

CONNEX: Endpoint Mobility Primitives for Dynamic LLM Serving

Yanying Lin^{†‡§}, Vincent Liu[§], Tao Luo[§], Chengzhong Xu[¶], and Kejiang Ye^{†*}

[†]Shenzhen Institutes of Advanced Technology, CAS [‡]University of Chinese Academy of Sciences

[§]University of Pennsylvania [¶]University of Macau

ABSTRACT

Modern LLM serving systems increasingly adopt elastic inference pipelines where stages frequently join, leave, and migrate across nodes. However, existing GPU communication frameworks like NCCL assume static topologies, causing routing failures and P99 latency spikes during worker transitions that violate sub-millisecond tail latency requirements. We present CONNEX, a communication system that elevates endpoint mobility from exceptional failure to first-class primitive. Rather than optimizing individual mechanisms in isolation, CONNEX defines a *mobility contract* that the communication layer enforces whenever workers join, leave, or migrate while token streams, activations, or KV transfers are in flight. The contract is realized through three cooperating mechanisms: (1) epoch-based routing that bounds staleness without global coordination, (2) explicit handover protocols that preserve stream ordering and provide exactly-once delivery across migrations, and (3) credit-based backpressure with traffic-class isolation that prevents churn-induced interference with latency-critical paths. Evaluation on a 5-node GPU cluster under synthetic and production-derived churn shows that CONNEX reduces P99 tail spikes by up to 85% compared to NCCL-based baselines, achieves sub-second cutover, and maintains 100% goodput at moderate loads where baselines collapse to 0–28%, while incurring less than 5% steady-state overhead.

CCS CONCEPTS

• **Networks** → **Network algorithms**; **Network mobility**; Network dynamics.

KEYWORDS

GPU Communication, Endpoint Mobility, Elastic Inference, Communication Primitives

ACM Reference Format:

Yanying Lin, Vincent Liu, Tao Luo, Chengzhong Xu, and Kejiang Ye. 2026. CONNEX: Endpoint Mobility Primitives for Dynamic LLM Serving. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26)*, August 17–21, 2026, Denver, CO, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3789240.3829200>

*Corresponding author.

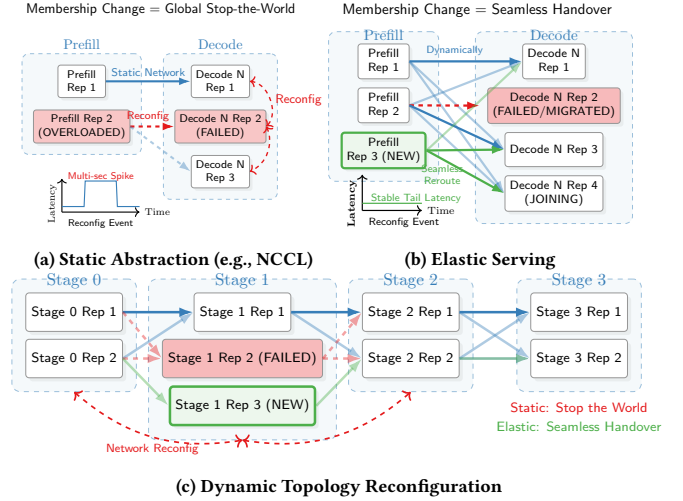


Figure 1: Static communication frameworks vs. elastic serving: (a) Static prefill-decode requires stop-the-world reconfiguration on any node churn, (b) Elastic serving demands dynamic load balancing and non-uniform scaling across heterogeneous nodes, (c) Pipeline parallelism needs stage-specific adaptation to request patterns and resource constraints.

1 INTRODUCTION

Modern LLM serving increasingly hinges on elasticity. To handle fluctuating availability, demand, and resource constraints, the serving infrastructure may need to support workers joining, leaving, and migrating across nodes dynamically.

Opportunities and obligations for elasticity span the entire inference pipeline. A key benefit of Prefill-Decode (PD) disaggregation [35, 54], for instance, is the ability to scale prefill and decode stages independently based on workload characteristics. Similarly, recent proposals for pipeline parallelism in inference [26, 33] benefit from stage-level worker rebalancing as different pipeline stages experience varying computational loads and memory pressure. Finally, fault tolerance and preemption require immediate worker migration, as any delay directly affects service availability and user experience.

Unfortunately, today’s inference frameworks are ill-equipped for dynamic membership changes (Figure 1). Part of the issue lies in the management of collectives, e.g., the possibility of deadlock and the challenges of handling partially completed transfers [44]. Potential workarounds include building custom, application-level communication strategies on top of point-to-point transfers; however, even after addressing these problems, existing GPU communication sub-systems are simply not designed to handle churn.

NCCL is both the best and most common example of this limitation. In NCCL, communication groups are configured and built a priori. Any change to those groups requires a wholesale global

reconfiguration, including communicator rebuilding, rank reassignment, and barrier synchronization, which imposes multi-second disruptions that take all involved GPUs out of commission while the system reorganizes.

We note that recently proposed alternatives to NCCL, such as UCCL [55], MoonCake [38], and NCCLX [40], also suffer from these limitations. While they do improve transmission, routing, and scheduling during steady-state operation, these are on top of fixed communication group topologies. In the end, support for true elasticity is often a secondary concern.

In this paper, we investigate the question: *How do we build a communication framework that enables and supports dynamic elasticity in LLM serving?* What are the requirements? What does such a system look like? What are the potential benefits? In answering these questions, we identify three issues that make elasticity in LLM serving difficult under today’s communication abstractions:

C1. Broken connections and routing state. Under churn, established connections and existing routing information inevitably become stale: senders transmit to old addresses, receivers disappear, and systems enter a convergence window with blackholes and retry storms. Existing stacks either (i) require global reconfiguration (barriers/communicator rebuild) to restore consistency, or (ii) push staleness handling to applications through timeout-based retry mechanisms that propagate failures across dependent communication paths. Both approaches can incur massive performance penalties in critical-path request processing.

C2. Non-existent handover for in-flight requests. In pipeline serving, requests form token-iterative streams that may span topology changes. Migration introduces unavoidable transient behaviors such as dual delivery paths, reordering, duplication, and partial delivery. Current systems lack an explicit handover protocol that defines how in-flight streams transition across endpoints while preserving ordered continuation and idempotent replay. Applications are forced to either drain globally (availability loss) or drop/restart requests (correctness loss).

C3. Congestion caused by migration traffic. Dynamic membership causes bottlenecks to shift unpredictably across endpoints. Without enforceable flow control, worker migrations and background data movement can overwhelm receiver queues, blocking latency-critical traffic through head-of-line blocking. Existing monitoring-based congestion signals are too slow or coarse-grained to drive real-time routing and scheduling decisions. Elastic communication requires actionable backpressure, namely explicit capacity signals that (i) bound queue growth during churn, (ii) prioritize critical traffic classes, and (iii) guide load-aware routing decisions.

Our solution. We present CONNEX, a mobility-first communication layer that makes endpoint mobility a first-class primitive for elastic LLM inference. CONNEX enforces a mobility contract that guarantees stable logical identities, ordered continuation, and local reconfiguration, using three key mechanisms:

Epoch routing with bounded staleness. CONNEX decouples logical endpoints from physical locations via stable endpoint IDs (LIDs). Each topology change advances a routing epoch. Messages carry (LID, epoch) so receivers can detect stale routes and trigger fast redirection. This provides a bounded staleness window without

global barriers: reconfiguration becomes a local route update rather than communicator rebuild.

Explicit handover protocol for ordered continuation. CONNEX defines handover semantics for in-flight streams using sequence numbers, idempotent delivery, and selective retransmission. During migration, senders temporarily enable dual-path cutover (or shadow warmup) while receivers enforce ordered delivery and deduplication. Handover completes locally between adjacent stages without draining the entire pipeline.

Credit-based backpressure with mobility-aware scheduling. CONNEX provides enforceable flow control via credit grants tied to receiver capacity (GPU buffers/compute pace). Traffic is classified (latency-critical token streams vs background transfers), scheduled in separate queues, and paced by credits to bound queue growth. Credit signals also feed routing choices during churn to avoid overloaded endpoints.

CONNEX implements these mechanisms through a three-layer architecture: (1) a lightweight ZeroMQ-based control plane for topology management using stable logical endpoints, (2) a high-performance UCX/RDMA data plane with automatic transport selection, and (3) a mobility and backpressure semantics layer that provides dynamic communication guarantees. Crucially, CONNEX maintains transport independence, meaning that the endpoint mobility semantics work consistently across different underlying transports, from RDMA to TCP fallback.

We evaluate CONNEX across diverse workloads with synthetic and production-like churn patterns. Our results show that CONNEX reduces P99 latency spikes during worker transitions by up to 85% compared to NCCL-based baselines, while achieving cutover times under 200ms even for large model migrations. Under steady-state conditions, CONNEX maintains performance parity with existing frameworks, demonstrating that mobility support does not compromise baseline efficiency.

Contributions. Our key contributions are:

- We formalize endpoint mobility as a missing communication primitive for elastic LLM serving, capturing three required semantics (bounded-staleness routing, seamless in-flight handover, and actionable backpressure) to preserve tail-latency invariants under churn.
- We design and implement CONNEX, a mobility-first communication layer that enforces the mobility contract through epoch routing, explicit handover, and credit-based backpressure, while remaining transport-pluggable.
- We demonstrate substantial improvements under churn, reducing join-ready and cutover latency, suppressing retry storms, and significantly lowering P99 spikes, while achieving comparable steady-state performance.

2 BACKGROUND AND MOTIVATION

2.1 Elastic Serving in Practice

Worker churn is the steady-state operating mode in multi-tenant LLM inference, not a rare exception. We identify three representative use cases where dynamic membership changes are unavoidable.

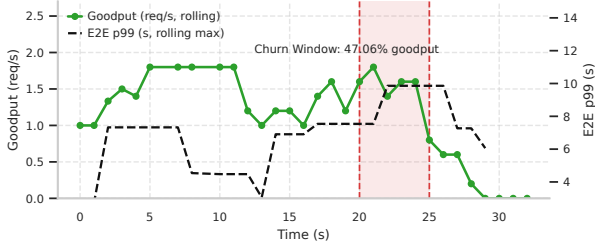


Figure 2: Worker failures cause P99 latency spikes in inference pipelines. A single replica failure triggers cascading timeouts upstream and downstream, degrading tail latency.

Use case 1: Prefill-decode disaggregation. DistServe [54] and Splitwise [35] separate prefill and decode stages onto dedicated GPU pools, allowing each to scale independently. During load spikes, prefill workers are added without interrupting active decode sessions; when more sequences run concurrently, decode capacity grows while prefill workers drain. KV cache transfer is the main inter-stage communication: prefill workers send large KV tensors (tens to hundreds of MB) to decode workers via RDMA or TCP [38]. Since prefill and decode pools scale at different times and rates, communication endpoints change frequently.

Use case 2: Pipeline-parallel inference. Pipeline parallelism partitions an LLM into multiple stages, each running on a separate GPU or node [26, 33]. At every token step, small activation tensors are passed in strict order between stages, requiring low-latency delivery. In practice, many sequences are processed at once; systems send multiple logical activation streams over a limited set of physical network connections to increase utilization and stay within resource limits. When workloads shift—such as a stage becoming a bottleneck or scaling out—the set of active pipeline stages changes on the fly. Elasticity here requires seamless rerouting and stream remapping in flight to preserve ordered token delivery and prevent one sequence’s delays from impacting others.

Use case 3: Fault tolerance and spot preemption. Multi-tenant GPU clusters experience continuous resource churn. Workload characterization studies report GPU failure rates of 1–5% per day in large-scale clusters [14, 48], while spot instance preemptions can revoke capacity with as little as 30 seconds’ notice [32]. In either case, the serving system must migrate affected workers immediately, because any delay directly translates to dropped requests and SLO violations. SpotServe [32] and Bamboo [43] demonstrate the need for preemption-resilient serving, but both rely on heavyweight global repartitioning when workers depart.

Churn events are both frequent and often correlated. Auto-scaling or spot preemption can simultaneously trigger joins or leaves across many pipeline stages, creating bursts of endpoint changes every few seconds or minutes—while user-facing latency targets remain at the millisecond level.

2.2 Today’s Frameworks Under Churn

We now examine how the dominant GPU communication stack, NCCL as integrated into PyTorch [34] and frameworks like vLLM [18], operates in the common case, and then show concretely why it breaks under churn.

2.2.1 NCCL: Architecture and Normal-Case Operation. NCCL, the NVIDIA Collective Communication Library, provides optimized GPU-to-GPU data movement through collective operations (AllReduce, AllGather, Send/Recv) over NVLink, PCIe, and InfiniBand/RoCE fabrics. Its programming model is built on three core abstractions:

- **Communicators** (`ncclComm_t`): An opaque handle representing a fixed group of GPUs. Created via `ncclCommInitRank`, which performs a global handshake: every participating rank must call this function with the same `ncclUniqueId` and total rank count. Internally, the initialization discovers the interconnect topology (NVLink meshes, PCIe switches, network cards), selects algorithms (ring, tree, or collnet), and pre-allocates channel buffers. This process takes hundreds of milliseconds to seconds, depending on the topology complexity.
- **Ranks:** Each GPU is assigned a fixed integer rank $\in [0, n_{\text{ranks}})$ at communicator creation time. Rank assignments determine the communication pattern: ring algorithms route data through rank $(r + 1) \bmod n$; tree algorithms assign parent/child roles based on rank numbering. Ranks are immutable for the lifetime of the communicator.
- **Streams and groups:** NCCL operations are enqueued onto CUDA streams and can be grouped via `ncclGroupStart/End` for coalescing. Operations within a group are launched atomically, but the group boundaries assume all ranks are live and responsive. A hung or departed rank causes the entire group to block or time out.

In a typical pipeline-parallel serving deployment, the framework creates one NCCL communicator per adjacent stage pair (e.g., stage 0↔stage 1, stage 1↔stage 2, etc.). At each token step, stage i calls `ncclSend(activation, dst_rank, comm, stream)` and stage $i+1$ calls `ncclRecv(buf, src_rank, comm, stream)`. For PD disaggregation, a communicator connects each prefill-decode pair for KV cache transfer. In steady state, this works well: NCCL achieves near-line-rate throughput with low latency by leveraging pre-computed topology-aware routing and pre-registered GPU memory buffers.

2.2.2 What Breaks Under Churn. The problem surfaces when any rank in a communicator departs, whether due to failure, preemption, or deliberate scale-down. We illustrate with a concrete scenario shown in Figure 2: a 4-stage pipeline with 2 replicas serving Llama3-13B, where one stage-2 worker fails at $t=20s$ under QPS=4.

C1: Broken connections and routing state. When the stage-2 worker on replica 1 crashes, its NCCL rank becomes unreachable. Stage 1 workers that have already called `ncclSend` on the shared communicator now block waiting for the peer’s `ncclRecv`, since NCCL’s internal point-to-point protocol uses a rendezvous handshake: the sender writes a `ConnectAccept` message to a shared memory/network buffer and spins until the receiver posts a matching `ConnectReply`. With the receiver gone, the sender spins until NCCL’s internal watchdog fires (`NCCL_TIMEOUT`, default 300s, typically reduced to 5–30s in serving deployments). Meanwhile, stage 3 workers waiting for activations from the departed stage 2 also block. The net effect is that a single departed rank wedges communication on *all* ranks that share any communicator with it, including ranks

Static Assumptions	Elastic Reality
Fixed membership	Continuous churn
Stable routes	Stale routing windows
Long-lived connections	Ephemeral endpoints
Stationary bottlenecks	Moving congestion
Planned reconfiguration	Reactive adaptation

Table 1: Assumption mismatch between static communication frameworks and elastic serving requirements.

on the healthy replica if they participate in the same communicator group.

In our experiment (Figure 2), this manifests as an immediate goodput collapse: all in-flight requests on the affected replica fail, and requests on the healthy replica experience elevated latency as the framework’s scheduling and memory management logic stalls waiting for communication completions.

C2: No handover for in-flight requests. NCCL provides no mechanism to redirect an in-flight `ncclSend/Recv` to a different rank. When stage 2 fails, requests that have already completed stages 0 and 1 on that replica cannot be “handed over” to the surviving replica’s stage 2. The framework has two options: (a) drain the entire pipeline and restart all affected requests from scratch, or (b) drop the in-flight requests and report errors to clients. Both options waste the compute already invested in partial processing and violate latency SLOs.

In frameworks like vLLM, the absence of handover is compounded by the process group abstraction inherited from PyTorch’s `torch.distributed`. Each `ProcessGroup` wraps an NCCL communicator and is tightly coupled to a fixed set of ranks. When a rank dies, the entire process group enters an error state. Recovering requires *destroying all process groups, re-spawning the failed rank, and re-initializing every communicator from scratch*, a sequence that the vLLM community has identified as a fundamental blocker for elastic serving [44]. During this reinitialization window (typically 5–15s in our measurements), no inference can proceed on any pipeline that shared a process group with the failed rank.

C3: Congestion from recovery traffic. After the failed worker is replaced, recovery generates substantial background traffic: model weight reloading, KV cache reconstruction, and NCCL communicator initialization (topology discovery, ring/tree setup, buffer registration). This recovery traffic shares the network with latency-critical activation streams from the healthy replica. Without traffic isolation, recovery traffic saturates receiver queues and causes head-of-line blocking, extending P99 degradation beyond the raw recovery time.

In Figure 2, P99 latency remains elevated for ~8s after failure, far longer than the ~2s needed to detect failure and spawn a replacement. This extended tail results from communicator rebuild overhead and recovery traffic interference.

2.3 Communication Frameworks Under Churn

Recent systems improve specific aspects of GPU communication but share NCCL’s fundamental assumption of stable endpoints (Table 1).

UCCL. UCCL [55] provides a software-defined, extensible transport layer that enables user-space packet spraying, custom congestion

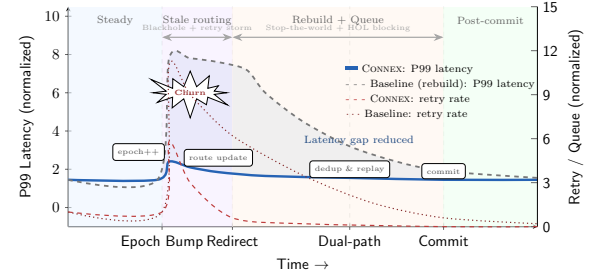


Figure 3: Endpoint mobility creates routing ambiguity: senders retain stale paths while receivers must reconcile overlapping streams during worker transitions.

control, and flexible path selection. It improves steady-state bandwidth utilization by exploiting multi-path diversity within a fixed topology. However, UCCL assumes a relatively stable set of endpoints: its routing and flow-scheduling decisions are derived for a fixed membership and topology. When nodes are added or removed, these decisions must be recomputed and redistributed by the control plane, introducing non-trivial reconfiguration overhead and transient disruption. This contrasts with serving workloads that require seamless, failure-tolerant communication under frequent membership churn.

Mooncake. Mooncake [38] optimizes KV cache transfer for prefill-decode disaggregation through a KV-cache-centric architecture with zero-copy RDMA. Its transfer engine achieves high throughput for large, sequential KV blocks but assumes that prefill-decode endpoint pairs are established before transfers begin. Connection setup involves RDMA queue pair creation and memory registration, which takes tens of milliseconds per endpoint pair. Under churn, each new or migrated decode worker requires fresh connection establishment with every prefill worker that might send it data, creating $O(P \times D)$ setup operations for P prefill and D decode workers.

NIXL. NIXL maximizes network utilization through packet-level optimizations and efficient memory registration. Like Mooncake, it provides high-performance point-to-point transfers but does not define behavior when endpoints disappear mid-transfer. Our measurements (Figure 2) show that when a NIXL-connected worker fails, in-flight RDMA operations complete with errors that propagate up as unrecoverable transport failures, requiring full connection teardown and re-establishment.

The common pattern is that these frameworks solve *how to send faster* within a fixed topology, but none address the semantic questions that churn raises (Figure 3): *where to send* when routes become stale, *how to hand over* in-flight streams during migration, or *how to isolate* recovery traffic from latency-critical paths. Transport efficiency and mobility semantics are orthogonal concerns that must be addressed at different layers.

2.4 Design Principles

We conclude that elastic serving requires *endpoint mobility* as a first-class communication primitive, not merely faster transport but a contract that defines how the communication system behaves

under continuous membership changes. CONNEX enforces three guarantees:

G1: Mobility-safe continuity. Workers may join, leave, or migrate at runtime, yet each stage replica remains a stable logical endpoint. Topology changes must preserve ordered continuation for in-flight request streams rather than forcing global draining or request restarts.

G2: Local, bounded reconfiguration impact. Reconfiguration should not trigger stop-the-world barriers or communicator rebuilds. The performance impact of churn must be localized to affected neighbors and bounded in time, preventing retry storms and long-lived tail spikes.

G3: Actionable performance control under moving bottlenecks. Since churn shifts bottlenecks across stages, the communication layer must provide enforceable backpressure that constrains senders and guides routing decisions, protecting latency-critical token traffic while preserving competitive steady-state throughput.

3 SYSTEM OVERVIEW

CONNEX provides a *communication-layer mobility contract* for elastic LLM serving: when workers join, leave, fail, or migrate while token streams, activations, or KV transfers are in flight, the communication layer guarantees bounded-staleness routing, ordered stream continuation, and enforceable backpressure. The mechanisms described in Section 4 (epoch routing, handover, credit-based flow control, and connection pooling) are the concrete enforcement of this contract rather than independent optimizations. CONNEX contains the following components (Figure 4).

- **Sender/Receiver workers:** User applications running on GPUs implement the data plane and issue send/receive calls to the underlying communication infrastructure. CONNEX supports pluggable data plane transports, including NCCL, UCX, and automatic TCP fallback. Transport selection is driven by hardware capabilities and can adapt to heterogeneous network conditions.
- **Sender/Receiver agents:** An agent running on each machine’s CPU is responsible for machine-to-machine signaling and coordination, including health monitoring, handover procedures, and backpressure flow control.
- **Central directory:** The central directory is a pure control-plane component, separate from data communication and never on the payload fast path. It manages LID-to-(physical address, epoch, capabilities) mappings, triggers epoch updates on membership changes, and handles membership queries or subscriptions. Senders locally cache resolved routes (Section 4.1) and send payloads directly between workers via the selected transport; the directory is only queried on cache miss, epoch mismatch, or subscription update. Our prototype uses a single directory instance; scaling via replication or sharding is future work, as it is independent of the data-plane protocol.

Communication channels. CONNEX is built around point-to-point communication; larger collectives, when necessary, can be built on top of these individual channels. However, partly due to the number of components involved, there are potentially a number of

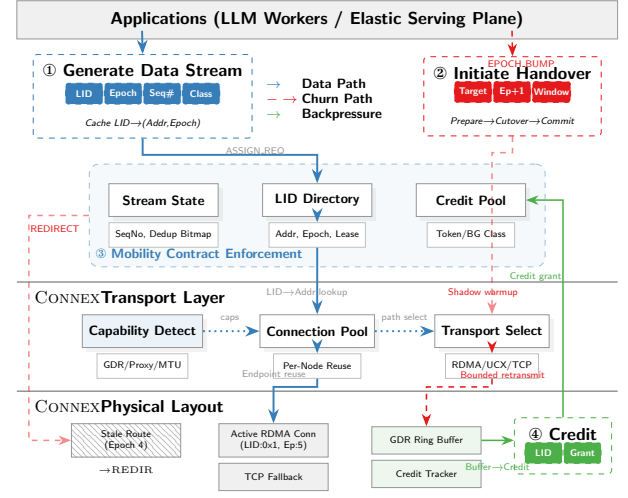


Figure 4: CONNEX architecture.

concurrent data streams in CONNEX at any given moment. Focusing only on the data plane:

- **Payload packets:** Typically sent directly between the sender and the receiver GPU workers, these packets contain the contents of the inter-GPU transfers, preceded by a small header that uniquely identifies the associated stream.
- **Metadata packets:** Sent between the sender and receiver CPU agents, these messages contain metadata about the incoming chunks, including their sequence number, epoch number (Section 4.1), and send timestamp.
- **Background traffic:** This class includes migration transfers, state synchronization, and connection-establishment traffic that can tolerate interruptions and delays.

Credit-based flow control optionally operates per traffic class, enabling receivers to enforce different service levels and prevent head-of-line blocking between classes (especially from background traffic). This isolation is essential because churn events inject unpredictable background traffic that would otherwise overwhelm critical token processing.

Membership queries and subscriptions. When a worker joins the system and wants to register as a receiver, it registers its physical address (e.g., IP:port), transport capabilities (e.g., GPUDirect, UCX, host TCP, etc.), and an application-level type identifier (e.g., denoting a decode instance for model X). The worker will receive an LID at epoch 0 in response. Applications can subscribe to membership changes using these features as filters. For example, prefill instances in a PD disaggregation system might subscribe to a specific application-level type identifier. Senders will automatically subscribe to their partners’ LIDs.

The central directory will broadcast membership changes to all subscribers, though they may also learn of the changes more quickly via in-band epoch verification (Section 4.1).

Typical workflow. Once the application obtains LIDs, the typical operation is as follows, depending on whether the system is in steady-state operation or currently dealing with churn.

Steady-state: Senders use LIDs to specify the destination of communication, fetching and caching the current epoch’s mapping of those location-independent addresses to physical endpoints. On a cache hit, no directory interaction is required for the payload transfer itself. The send operation involves both (a) chunked payload transmission between the send and receive workers, and (b) per-chunk metadata transmission between the send and receive agents. Messages carry explicit stream identifiers and sequence numbers, enabling multiplexed, ordered delivery. Receivers return acknowledgments and credit grants tied to available buffer space and processing capacity.

During churn: Any change in system membership (e.g., scaling up/down the prefill instances in a PD-disaggregation deployment or the migration of a worker from a reclaimed spot instance to an on-demand GPU) triggers an epoch advancement for the involved nodes (Section 4.1) and causes updates to be sent to any subscribed application node.

Newly joined nodes will eventually be integrated into regular operation as applications’ scheduling and load-balancing logic add their LIDs to the rotation. Connections to newly migrated or unavailable nodes will be quickly redirected as controller updates propagate and sender agents encounter epoch mismatches and timeouts. In the case of a migration, CONNEX includes a sub-second handover protocol with support for packet deduplication for exactly once semantics. In the case of a scale-down or crash failure, the exception is immediately passed to the application layer for handling.

4 CONNEX DESIGN

CONNEX constructs an elastic GPU communication substrate through four core mechanisms, each addressing a specific aspect of the challenge: (1) routing consistency under churn, (2) stream continuity during migration, (3) performance isolation under load shifts, and (4) transport efficiency despite frequent reconfiguration.

4.1 Epoch Routing for Fast Convergence

Existing GPU communication frameworks handle membership changes through global barriers or full communicator rebuilds that can take down the entire system. CONNEX instead implements an *epoch-based routing mechanism* that bounds staleness while avoiding global coordination.

With epoch routing, CONNEX transforms membership changes into versioned routing updates. Each logical endpoint (LID) maintains a stable identity across physical migrations, while its routing entry (physical_addr, epoch, lease_expiry, capabilities, health) evolves with topology changes. Data packets carry explicit epochs, enabling receivers to detect staleness and trigger fast convergence.

Data structures. Senders maintain a cache of recent communication partners, mapping (LID, request_id) → RoutingEntry with a TTL-based eviction policy. On the receive side, they maintain the current valid epoch for each of their local endpoints, accept_epoch[LID]. The sender-side cache is an example of soft-state, while the receive-side array exhibits fate sharing.

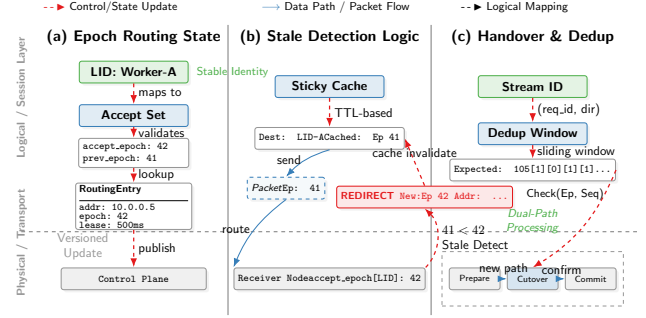


Figure 5: Seamless migration for in-flight streams.

Protocol. When a worker migrates, the control plane advances the destination LID’s epoch and updates the directory. Senders currently using stale cached routes receive immediate REDIRECT(LID, new_epoch, new_addr, new_lease) responses. The receiver’s epoch validation, thus, boils down to three cases:

- epoch < accept_epoch[LID] → immediate redirect
- epoch == accept_epoch[LID] → normal processing
- epoch > accept_epoch[LID] → sender too new, trigger directory refresh

If the worker fails gracefully (i.e., with its agent still active and no migration planned), the agent will send an immediate RESET(LID) response. Crash failures are handled through sender timeouts and missed directory heartbeats.

Bounded staleness guarantee. The staleness window is bounded by several factors, including the lease period and heartbeat frequency, both of which can be tuned to tens to hundreds of milliseconds, depending on the desired detection latency and messaging overheads. Note that epochs are per-destination and per-LID rather than global, ensuring that churn only affects local neighbors rather than triggering system-wide coordination.

4.2 Handover for Seamless Migration

Preserving stream continuity during migration requires explicit coordination between sender and receiver pairs to handle in-flight messages without impacting the broader system. CONNEX implements a *handover protocol* that ensures ordered, duplicate-free delivery across endpoint transitions.

Data structures. Each logical stream is identified by (request_id, direction, stream_type) with explicit sequence numbers. Senders maintain per-stream state tracking unacknowledged ranges, while receivers maintain expected sequence numbers and sliding-window deduplication bitmaps based on (epoch, stream_id, seqno) tuples. This explicit stream identification enables multiplexed communication while preserving ordering guarantees across migrations.

Protocol. The handover protocol is depicted in Figure 5. Migration proceeds through three phases.

- (1) During the **prepare** phase, the sender learns the replacement address either from a directory epoch-update notification or from a REDIRECT response on the old path. The receiver (or its agent) obtains the new LID address from the same epoch update during this phase. The sender then establishes a shadow

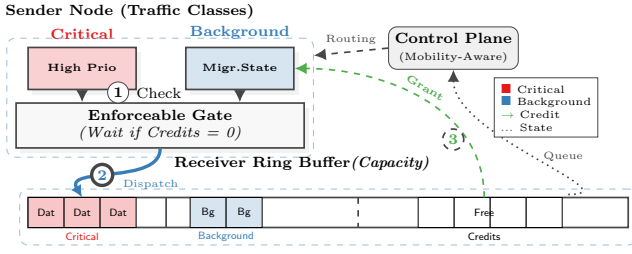


Figure 6: Backpressure and mobility-aware scheduling.

connection to the new destination while continuing normal operation on the old path.

- (2) The **cutover** phase begins when the sender retransmits unacknowledged sequences on the new path; the receiver deduplicates based on the (epoch, seqno) tuple and maintains both paths temporarily to handle overlapping traffic.
- (3) Finally, during the **commit** phase, the receiver confirms the highest contiguous sequence received, enabling sender cleanup and old path teardown.

To prevent excessive bandwidth usage during cutover, retransmission is limited to a sliding window sized to cover typical in-flight volumes ($RTT \times \text{sending rate}$), ensuring that only genuinely unacknowledged data is retransmitted rather than the entire stream history. This window-based approach balances migration completeness with network efficiency: too small a window risks missing legitimate packet loss, while too large a window wastes bandwidth on redundant retransmissions. This bounds both cutover duration and resource overhead while ensuring no message loss. The protocol provides exactly-once delivery guarantees through explicit deduplication, and unlike drain-based approaches that require global quiescence, streams continue processing during cutover, maintaining system utilization. Even under high message rates, CONNEX's approach enables sub-second cutover windows.

4.3 Backpressure for Flow Control

Churn creates moving bottlenecks that shift unpredictably across distributed components, particularly during periods of cutover and high signaling traffic. To address potential queue buildup and HoL blocking of important payloads, CONNEX implements *backpressure*, in which receiver capacity constraints control sending behavior and routing decisions.

Protocol. Figure 6 illustrates the operation of the backpressure mechanism. Credits are managed as **per-peer, per-class local windows**: each sender maintains a separate credit balance for every peer and traffic class (e.g., critical datapath vs. background). Receivers replenish these windows **asynchronously**, either via standalone credit-update messages or **piggybacked** on the 2-byte credit field of the data header. A sender transmits as long as its local window is positive, so **no synchronous RTT is required before each transfer**; backpressure only stalls a sender when its window is exhausted. For instance, critical datapath traffic receives dedicated credit pools and priority scheduling, while background traffic (e.g., migration transfers and state synchronization) operates under separate, restrictive credit limits. Credits are tied to actual ring buffer space, making backpressure naturally enforceable because senders cannot transmit without sufficient credits.

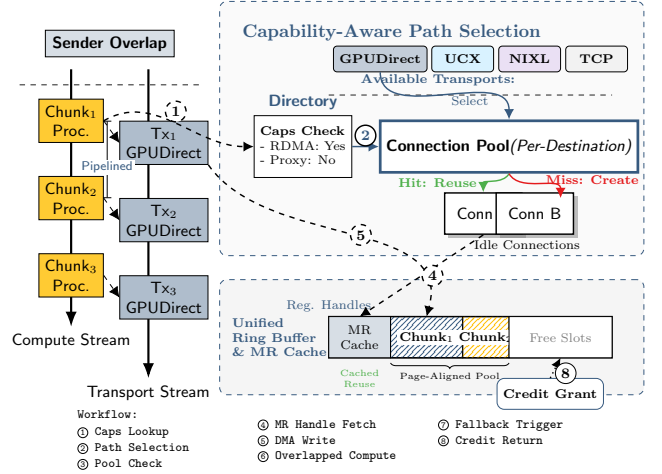


Figure 7: Capability-aware transport with connection pooling. Steps 1–5: capability discovery, pool lookup, and transport selection. Step 6: the sender reuses or creates a connection (rate limited) and sends the payload. Step 7: the connection is returned to the pool for reuse.

Applications aggregate lightweight capacity summaries from receivers, including available credits and queue depths, and use them to inform *new* request or microbatch assignment decisions. Importantly, established streams are not rerouted mid-flight: once a request's activation stream is bound to a destination, it continues on that path until completion or explicit handover. If a sender's credit window is exhausted before the stream finishes, the sender applies local queueing or pacing rather than redirecting to a different peer. New requests avoid endpoints with low available credits or high queuing delays, naturally load-balancing around temporary bottlenecks created by churn. Credit grants adapt dynamically to receiver state, including processing capacity, buffer availability, and current load, creating a feedback loop in which overloaded endpoints automatically reduce their credit grants, throttling incoming traffic and preventing queue explosion.

4.4 Connection Pooling for Efficiency

Providing mobility semantics while maintaining competitive performance requires minimizing the overhead of frequent endpoint changes. CONNEX implements *capability-aware transport selection* with aggressive connection reuse to minimize resource usage even under churn.

The high-level approach is illustrated in Figure 7. Endpoints advertise capabilities through the directory, including GPUDirect RDMA support, maximum inline size, proxy requirements, and memory registration constraints. Senders select optimal transport paths based on these capabilities: direct GPU-to-GPU zero-copy when both endpoints support GPUDirect, host staging for mixed GPU/CPU environments, and proxy-mediated transfers for restricted network configurations.

Rather than establishing fresh connections for each migration, CONNEX maintains per-destination endpoint pools organized by address or node. Cutover operations preferentially reuse existing

connections, reducing join-ready latency from seconds to milliseconds. Pool exhaustion triggers rate-limited connection creation to prevent churn storms. RDMA memory registration overhead is amortized through pre-allocated ring buffers with cached registration handles, page-aligned memory pools for efficient registration reuse, and credit grants that directly correspond to available ring buffer space.

Large data transfers are automatically chunked to enable computation-communication overlap. Chunk sizes adapt to MTU and bandwidth-delay product, balancing throughput and latency. When fast paths are unavailable, CONNEX automatically falls back to alternative transports (UCX transport fallback or TCP) without breaking mobility semantics, ensuring robustness across heterogeneous environments.

4.5 Integration and Failure Handling

Responsibility boundary. CONNEX enforces the mobility contract at the communication layer, while responsibilities that require application semantics remain with the serving framework. Specifically, CONNEX handles endpoint migration (prepare, cutover, commit), transient stale-route detection and redirect, ordered handover with deduplication, credit-based pacing with traffic-class isolation, and transport fallback from RDMA to TCP. The serving framework, in turn, is responsible for permanent KV or model-state loss, sender request-state loss, pipeline rescheduling, and request replay or restart. The guiding principle is that CONNEX guarantees communication continuity whenever a replacement endpoint exists or the affected state can be recovered through retransmission. When state is permanently lost (e.g., GPU memory freed before check-pointing), CONNEX reports the failure upward to the serving layer, which decides whether to replay, reschedule, or abort the affected requests.

Failure escalation. The four mechanisms work together to provide end-to-end mobility guarantees with explicit failure handling at each layer. Epoch routing failures trigger directory refresh through redirect timeouts, with persistent routing failures activating alternative path discovery. Handover failures fall back to request restarts with exponential backoff, while deduplication window overflow triggers a stream reset. Credit exhaustion prevents senders from overwhelming receivers, with persistent credit starvation triggering load balancing. Transport failures activate automatic fallback to alternative transports, and pool exhaustion triggers rate-limited connection establishment.

Unlike existing systems that treat churn as exceptional, CONNEX makes endpoint mobility a first-class primitive with predictable performance characteristics and bounded failure modes. The combination of bounded staleness, explicit handover, enforceable backpressure, and connection reuse ensures that brief membership changes (seconds) cause only bounded performance impact (sub-second cutover windows) rather than persistent degradation.

5 IMPLEMENTATION

CONNEX is implemented as a mobility-first communication runtime comprising three layers: a ZeroMQ-based control plane for dynamic membership and routing coordination, a unified mobility layer

implementing the core contract mechanisms, and pluggable data-plane transports optimized for different workload characteristics.

5.1 Architecture and Transport Selection

Rather than building a monolithic communication stack, CONNEX implements a layered architecture where mobility semantics are provided by a unified middleware layer above pluggable transport backends. This design enables reuse of existing high-performance transports while adding mobility guarantees they lack.

The **control plane** manages dynamic membership, routing epochs, capability discovery, and failure event dissemination using ZeroMQ for lightweight coordination. The **mobility layer** implements the unified contract, including epoch routing, handover protocols, credit-based backpressure, and stream multiplexing, as transport-agnostic mechanisms. The **data plane** consists of pluggable transport adapters: UCX-based Connex Transport Engine (Connex TE) for latency-critical activation streams, and configurable backends (NIXL, Mooncake, or Connex) for KV operations based on environment capabilities.

A key implementation challenge is preventing resource contention between multiple transport backends. CONNEX includes a *Transport Arbiter* that dynamically selects backends and allocates resources based on workload characteristics, network capabilities, and current congestion. For pipeline parallelism with cross-node activation flows, KV transfers are relegated to background priority and rate-limited to prevent interference with token-critical paths. In pure parameter-disaggregation scenarios without cross-node activations, KV-optimized transports like Mooncake can operate at full capacity.

5.2 ConnexTE: Specialized Transport

While existing systems like NIXL and Mooncake excel at bulk KV transfers, activation streams have fundamentally different requirements that necessitate specialized handling. Activation transfers occur at token frequency (every inference step), are latency-critical for pipeline utilization, require strict ordering within request streams, and must support GPU-direct zero-copy for computation-communication overlap.

Connex implements these requirements through UCX with several activation-specific optimizations. Message multiplexing uses (request_id, token_step) stream identifiers to prevent head-of-line blocking between concurrent requests. Credit-driven flow control directly corresponds to GPU ring buffer space, making backpressure enforceable rather than advisory. Memory management uses pre-registered buffer pools and cached UCX memory handles to amortize registration overhead across frequent small transfers. The handover protocol is implemented natively in Connex headers and state machines, enabling sub-RTT cutover without global coordination.

6 EVALUATION

6.1 Experimental Setup

We evaluate CONNEX on a cluster of 5 GPU servers, each equipped with 4 NVIDIA A40 GPUs (48GB memory per GPU) and 100 Gbps Mellanox ConnectX-6 network cards. The servers are connected

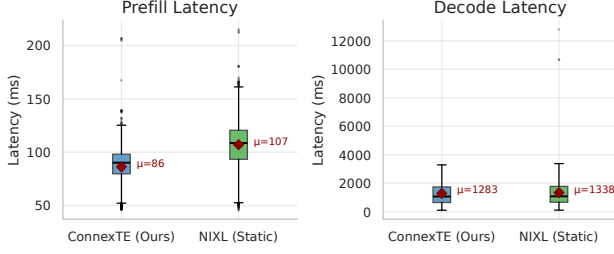


Figure 8: End-to-end performance comparison in PD-Disaggregation: (a) prefill latency and (b) decode latency across different systems.

through a single-switch topology with full bisection bandwidth, representing a typical small-scale inference deployment.

Baseline. We compare CONNEX against four representative baselines: **NCCL** [40] provides optimized collective operations but requires static membership with global reconfiguration on topology changes; **MooncakeTE** [38] optimizes tensor transfers for LLM inference but assumes stable endpoint configurations; **NIXL** maximizes network utilization through packet-level optimizations but lacks dynamic endpoint mobility support; and **ZeroMQ** offers dynamic membership capabilities but provides no GPU-aware optimizations or ordering guarantees for tensor communication. These baselines represent the current state-of-practice across collective communication, inference-specialized transport, network-optimized protocols, and dynamic messaging systems.

Workload. We use the Llama3-13B model [8] for evaluation with request loads constructed from the Splitwise dataset [35], which provides realistic distributions of input and output token counts. We generate both steady-state and bursty traffic patterns by sampling request characteristics (input length, output length, arrival patterns) from this dataset to evaluate system performance under diverse workload conditions. Throughout the evaluation, success rate (also labeled *goodput*) denotes the fraction of issued requests that complete without an application-level restart, client-visible timeout, or dropped stream; a request that is internally retransmitted by CONNEX’s handover protocol but eventually delivered counts as successful, and only requests requiring serving-framework intervention are marked failed.

6.2 Microbenchmarks

E2E Performance in PD-Disaggregation. We evaluate CONNEX in prefill-decode disaggregation, where KV cache transfers between stages create the primary communication bottleneck. This workload tests elastic serving’s core requirement: independent scaling while preserving low-latency tensor transfers.

CONNEX achieves 5.3% better average latency with 100% success rate during scaling events versus NIXL’s connection failures (Figure 8). The advantage stems from mobility-aware routing that eliminates topology-change disruptions.

Breakdown by stage: prefill operations see 19.4% lower latency with stable P99 behavior, while decode operations achieve 3x better P99 latency. This tail latency improvement is critical for maintaining SLA compliance during dynamic scaling.

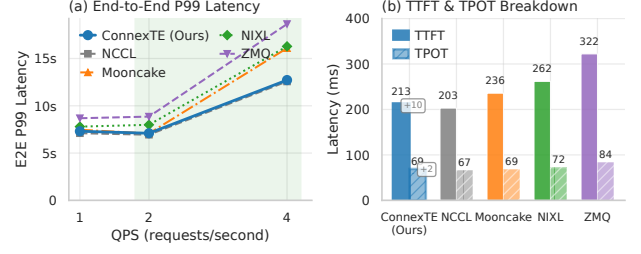


Figure 9: End-to-end performance comparison: (a) P99 latency across QPS levels, (b) TTFT and TPOT breakdown showing CONNEX introduces minimal overhead.

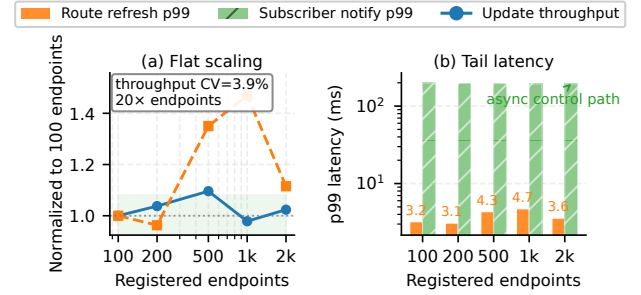


Figure 10: Control-plane scalability: (a) normalized throughput and route-refresh p99 stay flat over 20× scale; (b) route-refresh p99 vs. 10 ms SLO and async subscriber notify.

E2E Performance in Pipeline Parallelism. CONNEX achieves end-to-end performance comparable to NCCL across varying query rates (Figure 9a). At low to moderate QPS (1.0–4.0), CONNEX maintains P99 latencies within 200–400 ms of NCCL’s performance, corresponding to less than 5% of total end-to-end latency at these load levels. This overhead includes mobility protocol processing.

A breakdown of end-to-end latency (Figure 9b) shows that CONNEX introduces minimal additional cost: approximately 10 ms in Time-to-First-Token (TTFT) and 2 ms in Time-per-Output-Token (TPOT) compared to static NCCL configurations. These overheads account for a small fraction of request latency, indicating that mobility guarantees can be achieved with minimal performance impact. Across varying loads, P99 latencies remain within 2.2% of NCCL, confirming that mobility overhead stays bounded and does not introduce additional tail amplification.

Control-Plane Scalability. Our prototype deploys a single front-router/directory instance; replication and sharding are orthogonal to the data-plane mobility protocol and left to future work. To quantify control-plane headroom, we stress-test the directory with 100–2000 mock endpoints, burst update_route operations, and measure update throughput, route-refresh latency, and subscriber notification delay (Figure 10). The directory sustains ~950–1060 updates/s with p99 get_route latency below 8 ms even at 2000 endpoints; subscriber notification p99 remains below 200 ms. These results confirm that the directory is not on the payload fast path, since senders cache routes locally and transmit directly over the selected transport, and does not become a bottleneck at scales well beyond our deployment.

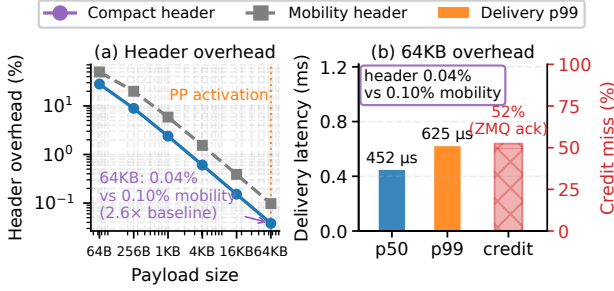


Figure 11: Transfer microbenchmark: (a) compact vs. mobility header overhead (64 KB: 0.04% vs. 0.10%); (b) 64 KB delivery latency and credit miss under ZMQ replenishment.

Transfer and Credit Overhead. We microbenchmark GPU-direct transfers over UCX RDMA between two A40 pods on IB net1, sweeping payload sizes from 64 B to 64 KB (Figure 11). The production fast path uses a 25-byte compact header; the 64-byte mobility header applies only to control-plane churn messages. The compact header imposes 28% overhead on 64 B payloads but only 0.04% on 64 KB activations (mobility header would add 0.10%). At 64 KB, receiver-local delivery latency is 625 μ s (p99, sync path). A separate credit-gated run with receiver-driven ZMQ replenishment (window=32) reports $\sim$ 52% credit miss under sustained sends, reflecting per-ack replenishment latency rather than a per-transfer RTT; credits are per-peer local windows replenished asynchronously.

NCCL Reconfiguration (Shrink) Cost. NCCL 2.27+ introduces `ncclCommShrink` as a lighter alternative to full communicator destroy+reinit on member leave. We microbenchmark both paths on idle A40 GPUs (NCCL 2.27.5, same-node 2-rank communicator) and compare against CONNEX’s pair-local traffic switch (Figure 12).

`ncclCommShrink` completes in $\sim$ 30 ms (p50), while destroy+reinit costs $\sim$ 554 ms, representing an $\sim$ 18 $\times$ gap. The cost gap widens on cross-node PP groups where RDMA bootstrap, queue-pair creation, and NIC memory registration dominate reinit latency. Even the lighter shrink path carries two fundamental constraints that make it unsuitable for elastic serving. First, `ncclCommShrink` requires a *global quiesce*: all ranks in the communicator must drain outstanding collectives before the API can proceed, meaning that a single departing rank blocks progress on every remaining rank that shares any communicator with it. In a multi-replica pipeline deployment, this quiesce propagates to healthy replicas if they share cross-replica communicators (e.g., for AllReduce-based gradient aggregation or KV broadcast). Second, the resulting communicator retains its original topology decisions (ring order, tree structure, channel allocation), which may be suboptimal for the reduced membership; a subsequent rebalance still requires destroy+reinit.

By contrast, CONNEX’s blocking cutover ($\sim$ 11 ms, cross-pod UCX microbench) operates entirely between the affected sender-receiver pair. Unrelated pairs continue transmitting without pause, and the new endpoint inherits transport state (pooled connections, registered memory handles) from the pool rather than re-negotiating from scratch. This locality difference explains why CONNEX sustains 100% goodput during scaling events while NCCL-based deployments suffer cascading stalls proportional to the communicator fan-out of the departed rank.

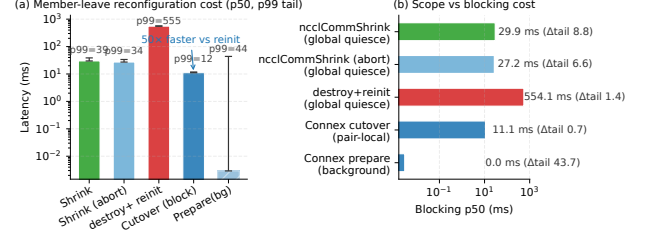


Figure 12: NCCL reconfiguration vs. CONNEX cutover (world size=2): (a) p50 latency with p99 tail for `ncclCommShrink`, destroy+reinit, and CONNEX blocking cutover / background prepare; (b) blocking cost by scope (global quiesce vs. pair-local).

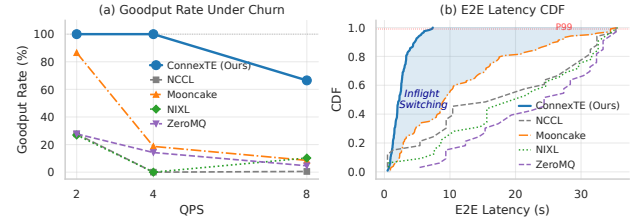


Figure 13: Performance during worker join/leave: (a) Goodput across QPS levels, (b) Latency distribution at QPS=2. CONNEX maintains high success rate and low latency during scaling events.

6.3 Performance in Worker Join/Leave

We evaluate CONNEX’s ability to maintain service continuity during dynamic scaling by measuring goodput and latency during worker join and leave events. We configure a pipeline-parallel serving workload with a scale-down at 30s and a scale-up at 60s, and compare CONNEX against NCCL, Mooncake, NIXL, and ZeroMQ across varying QPS.

As shown in Figure 13a, CONNEX consistently maintains higher goodput during scaling events. At moderate load, CONNEX achieves a 100% success rate, while baselines suffer substantial degradation. As load increases, baselines collapse due to failed reconfiguration and retries, whereas CONNEX continues to sustain high success rates. The latency distribution during the leave window (Figure 13b) further highlights CONNEX’s advantage. CONNEX maintains bounded tail latency and preserves request continuity across cutover, while baselines exhibit severe tail amplification and request drops caused by global reconfiguration and retry storms.

These results demonstrate that CONNEX’s mobility guarantees—**local reconfiguration** (G2) and **continuity-preserving handover** (G1)—enable seamless scaling without global coordination, preventing retry cascades that destabilize static systems under churn.

6.4 Performance in Non-Steady-State Env

We evaluate CONNEX’s resilience under realistic churn scenarios by simulating worker preemption events that mirror spot instance evictions in cloud environments. We configure churn events at varying frequencies (every 30-120 seconds) and measure performance across different load levels.

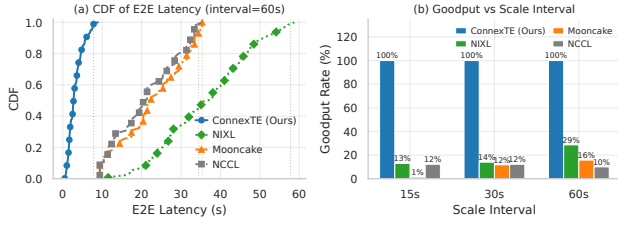


Figure 14: Performance under worker churn: (a) Latency distribution during churn events every 60s, (b) Goodput across churn frequencies.

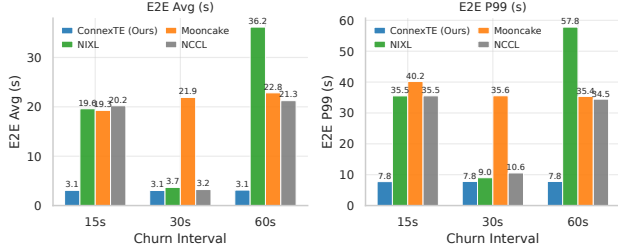


Figure 15: Performance across churn frequencies: (a) Average end-to-end latency remains stable under varying churn intervals, (b) P99 latency shows bounded tail behavior with frequent topology changes.

CONNEX maintains 100% success rate at moderate loads while baselines suffer catastrophic degradation (Figure 14b). At QPS=2, Mooncake drops to 87% success, while NCCL-Reconfig, NIXL, and ZeroMQ collapse to 28% due to failed global reconfiguration. Under higher load (QPS=4), baselines fail completely (0–19% success) while CONNEX sustains perfect reliability. Even at extreme load (QPS=8), CONNEX achieves 66% success versus baselines’ near-zero performance.

CONNEX maintains P99 latency of 7.2s during churn versus baselines’ 21–35s tail degradation (Figure 14a). Across varying churn frequencies, CONNEX exhibits consistent performance—average latency remains 2.5–3.2s with P99 bounded below 8s—while baselines suffer exponential degradation as topology changes accelerate (Figure 15).

The key insight is that CONNEX’s mobility contract enables non-disruptive, localized reconfiguration: workers can join or leave without forcing global synchronization barriers or draining in-flight requests. In contrast, baseline systems either rely on expensive global rebuilds (e.g., NCCL communicator re-initialization) or lack explicit handover semantics, causing in-flight requests to fail and triggering correlated retries during topology changes.

6.5 Bursty Workload Resilience

In GPU-CPU-network heterogeneous architectures, bursty traffic patterns introduce significant challenges that can destabilize communication systems and lead to node failures [44]. To evaluate CONNEX’s resilience under such conditions, we conduct experiments with varying levels of traffic burstiness, with a coefficient of variation (CV) ranging from 1 to 8.

The performance comparison across different CV levels reveals several critical issues in baseline systems (Figure 16). As burstiness increases, the following problems emerge:

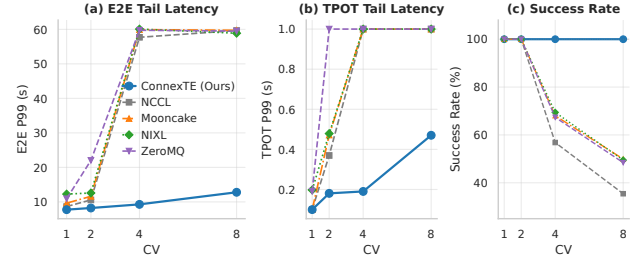


Figure 16: Performance in Bursty Workload: (a) P99 end-to-end latency across different CV levels, (b) P99 TPOT, (c) Goodput rate.

Cascading Communication Failures: At CV=4 and above, baseline systems experience dramatic performance degradation. NCCL’s success rate drops to 56.8% at CV=4 and further deteriorates to 35.4% at CV=8. Similarly, other baselines (Mooncake, NIXL, ZeroMQ) maintain only 48–69% success rates under high burstiness, indicating widespread communication timeouts and connection drops.

Traffic burstiness violates a key implicit assumption made by existing GPU communication stacks: that queueing delays and peer progress remain bounded. Under high CV, transient injection rates exceed network and peer drain capacity, causing buffers to accumulate and progress to desynchronize across ranks. In baseline systems, backpressure is coarse-grained and reactive, triggering only after buffers overflow. This leads to timeouts, correlated retries, and communicator-wide head-of-line blocking, resulting in cascading failures.

CONNEX breaks this failure chain by explicitly decoupling communication progress from instantaneous load and physical placement. First, *actionable backpressure* enforces credit-based flow control *before* enqueue, preventing burst-induced buffer buildup. Second, *mobility-aware scheduling* dynamically redistributes communication load away from congested peers and paths, avoiding global stalls caused by localized slowdowns. Finally, *ordered replay* serializes retries deterministically, eliminating correlated retry storms that amplify latency under congestion.

Together, these mechanisms allow CONNEX to remain stable across burst regimes where baseline systems enter pathological states, explaining its consistent success rate and bounded tail latency even at CV=8.

6.6 System Overhead

CONNEX introduces additional control and coordination overheads in exchange for robustness under bursty traffic and dynamic membership. These overheads fall into three categories.

First, actionable backpressure enforces explicit flow control before enqueue via per-peer, per-class local credit windows replenished asynchronously or piggybacked on headers, so no synchronous RTT is required before each transfer. Under light load this may slightly reduce peak throughput; the cost is fundamental to preventing buffer overflow and cascading timeouts under unbounded burstiness. Second, mobility-aware scheduling introduces modest control-plane overhead to track peer and path states and to redirect traffic away from congested or failed components. This overhead is amortized under stable conditions and becomes active only when

congestion or churn is detected. Third, ordered replay trades immediate retry for deterministic recovery, incurring bounded buffering and reordering delay during failure recovery, but preventing correlated retry storms that dominate tail latency in baseline systems.

Importantly, these mechanisms are dormant on the fast path and are activated only under adverse conditions. As a result, CONNEX matches baseline performance under low burstiness while providing substantial gains in success rate and tail latency under high CV. Microbenchmarks corroborate that this overhead is small in practice: the control plane sustains $\sim 1\text{k}$ updates/s for 2000 endpoints with sub-10 ms route-refresh tails, the 25-byte fast-path header adds $<0.05\%$ tax on 64 KB activations, and cross-pod UCX RDMA delivers 64 KB tensors with $625\ \mu\text{s}$ p99 delivery latency under credit-gated flow control. CONNEX is, thus, best suited for elastic ML serving workloads with bursty traffic and dynamic resource availability, rather than tightly synchronized, topology-stable HPC jobs.

7 RELATED WORK

Elastic and Fault-Tolerant Distributed ML. Elastic training and serving systems handle dynamic resources through various approaches. SpotServe [32] enables LLM serving on preemptible instances, while systems like Ooblock [13], Bamboo [43], and FlexPipe [25] provide resilient training through pipeline adaptation and redundant computation. ServerlessLLM [5], MultiWorld [19], and Sailor [42] optimize serving through checkpoint loading and dynamic parallelism. These systems optimize checkpoint placement, redundant computation, or parallelism strategies under the assumption of stable communication groups. In contrast, CONNEX specifies what the communication layer itself must guarantee when workers join, leave, or migrate with token streams and activations in flight, enabling transparent endpoint transitions with sub-second cutover instead of multi-second global rebuilds.

GPU Collective Communication. NCCL and its extensions [40] provide highly optimized collective operations for GPU clusters but assume static communicator groups with fixed rank assignments. Recent advances include algorithm synthesis (TACCL [39], MSCCL [4]), topology-aware optimization (TACOS [49], TCCL [16], TE-CCL [28]), fault tolerance (Hoplite [56]), and automated tuning (AutoCCL [51], UCCL [55]). These frameworks optimize data transfer throughput and collective scheduling under stable membership; none specifies what the communication layer should guarantee when endpoints change for in-flight streams. CONNEX complements these libraries by providing the point-to-point mobility contract (bounded-staleness routing, ordered handover, enforceable backpressure) that enables elastic inference pipelines where traditional collectives' static membership assumption fundamentally breaks.

RDMA and High-Performance Transport. RDMA-based communication underpins modern GPU cluster networking. Recent advances include efficient inter-node tensor transfer [23], next-generation cloud AI networks [30], scalable NIC architectures [46], and rethinking RDMA interfaces [3]. Systems like FaSST [15], KR-CORE [47], and PeRF [20] optimize RPCs, control planes, and scheduling. Live migration approaches [21, 36, 37] and specialized systems like Mooncake [38] address connection migration and KV cache transfer. Prior transport work optimizes raw data movement,

connection setup, or NIC-level scheduling under stable endpoint configurations. CONNEX leverages these innovations through its pluggable transport plane but adds the mobility contract (epoch routing, ordered handover, credit-based backpressure) that specifies what the communication layer must guarantee when workers migrate or fail with tensor transfers in flight.

Flow Control and Congestion Management. Datacenter congestion control has evolved from end-to-end schemes [17, 50] to credit-based approaches [24, 31] and programmable control [12]. Recent advances include host-based [1] and sub-RTT control [2], backpressure-based [7] and congestion-root approaches [27]. These schemes optimize network-level congestion control under stable topology and flow membership. CONNEX operates at the middleware layer above the network, specifying how credit-based backpressure and traffic-class isolation must interact with endpoint mobility so that churn-induced congestion does not degrade latency-critical inference traffic.

KV Cache Management and State Transfer. Recent work optimizes KV cache handling for disaggregated LLM serving through fast context loading [29], fault-tolerant streaming [41], elastic memory pools [11], and accelerated recovery [6, 52]. These systems optimize what state to transfer and how to cache it under stable serving configurations. CONNEX addresses the orthogonal question of what the communication layer must guarantee during the endpoint transitions that trigger such transfers, ensuring streams and KV flows survive worker changes without application-level restarts.

Membership and Reconfiguration. Dynamic membership management has been extensively studied through microsecond-scale membership services [10], latency-efficient rescaling [9], fast failure recovery [45], flexible checkpointing [22], and sub-second elasticity for web services [53]. These systems address membership tracking and state migration at the orchestration level under stable communication assumptions. CONNEX contributes the missing communication-layer contract that specifies what guarantees (bounded-staleness routing, ordered handover, enforceable backpressure) the transport must provide when workers join, leave, or migrate with in-flight data, complementing higher-level membership services.

8 CONCLUSION

Modern LLM serving systems require dynamic resource allocation and endpoint mobility to handle elastic workloads efficiently. CONNEX provides the missing communication layer that enables seamless endpoint transitions with bounded impact on latency-critical inference traffic, demonstrating significant improvements in P99 latency and system resilience under bursty workloads. This work does not raise any ethical issues.

ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China (No. 2025YFE0204100), Science and Technology Development Fund of Macao S.A.R (FDCT) under number 0074/2025/AMJ, and Guangdong Basic and Applied Basic Research Foundation (No. 2023B1515130002).

REFERENCES

- [1] Saksham Agarwal, Arvind Krishnamurthy, and Rachit Agarwal. 2023. Host Congestion Control. In *Proceedings of the ACM SIGCOMM 2023 Conference*. New York NY USA, 275–287.
- [2] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkkipati. 2023. Bolt: Sub-RTT Congestion Control for Ultra-Low Latency. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI 2023, Boston, MA, April 17–19, 2023. 219–236.
- [3] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. 2021. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. Virtual Event Germany, 228–242.
- [4] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. 2023. MSCCLang: Microsoft Collective Communication Language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Vancouver BC Canada, 502–514.
- [5] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation*, OSDI 2024, Santa Clara, CA, USA, July 10–12, 2024. 135–153.
- [6] Shiwei Gao, Youmin Chen, and Jiwei Shu. 2025. Fast State Restoration in LLM Serving with HCache. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*. 128–143.
- [7] Prateesh Goyal, Preet Shah, Naveen Kr. Sharma, Mohammad Alizadeh, and Thomas E. Anderson. 2019. Backpressure Flow Control. In *Proceedings of the 2019 Workshop on Buffer Sizing*. Palo Alto CA USA, 779–805.
- [8] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs]
- [9] Rong Gu, Han Yin, Weichang Zhong, Chunfeng Yuan, and Yihua Huang. 2022. Mecas: Latency-efficient Rescaling via Prioritized State Migration for Stateful Distributed Stream Processing Systems. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11–13, 2022*. 539–556.
- [10] Rachid Guerraoui, Antoine Murat, Javier Picorel, Athanasios Xygkis, Huabing Yan, and Pengfei Zuo. 2022. uKharon: A Membership Service for Microsecond Applications. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11–13, 2022*. 101–120.
- [11] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, et al. 2024. MemServe: Context Caching for Disaggregated LLM Serving with Elastic Memory Pool. arXiv:2406.17565 [cs]
- [12] Hongjing Huang, Jie Zhang, Xuzheng Chen, Ziyu Song, Jiajun Qin, and Zeke Wang. 2025. SwCC: Software-Programmable and Per-Packet Congestion Control in RDMA Engine. In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7–9, 2025*. 1243–1260.
- [13] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. 2023. Ooblock: Resilient Distributed Training of Large Models Using Pipeline Templates. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 382–395.
- [14] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, unjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Conference on Unix Annual Technical Conference*. USA, 947–960.
- [15] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2–4, 2016*. 185–201.
- [16] Heehoon Kim, Junyeol Ryu, and Jaemin Lee. 2024. TCCL: Discovering Better Communication Paths for PCIe GPU Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. La Jolla CA USA, 999–1015.
- [17] Gautam Kumar, Nandita Dukkkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, et al. 2020. Swift: Delay Is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Virtual Event USA, 514–528.
- [18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, et al. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 611–626.
- [19] Myungjin Lee, Akshay Jajoo, and Ramana Rao Kompella. 2024. Enabling Elastic Model Serving with MultiWorld. arXiv:2407.08980 [cs]
- [20] Sugi Lee, Mingyu Choi, Ikjun Yeom, and Younghoon Kim. 2024. PeRF: Preemption-enabled RDMA Framework. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10–12, 2024*. 209–225.
- [21] Xiaoyu Li, Ran Shu, Yongqiang Xiong, and Fengyuan Ren. 2025. Software-Based Live Migration for RDMA. In *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, S ao Francisco Convent, Coimbra, Portugal, September 8–11, 2025*. 99–113.
- [22] Xinyu Lian, Sam Ade Jacobs, Lev Kurilenko, Masahiro Tanaka, Stas Bekman, Olatunji Ruwase, and Minjia Zhang. 2025. Universal Checkpointing: A Flexible and Efficient Distributed Checkpointing System for Large-Scale DNN Training with Reconfigurable Parallelism. In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7–9, 2025*. 1519–1534.
- [23] Nandor Licker, Kevin Hu, Vladimir Zaytsev, and Lequn Chen. 2025. RDMA Point-to-Point Communication for LLM Systems. arXiv:2510.27656 [cs]
- [24] Hwijoon Lim, Jaehong Kim, Inho Cho, Keon Jang, Wei Bai, and Dongsu Han. 2023. FlexPass: A Case for Flexible Credit-based Transport for Datacenter Networks. In *Proceedings of the Eighteenth European Conference on Computer Systems*. Rome Italy, 606–622.
- [25] Yanying Lin, Shijie Peng, Chengzhi Lu, Chengzhong Xu, and Kejiang Ye. 2025. FlexPipe: Adapting Dynamic LLM Serving Through Inflight Pipeline Refactoring in Fragmented Serverless Clusters. arXiv:2510.11938 [cs]
- [26] Yanying Lin, Shijie Peng, Shuaipeng Wu, Yanbo Li, Chengzhi Lu, Chengzhong Xu, and Kejiang Ye. 2024. Planck: Optimizing LLM Inference Performance in Pipeline Parallelism with Fine-Grained SLO Constraint. In *2024 IEEE International Conference on Web Services (ICWS)*. 1306–1313.
- [27] Kexin Liu, Zhaochen Zhang, Chang Liu, Yizhi Wang, Vamsi Addanki, Stefan Schmid, Qingyue Wang, Wei Chen, et al. 2025. Pyrrha: Congestion-Root-Based Flow Control to Eliminate Head-of-Line Blocking in Datacenter. In *22nd USENIX Symposium on Networked Systems Design and Implementation, NSDI 2025, Philadelphia, PA, USA, April 28–30, 2025*. 379–405.
- [28] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. 2024. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. In *Proceedings of the ACM SIGCOMM 2024 Conference (Sydney, NSW, Australia) (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 16–37.
- [29] Yuhao Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, et al. 2024. CacheGen: Fast Context Loading for Language Model Applications via KV Cache Streaming. In *SIGCOMM*.
- [30] Jie Lu, Jiaqi Gao, Fei Feng, Zhiqiang He, Menglei Zheng, Kun Liu, Jun He, Binbin Liao, et al. 2025. Alibaba Stellar: A New Generation RDMA Network for Cloud AI. In *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, S ao Francisco Convent, Coimbra, Portugal, September 8–11, 2025*. 453–466.
- [31] Federico De Marchi, Jialong Li, Ying Zhang, Wei Bai, and Yiting Xia. 2025. Unlocking Superior Performance in Reconfigurable Data Center Networks with Credit-Based Transport. In *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, S ao Francisco Convent, Coimbra, Portugal, September 8–11, 2025*. 842–860.
- [32] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. La Jolla CA USA, 1112–1127.
- [33] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville Ontario Canada, 1–15.
- [34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8–14, 2019, Vancouver, BC, Canada*. 8024–8035.
- [35] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, I nigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. 118–132.
- [36] Maksym Planeta, Jan Bierbaum, Leo Sahaya Daphne Antony, Torsten Hoefler, and Hermann H'artig. 2021. MigrOS: Transparent Live-Migration Support for Containerised RDMA Applications. In *Proceedings of the 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14–16, 2021*. 47–63.
- [37] Artem Y. Polyakov, Gal Shalom, Asaf Schwartz, Aviad Yehezkel, Omri Ben David, Omri Kahalon, Ariel Shahar, and Liran Liss. 2025. Device-Assisted Live Migration of RDMA Devices. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSP 2025, Lotte Hotel World, Seoul, Republic of Korea, October 13–16, 2025*. 153–168.

- [38] Ruoyu Qin, Zheming Li, Weiran He, Jiale Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, et al. 2025. 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 2025, Santa Clara, CA, February 25–27, 2025*. 155–170.
- [39] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2023. TACCL: Guiding Collective Algorithm Synthesis Using Communication Sketches.. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17–19, 2023*. 593–612.
- [40] Min Si, Pavan Balaji, Yongzhou Chen, Ching-Hsiang Chu, Adi Gangidi, Saif Hasan, Subodh Iyengar, Dan Johnson, et al. 2025. Collective Communication for 100k+ GPUs. arXiv:2510.20171 [cs]
- [41] Foteini Strati, Sara Mcallister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. 2024. DejaVu: KV-cache Streaming for Fast, Fault-tolerant Generative LLM Serving. arXiv:2403.01876 [cs]
- [42] Foteini Strati, Zhendong Zhang, George Manos, Ixeia S'anchez P'eriz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, et al. 2025. Sailor: Automating Distributed Training over Dynamic, Heterogeneous, and Geo-distributed Clusters. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles, SOSOP 2025, Lotte Hotel World, Seoul, Republic of Korea, October 13–16, 2025*. 204–220.
- [43] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. 2023. Bamboo: Making Pre-emptible Instances Resilient for Affordable Training of Large DNNs.. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17–19, 2023*. 497–513.
- [44] vLLM Team. 2026. Elastic DP Rank Recovery for Unrecoverable Process Failures. <https://github.com/vllm-project/vllm/issues/30112>. RFC Issue #30112, vLLM Project.
- [45] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. 2023. GEMINI: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Koblenz Germany, 364–381.
- [46] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchun Wan, Peng Xie, Tao Feng, et al. 2023. SRNIC: A Scalable Architecture for RDMA NICs.. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17–19, 2023*. 1–14.
- [47] Xingda Wei, Fangming Lu, Rong Chen, and Haibo Chen. 2022. KRCORE: A Microsecond-scale RDMA Control Plane for Elastic Computing.. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11–13, 2022*. 121–136.
- [48] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, et al. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters.. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4–6, 2022*. 945–960.
- [49] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Swati Gupta, and Tushar Krishna. 2024. TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning. In *57th IEEE/ACM International Symposium on Microarchitecture, MICRO 2024, Austin, TX, USA, November 2–6, 2024*. 856–870.
- [50] Yaxiong Xie, Fan Yi, and Kyle Jamieson. 2020. PBE-CC: Congestion Control via Endpoint-Centric, Physical-Layer Bandwidth Measurements. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. Virtual Event USA, 451–464.
- [51] Guanbin Xu, Zhihao Le, Yinhe Chen, Zhiqi Lin, Zewen Jin, Youshan Miao, and Cheng Li. 2025. AutoCCL: Automated Collective Communication Tuning for Accelerating Distributed and Parallel DNN Training. In *22nd USENIX Symposium on Networked Systems Design and Implementation, NSDI 2025, Philadelphia, PA, USA, April 28–30, 2025*. 667–683.
- [52] Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful Large Language Model Serving with Pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*. 144–158.
- [53] Ziming Zhao, Mingyu Wu, Jiawei Tang, Binyu Zang, Zhaoguo Wang, and Haibo Chen. 2023. BeeHive: Sub-second Elasticity for Web Services with Semi-FaaS Execution. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. Vancouver BC Canada, 74–87.
- [54] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10–12, 2024*. 193–210.
- [55] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, et al. 2025. An Extensible Software Transport Layer for GPU Networking. arXiv:2504.17307 [cs]
- [56] Siyuan Zhuang, Zhuohan Li, Danyang Zhuo, Stephanie Wang, Eric Liang, Robert Nishihara, Philipp Moritz, and Ion Stoica. 2021. Hoplite: Efficient and Fault-Tolerant Collective Communication for Task-Based Distributed Systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. Virtual Event USA, 641–656.