TENT: A Declarative Slice Spraying Engine for Performant and Resilient Data Movement in Disaggregated LLM Serving

Feng Ren[†], Ruoyu Qin[†♡], Teng Ma[‡], Shangming Cai[‡], Zheng Liu[‡], Chao Lei[♣], Dejiang Zhu[♣], Ke Yang[◇], Jinyang Su[◇], Weixiao Huang[♡], Yikai Zhao[♡], Yongwei Wu[†], Weimin Zheng[†], Mingxing Zhang[†]
[†]Tsinghua University; [♡]Moonshot AI; [‡]Alibaba Group; [♣]Ant Group;[◇]Approaching AI

Abstract

The disaggregation of LLM serving has transformed GPU clusters into a complex mix of heterogeneous interconnects, such as PCIe, multi-rail RDMA, and NVLink. However, existing data transfer frameworks lack the ability to intelligently coordinate these diverse links, resulting in performance bottlenecks, reduced resilience, and increased operational overhead. This paper introduces Mooncake Transfer Engine NT (TENT), a novel data movement engine that shifts from the traditional *Imperative Path Selection* to a *Declarative Slice Spraying* approach. Instead of statically binding to a single transport, TENT automatically discovers all available interconnects and topologies, unifying them into a dynamic resource pool. At runtime, TENT's adaptive scheduler intelligently "*sprays*" data slices across the healthiest and best-performing links based on real-time telemetry. Furthermore, its multi-layered resilience mechanism provides seamless failover within and across transports, enabling self-healing from link failures. Our evaluation shows that TENT simplifies operations in heterogeneous environments, increases elephant flow transfer throughput by over 33.7% compared to its predecessor. TENT is also effective in updating model weights for LLM reinforcement learning, reducing the update time by up to 50.5%.

1 Introduction

The demands of disaggregated LLM serving have shattered the traditional view of GPU networking. A modern GPU cluster is no longer a single, uniform fabric but a complex amalgamation of heterogeneous interconnects—PCIe, multi-rail RDMA, NVLink [12], and emerging fabrics such as CXL, Multi-node NVLink (MNNVL) [12], and Ascend UB [7]—each with distinct performance characteristics and control planes. This transforms data path management into a multi-dimensional problem involving protocol selection, traffic striping across multiple rails, and even the orchestration of compound, relayed paths. The critical challenge in this environment is not the lack of peak bandwidth, but rather the absence of a framework that can intelligently coordinate these diverse links into a single, resilient, and programmable system.

This challenge became starkly evident in the evolution of **Mooncake** [15], our production disaggregated LLM serving system. Its original **Mooncake Transfer Engine (TE)**, was a significant step forward, offering a unified, intent-based `BatchTransfer` API over multiple transports to move a batch of discontiguous memory buffers to another batch of destination addresses. This design, with its priority-based path selection, was effective in simpler, predictable environments. However, its core philosophy was still an imperative model that made binding decisions once at startup and executed a fixed scheduling policy, effectively assuming a stable and homogeneous world. This static, "pick-a-path" strategy proved brittle and inefficient as our production clusters grew in complexity and dynamism.

While recent academic research has addressed pieces of this puzzle, the overarching coordination gap remains. Systems like FuseLink [17] focus on GPU-relay and NIC aggregation rather than a cross-transport framework, and UCCL [21] remains centered on optimizing RDMA over unreliable datagrams, which are not universally supported across platforms (e.g., eRDMA). These solutions improve isolated axes but fail to provide holistic, end-to-end orchestration.

We argue that overcoming these limitations requires a paradigm shift from *Imperative Path Selection* to a novel *Declarative Slice Spraying* approach. In this model, the application simply declares its data movement intent, and the engine takes full responsibility for fulfilling it. It transforms all available interconnects into a dynamic, unified resource pool and intelligently *"sprays"* data slices across the most reliable and best-performing links in real time. Unlike network-level packet spraying, which is blind to application intent and operates on homogeneous paths, our approach operates at the application layer to orchestrate data flow across a heterogeneous set of transports. The unique contribution of this work lies in leveraging a holistic, real-time view of all available links to enable intelligent scheduling decisions that are beyond the capability of any single network-level mechanism.

This paper introduces **Mooncake Transfer Engine NT (TENT)**, a ground-up redesign that embodies the declarative Slice Spraying philosophy. Building on the limitations of TE's imperative path selection model, TENT introduces three key innovations:

- **From *Static Binding* to *Dynamic Orchestration.*** TE is restricted to a single transport backend loaded at startup, creating communication silos where nodes using different transports cannot interoperate. TENT eliminates this rigidity by automatically discovering full topologies – including PCIe/UPI hierarchies, NVLink connectivity, and GPU Direct RDMA reachability – and then integrating them into a unified segment abstraction. On top of this, TENT performs dynamic per-request orchestration, transparently selecting the optimal path for each request, thereby enabling seamless cross-transport communication.
- **From *Fixed Scheduling* to *Adaptive Slice Spraying.*** By simply slicing requests into fixed-size chunks and randomly distributing them across NICs, TE fails to accommodate elephant flows, leaving bandwidth underutilized and increasing end-to-end latency. TENT instead employs telemetry-driven adaptive scheduling. Each slice is dynamically scored against

Corresponding author: Mingxing Zhang (zhang_mingxing@mail.tsinghua.edu.cn).

candidate links using real-time bandwidth, latency, and topology affinity, then sprayed across parallel rails. This adaptive scheme maximizes throughput while ensuring fairness across concurrent processes.

- **From *Brittle Execution* to *Resilient Self-Healing*.** TE's imperative design is fragile, as it lacks automated mechanisms to fall back across different transports. Consequently, a single faulty link can stall the pipeline and force operators to rely on manual resets. TENT introduces dual-layer resilience. At the link level, it continuously monitors health to enable instant failover and seamless reintegration. At the transport level, it retries requests across alternative backends without application involvement. This proactive, self-healing design ensures robust high availability for multi-rail, heterogeneous clusters.

Our experiments show that TENT achieves up to 33.7% improvement in throughput and 30.5% reduction in P99 tail latency compared to its predecessor, TE. Under KVCache transfer workloads, TENT attains up to 4.07× higher throughput. These policies can be also integrated into other systems that support multi-NIC communication. For example, when updating model weights for LLM reinforcement learning, TENT reduces the update time by up to 50.5%.

2 Background and Motivation

2.1 Heterogeneous Interconnects in AI Cluster

The rapid growth of large-scale LLM serving has driven GPU clusters to adopt increasingly heterogeneous communication substrates. Unlike conventional data centers that typically rely on a single network fabric (e.g., InfiniBand or Ethernet), modern AI clusters integrate multiple types of interconnects to meet the demands of higher bandwidth, lower latency, and stronger robustness.

Figure 1 shows the interconnects of an 8× H800 server. Within a node, CPUs, GPUs, NICs, and host memory are connected by UPI/PCIe, whose tree-like topology introduces non-uniform communication costs. To accelerate GPU-intensive workloads such as KVCache transfer, direct GPU-to-GPU links (e.g., NVLink and Ascend UB) provide much higher bandwidth and lower latency. Across nodes, multi-rail RDMA networks dominate large-scale training and inference, while emerging fabrics such as CXL and multi-node NVLink extend high-speed connectivity to rack scale.

In practice, these heterogeneous fabrics must be jointly managed by a communication stack that allocates resources and orchestrates data transfers across links. However, current solutions treat each interconnect in isolation, making it difficult to fully exploit their combined potential. This gap underscores the need for a unified framework that can efficiently and reliably harness diverse interconnects in modern AI clusters.

2.2 Limitations of Imperative Path Selection

Existing transfer engines such as TE [15], NIXL [11], and DLSlime [2] nominally expose a unified interface across multiple backends. In practice, however, they still follow an *imperative path selection* paradigm: users must explicitly bind requests to *one* chosen backend during initialization, regardless of the availability of other fabrics. This design is fundamentally misaligned with heterogeneous interconnects in AI cluster, where PCIe, NVLink, RDMA, CXL, and vendor-specific links coexist.

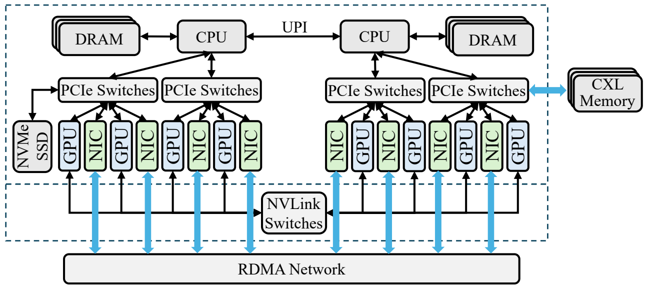


Figure 1: Interconnects of an 8× H800 server.

2.2.1 *Static Binding Creates Communication Silos*

With the current design, processes bound to different transport backends cannot interoperate directly, fragmenting the cluster into isolated communication silos. A common workaround is to run multiple transfer engines per node, but this introduces configuration overhead and complicates system maintenance.

The limitation becomes particularly evident in disaggregated LLM serving. The prefill phase is compute-intensive and favors GPU-dense servers, while the decode phase is memory-intensive and often constrained by KVCache capacity. To accommodate these divergent demands, prefill and decode are sometimes deployed on heterogeneous clusters (e.g., Ascend 910B as prefill nodes, and H20 as decode nodes). Static binding, however, rigidly ties communication to a single fabric, possibly preventing efficient end-to-end coordination across phases.

2.2.2 *State-Blind Scheduling Wastes Bandwidth*

TE's imperative scheduler slices large requests into fixed-size chunks (64 KB by default) and randomly maps them to NICs within the same NUMA node. This state-blind strategy works tolerably well under high concurrency with small requests, but it performs poorly on *elephant flows*, which refer to long-lived, high-volume transfers that dominate overall traffic and often lie on the critical path, making them highly sensitive to latency.

KVCache transfers in disaggregated LLM serving are a representative example. They are typically large and bursty, triggered at layer boundaries with only a few concurrent flows active. Considering the DeepSeek-R1-W8A8 model [8] with a 4K input, the total KVCache transfer volume reaches about 281 MB per request. In such cases, forced slicing and random mapping prevent balanced utilization of parallel links, inflating end-to-end latency and leaving bandwidth underused. In addition, during reinforcement learning based LLM training, transferring checkpoints from the training cluster to the inference cluster also constitutes a typical elephant flow, as these transfers are both large in volume and lie directly on the critical path.

2.2.3 *Fragile Execution Requires Manual Intervention*

Alibaba [14] reported that production clusters experience 5K–60K daily link flapping incidents, which introduce frequent transient performance degradation. Yet imperative designs lack robust resilience mechanisms. Similar to NIXL and NCCL, TE provides no automated cross-transport failover, so a single faulty link can stall the entire pipeline and require manual operator intervention. Even when fallback mechanisms exist, they are often brittle: once switched to a degraded path, the system tends to remain stuck-in-fallback and fails to reintegrate the recovered high-performance path. This results in prolonged performance degradation and unnecessary under-utilization of cluster resources.

3 Design

3.1 Overview

TENT inherits TE's compact API for high-performance data movement, including memory registration (*registerLocalMemory*), batch management (*allocateBatch*/*freeBatch*), batched submission (*submitTransfer*), and asynchronous completion monitoring (*getTransferStatus*). These primitives enable unified and efficient bulk transfers across host, GPU memory and SSD storage, serving as the foundation for higher-level services such as distributed KVCache.

We redesign TENT's execution model to eliminate static binding. Rather than assigning an entire batch to a single transport backend, TENT dynamically selects the optimal one for each individual request and spray it among multiple rails in necessary, by leveraging topology awareness, runtime telemetry, and adaptive scheduling. This design preserves full compatibility with TE while transparently extending applications to heterogeneous transports and resilience features, all without code changes.

3.2 Dynamic Orchestration

TENT replaces the rigid, static binding model with a paradigm shift toward Declarative Slice Spraying. At startup, it performs *topology discovery* automatically—including PCIe hierarchies, NVLink links, and GPUDirect RDMA reachability. These heterogeneous resources are then integrated into a *unified segment abstraction*, enabling applications to interact with local memory, remote storage, and network paths through a single, consistent interface. With this foundation, TENT performs *dynamic per-request orchestration*, eliminating the communication silos of imperative transport binding and simplifying both development and execution at scale.

3.2.1 *Application-Oblivious Topology Discovery*

During startup, TE broadcasts a user-specified *topology matrix* to all nodes. TENT first eliminates this manual step by automatically discovering NICs and GPUs through OS and driver queries. Building on this, it further associates each memory location (e.g., `cpu:X` or `cuda:Y`) with its set of connectable NICs. Finally, these NICs are categorized into three tiers according to UPI/PCIe connectivity, providing a structured view that supports fine-grained scheduling.

- *Tier 1*: closest affinity. For GPU memory, NICs under the same (or nearest) PCIe switch as the GPU. For host memory, all NICs within the same NUMA domain.
- *Tier 2*: still within the same NUMA domain but farther. For GPU memory, NICs under the same NUMA domain but different PCIe switches.
- *Tier 3*: cross-NUMA communication. Links between devices located on different NUMA domains.

The topology can be used by any transport backend, enabling dynamic per-request orchestration. For example, under the RDMA transport backend, TENT selects the NIC path with the smallest intra-node overhead by consulting both local and remote topology results.

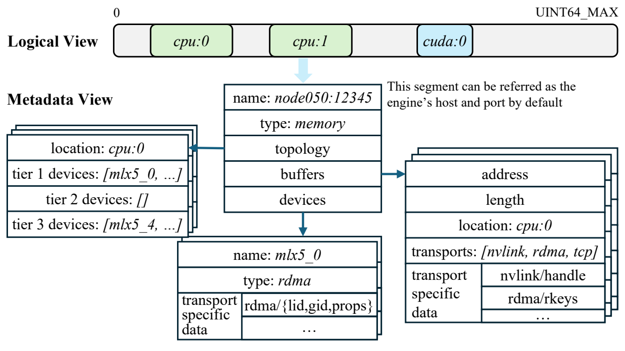


Figure 2: Logical and metadata view of a memory segment.

3.2.2 *Unified Segment Abstraction*

Building on the *segment* abstraction originally introduced in TE, TENT generalizes the concept to unify all transports and rails. Specifically, a *memory segment* denotes a virtual address space that may span multiple non-contiguous buffers across host and GPU memory, yet is presented to the engine as one unified logical region. A *file segment*, in contrast, corresponds to the address space of a file on persistent storage (e.g., an SSD), enabling direct, zero-copy style access without intermediate staging.

Importantly, a segment is transport-agnostic: once defined, it can be carried over any available backend. From the application's perspective, TENT exposes a single, consistent API for all data types—host memory, GPU memory, or file storage. A transfer batch may even mix heterogeneous requests, while the runtime transparently maps each to the best transport. From the system's perspective, a segment encapsulates the topology, registered buffers, and devices. It also contains metadata for specific transport protocols, as shown in Figure 2. This unified organization allows all transports to share the same abstraction, ensuring that new backends can be integrated without application changes.

3.2.3 *Dynamic Per-Request Orchestration*

TENT determines the transport backend at the granularity of each individual transfer request rather than per batch. Each request specifies a local address, a remote segment handle with an offset, and a length. From the segment metadata, TENT can infer both the local and remote memory locations (e.g., host DRAM, GPU memory, or file storage), as well as the list of transport backends commonly supported by both endpoints. Consequently, TENT could find the optimal transport according to a configurable policy. For example, if both endpoints reside on the same node, NVLink is preferred for the transfer.

Based on the new design, TENT can also construct relay paths. This is particularly useful when end-to-end communication cannot be achieved through a single transport. For example, if two nodes are interconnected via MNNVL, which only supports GPU-to-GPU transfers, TENT enables end-to-end host communication by a three-step relay: (1) host memory to GPU in source nodes, (2) GPU to GPU between nodes, (3) GPU to host memory in target nodes.

3.3 Telemetry-Driven Adaptive Scheduling

In multi-rail networks, bandwidth aggregation is a common optimization to fully utilize parallel links. TENT splits large request into multiple *slices*, with each slice independently assigned to a path formed by a pair of local and remote NICs. Efficient utilization of heterogeneous transports requires that both the slice size and the selected NIC pair be determined dynamically. TENT achieves this through telemetry-driven adaptive scheduling, continuously refining its decisions based on historical performance feedback. By combining path scoring with quota enforcement, TENT intelligently distributes slices to maximize throughput while ensuring fairness across concurrent processes.

3.3.1 *Weighted Slice Scheduling*

TENT divides large requests into multiple slices, each with a minimum size (64 KB by default). To maintain both flexibility and low tracking cost, we also limit the number of slices per request. For very large requests, slice sizes grow proportionally to reduce software overhead, striking a balance between granularity and efficiency.

When routing a slice, TENT first selects the reachable local NICs by consulting the local topology. Each candidate NIC is assigned a score computed as the product of three factors: the load factor, which reflects the NIC's current utilization relative to its capacity; the latency factor, derived from historical transfer latency; and the affinity factor, which accounts for NUMA and PCIe proximity to the source memory. The slice is then dispatched to the NIC with the highest score, ensuring adaptive, topology-aware, and balanced utilization of available links. After each slice transfer, TENT updates the NIC's latency estimate using an exponentially weighted moving average (EWMA) algorithm.

After selecting the optimal local NIC for a slice, TENT chooses a corresponding remote NIC. TENT first determines the candidate set based on topology; if a remote NIC mapped to the chosen local NIC falls within this set, it is selected by default. Otherwise it fallbacks to simple round-robin. TENT favors a one-to-one mapping, which can be determined using one of three strategies: (1) NUMA-aware NIC enumeration, (2) pairing NICs according to GPU affinity, or (3) using pretest latency measurements to infer optimal matches via a maximum-weight bipartite matching algorithm. This rail-aware strategy ensures each slice takes the most direct path, maximizing parallel NIC use while avoiding unnecessary switch hops.

3.3.2 *Cross-Process Fairness*

To avoid resource exhaustion in multi-tenant workloads, TENT enforces a quota-based fairness mechanism across processes on the same machine. Global quotas are applied during scheduling to ensure fair access to network resources. For instance, in KVCache transfer workloads, a typical deployment launches one process per GPU, and the quota mechanism prevents any single process from saturating NICs, ensuring balanced utilization.

TENT uses shared memory to maintain global quota state, allowing each process to directly update NIC usage in a decentralized manner. To further reduce scheduling overhead, TENT employs a two-layer quota system: the first layer ensures fair usage of local NICs within each process, and the second layer coordinates quotas across processes on the same machine, preventing any single process from monopolizing high-performance links.

3.4 Proactive Dual-Layer Resilience

A declarative engine must guarantee data movement intent even under failures. TENT achieves this with a proactive, self-healing dual-layer design.

3.4.1 *Link-Level Resilience*

For multi-rail transports (e.g., RDMA), each connection is represented by an EndPoint, which abstracts the link between a local NIC and a remote NIC. During normal operation, when a new connection is requested, an EndPoint is initialized and becomes ready to handle full-duplex communication. TENT continuously monitors all EndPoints for transmission errors. When an EndPoint experiences a failure, it is temporarily excluded from scheduling: new slices are routed through healthy EndPoints, while in-flight slices are either retried or completed according to their status. Once the connection is restored, the EndPoint is reinitialized and becomes eligible for scheduling again.

Another type of transmission failure occurs when the selected NIC's external link is temporarily or permanently unreachable. This can result in timeouts or connection failures even before data transfer begins. TENT handles such cases flexibly: the affected path is temporarily deprioritized, and subsequent scheduling favors alternative healthy EndPoints. This soft exclusion ensures that transfers continue without interruption while avoiding unnecessary retries on unreliable links.

For a slice that failed to deliver in the first attempt, multi-rail transports may perform additional attempts on different paths. In this case, the routing strategy differs from that described in Section 3.3. Specifically, TENT enumerates all reachable combinations of local and remote NICs and retries the transfer for a limited number of times. TENT also employs a timeout mechanism to prevent stalled transfers.

3.4.2 *Transport-Level Resilience*

In addition to individual link failures, a transport backend may also fail due to hardware faults or software issues. TENT handles such transport-level failures by transparently retrying transfers over alternative transport backends, as identified during Dynamic Per-Request Orchestration. This process requires no intervention from the application and ensures continuous progress despite backend-level faults. By combining with link-level and transport-level resilience, TENT provides a resilient foundation for high-performance, multi-rail, heterogeneous GPU clusters.

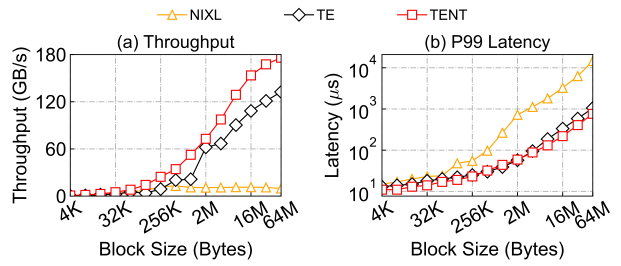
4 Preliminary Results

We evaluate TENT on a two-node GPU cluster. Each node is equipped with eight NVIDIA H800 GPUs interconnected via NVLink, dual-socket Intel Xeon Platinum 8468V CPUs, and eight 200 Gbps RoCE NICs for inter-node communication. To assess TENT's effectiveness, we compare it against existing transfer engines, including TE [15] and NIXL [11].

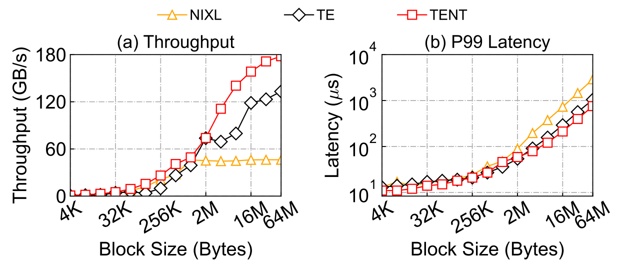
4.1 Basic I/O Benchmark

To benchmark both TE and TENT, we developed TEBench, a framework for measuring data transfer performance. Figure 3 illustrates the throughput and tail latency of one-sided reads and writes, where each experiment issues synchronous transfer requests repeatably with block sizes ranging from 4 KB to 64 MB, using two concurrent threads. For comparison, NIXL is evaluated using NIXLBench under the same workload.

For both read and write operations, TENT outperforms TE in terms of throughput by 32.9% and 33.7%, respectively. By default, NIXL employs the UCX backend with single-threaded worker model,



(a) Reads



(b) Writes

Figure 3: Throughput and latency for different block sizes.

which centralizes communication progress and thus becomes a bottleneck under aggregate transfers, particularly for read operations. Compared with TE, TENT leverages the Telemetry-Driven Adaptive Scheduling technique to reduce software overhead due to request slicing, and prevent bandwidth underutilization caused by imbalanced slice scheduling. Notably, TENT requires only four backend worker threads to sustain its performance, demonstrating higher efficiency compared to TE and NIXL. In addition, TENT achieves up to 30.5% lower P99 latency than TE, suggesting more efficient request handling and enhanced predictability of performance.

4.2 KVCache Transfer I/O Benchmark

We model the I/O pattern of KVCache transfers and evaluate performance using the DeepSeek-R1-W8A8 model [8] with a 4K input as a representative workload. The model comprises 61 layers, each containing 32 blocks of 144 KB, consisting of a 128 KB NoPE block and a 16 KB RoPE block. To evaluate performance under the worst-case scenario, we assume that all blocks are discontiguous, generating a separate request for each block. We use multithreading to accelerate KVCache transfers. The arrival rate is unlimited.

Figure 4 shows the aggregated bandwidth and P99 latency as the number of submission threads varies. The reported latency measures the transfer time of a single layer. TE performs worse mainly because its random scheduling leads to uneven NIC utilization when concurrency increases, which in turn causes queuing delays and higher latency. In terms of throughput, TENT fully utilizes the bandwidth capacity of all links, achieving up to 4.07× that of TE. Regarding P99 latency, TENT reduces it by 31.3% ~ 85.6% compared to TE, further mitigating the impact of KVCache transfers on inference performance.

4.3 End-to-End Performance with Checkpoint Engine

Moonshot AI's checkpoint engine [1, 18] is used in RL-based LLM training by updating parameters in the inference cluster. It has been already integrated with TE and used in the training of Kimi K2. We evaluate the end-to-end performance gains of TENT over TE using the checkpoint engine. We evaluate two representative models with TP=8 and PP=1, using the RDMA transport backend for both TE and TENT.

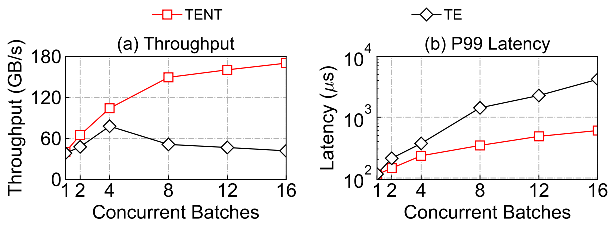


Figure 4: KVCache transfer I/O benchmark.

Table 1: Update time in seconds with Checkpoint Engine.

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

As shown in Table 1, TENT achieves speedups of 50.5% and 31.0% respectively. This improvement is mainly due to TENT's optimizations for typical elephant flows, which enable more efficient utilization of RDMA resources and reduce transfer latency.

5 Related Works

xCCL. Collective communication libraries [10, 13, 21] provide a widely used abstraction in distributed training and inference, implementing predefined patterns such as *broadcast*, *gather*, and *reduce*. NCCL [3, 10], maintained by NVIDIA, supports GPU-to-GPU transfers via GPUDirect RDMA and NVLink, but its cross-node topology awareness and fault tolerance remain limited. UCCL [21] can directly replace NCCL and achieve higher performance through unreliable RDMA datagrams. However, collective communication libraries are not well suited for KVCache transfers.

RDMA Performance Tuning. To fully exploit RDMA bandwidth and reduce latency, prior works [5, 20] summarize techniques such as doorbell batching, message inlining, and QP multiplexing. Microarchitectural analysis of RDMA NICs further identifies performance bottlenecks under multi-tenant workloads [6, 19]. Scale-out RDMA networks also face challenges in QP scalability, which has been addressed via UD transport modes (e.g., eRPC [4]), dynamic connection transport (DCT), and QP throttling techniques (e.g., Flock [9] and SMART [16]), each balancing throughput and resource usage.

6 Conclusion

Large-scale LLM serving exposes the limits of conventional transfer engines, whose imperative path selection, fixed scheduling, and fragile fault handling cannot keep up with heterogeneous GPU clusters. TENT addresses these challenges via declarative slice spraying: it automatically discovers interconnect topologies, abstracts them into a unified resource pool, and adaptively orchestrates data slices across multiple transports. Dual-layer resilience ensures self-healing under failures. Evaluation shows that TENT reduces tail latency by up to 30.5%, validating its effectiveness for high-performance, heterogeneous LLM serving.

References

[1] Moonshot AI. Checkpoint engine. https://github.com/MoonshotAI/checkpoint-engine, 2025.

[2] DeepLink. Dlslime: Flexible & efficient heterogeneous transfer toolkit. https://github.com/DeepLink-org/DLSlime, 2025. BSD-3-Clause License; toolkit for peer-to-peer communication via RDMA, NVLink, NVShmem etc.; supports heterogeneous transfer engines and slicing behaviour.

[3] Zhiyi Hu, Siyuan Shen, Tommaso Bonato, Sylvain Jeaugey, Cedell Alexander, Eric Spada, James Dinan, Jeff Hammond, and Torsten Hoefler. Demystifying nccl: An in-depth analysis of gpu communication protocols and algorithms. *arXiv preprint arXiv:2507.04786*, 2025.

[4] Anuj Kalia, Michael Kaminsky, and David Andersen. Datacenter rpcs can be general and fast. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 1–16, 2019.

[5] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design guidelines for high performance rdma systems. In *2016 USENIX annual technical conference (USENIX ATC 16)*, pages 437–450, 2016.

[6] Xinhao Kong, Jingrong Chen, Wei Bai, Yechen Xu, Mahmoud Elhaddad, Shachar Raindel, Jitendra Padhye, Alvin R Lebeck, and Danyang Zhuo. Understanding rdma microarchitecture resources for performance isolation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 31–48, 2023.

[7] Heng Liao, Bingyang Liu, Xianping Chen, Zhigang Guo, Chuanning Cheng, Jianbing Wang, Xiangyu Chen, Peng Dong, Rui Meng, Wenjie Liu, et al. Ub-mesh: a hierarchically localized nd-fullmesh datacenter network architecture. *arXiv preprint arXiv:2503.20377*, 2025.

[8] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[9] Sumit Kumar Monga, Sanidhya Kashyap, and Changwoo Min. Birds of a feather flock together: Scaling rdma rpcs with flock. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 212–227, 2021.

[10] NVIDIA. Nccl (nvidia collective communications library). https://github.com/NVIDIA/nccl, 2025.

[11] NVIDIA. Nvidia inference xfer library (nixl). https://github.com/ai-dynamo/nixl, 2025.

[12] NVIDIA. Nvlink and nvlink switch. https://www.nvidia.com/en-us/data-center/nvlink/, 2025.

[13] PyTorch. Gloo: Collective communications library with various primitives for multi-machine training. https://github.com/pytorch/gloo, 2025.

[14] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, et al. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, 2024.

[15] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA, February 2025. USENIX Association.

[16] Feng Ren, Mingxing Zhang, Kang Chen, Huaxia Xia, Zuoning Chen, and Yongwei Wu. Scaling up memory disaggregated applications with smart. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 351–367, 2024.

[17] Zhenghang Ren, Yuxuan Li, Zilong Wang, Xinyang Huang, Wenxue Li, Kaiqiang Xu, Xudong Liao, Yijun Sun, Bowen Liu, Han Tian, et al. Enabling efficient gpu communication over multiple nics with fuselink. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pages 91–108, 2025.

[18] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[19] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, et al. Srnic: A scalable architecture for rdma nics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1–14, 2023.

[20] Xingda Wei, Xiating Xie, Rong Chen, Haibo Chen, and Binyu Zang. Characterizing and optimizing remote persistent memory with rdma and nvm. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 523–536, 2021.

[21] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, et al. An extensible software transport layer for gpu networking. *arXiv preprint arXiv:2504.17307*, 2025.