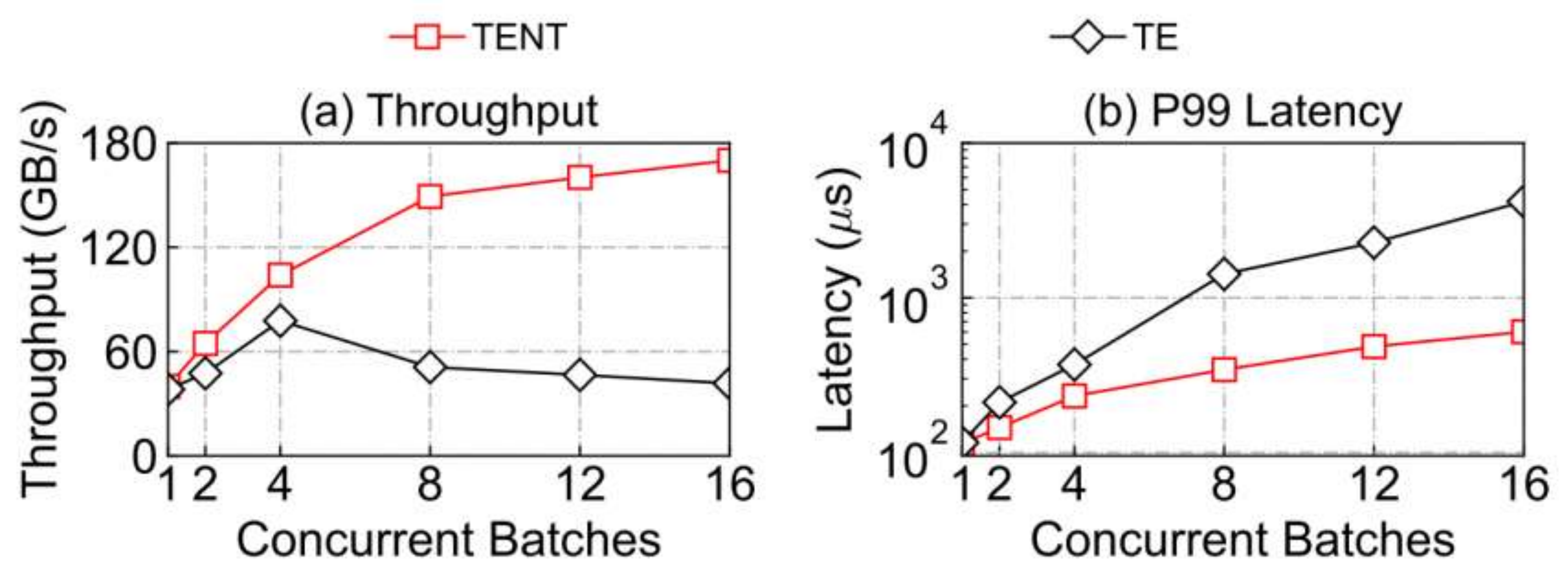


(a) Reads

(b) Writes

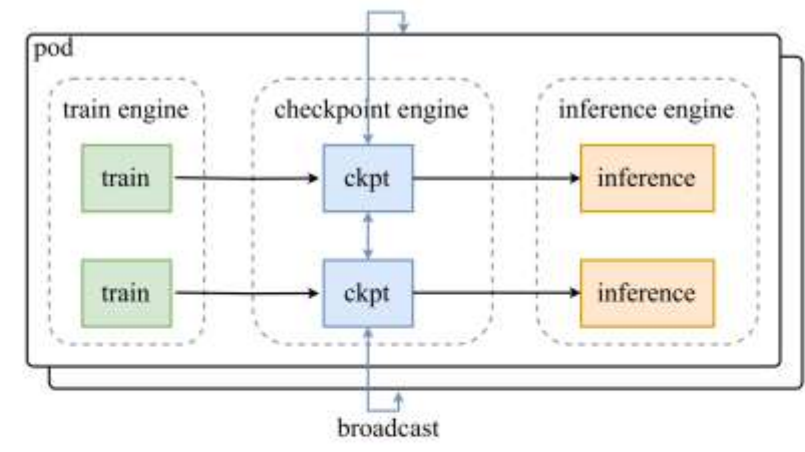
Evaluation

- KVCache Transfer Benchmark
  - **Workload:** DeepSeek-R1-W8A8 model with a 4K input
  - Comprises 61 layers, each containing 32 blocks of 144 KB, consisting of a 128 KB NoPE block and a 16 KB RoPE block



Evaluation

- Case Study: Moonshot AI Checkpoint Engine
  - Open source with Kimi K2
  - Update model weights in LLM inference engines
  - Update time ∝ transfer latency



Model	TE	TENT
Qwen3-235B-A22B-Instruct-2507	28.56	18.97
GLM-4.5-Air	14.75	11.26

Thanks!

kvcache.ai

KVCache.AI is a joint research project between MADSys and top industry collaborators, focusing on efficient LLM serving.

903 followers · https://madsys.cs.tsinghua.edu.cn/ · zhang_mingxing@mail.tsinghua.edu.cn

Pinned — Customize pins

Mooncake (Public)
Mooncake is the serving platform for Kimi, a leading LLM service provided by Moonshot AI.
C++ · 4.1k · 388

ktransformers (Public)
A Flexible Framework for Experiencing Cutting-edge LLM Inference Optimizations
Python · 15.2k · 1.1k

TrEnv-X (Public)
Go · 58

https://github.com/kvcache-ai