OptiNIC: A Resilient and Tail-Optimal RDMA NIC for Distributed ML Workloads

Ertza Warraich, Ali Imran[†], Annus Zulfiqar[†], Shay Vargaftik[*], Sonia Fahmy, Muhammad Shahbaz[†]
Purdue University [*]*Broadcom* [†]*University of Michigan*

Abstract

As distributed machine learning (ML) workloads scale to thousands of GPUs connected by high-speed interconnects, tail latency in collective communication has become a major bottleneck. Existing RDMA transports, such as RoCE, IRN, SRNIC, and Falcon, enforce strict reliability and in-order delivery, relying on retransmissions and packet sequencing to ensure correctness. While these approaches work well for general-purpose workloads, they introduce complexity and latency that scale poorly in ML, where even rare packet delays can stall entire model pipelines.

We present OptiNIC, a domain-specific RDMA transport that revisits traditional reliability guarantees based on ML's tolerance for partial or missing data. OptiNIC eliminates retransmissions and in-order delivery from the NIC, enabling a best-effort, out-of-order transport model for RDMA. Unlike traditional RDMA, which signals completion only after complete data delivery, OptiNIC introduces adaptive timeouts to trigger forward progress when data may be lost or delayed. OptiNIC retains standard congestion control mechanisms (e.g., DCQCN, EQDS, or Swift) while shifting loss recovery to the ML pipeline itself (e.g., via the Hadamard Transform and Erasure Coding).

Our evaluation shows that OptiNIC improves time-to-accuracy (TTA) by 2× and increases throughput by 1.6× for training and inference, respectively, across two public clouds (i.e., Hyperstack and CloudLab). OptiNIC also lowers 99th-percentile latency by 3.5×, cuts BRAM usage by 2.7×, and nearly doubles NIC resilience to faults—delivering a resilient, tail-optimized RDMA transport purpose-built for distributed ML workloads.

1 Introduction

As distributed machine learning (ML) workloads scale across thousands of GPUs connected by high-speed 100–400G fabrics, the performance bottleneck has shifted decisively from compute to communication. [31, 48] Collective operations (such as AllReduce, AllGather, and All-to-All) have become critical synchronization points in both data-parallel and model-parallel training and inference pipelines [11, 56, 59, 67]. These operations demand tight coordination among workers, where even minor tail delays in the communication fabric can stall overall progress [46]. As a result, *tail latency*, not average throughput, has emerged as the dominant barrier to scaling ML workloads efficiently across large clusters [56, 59].

To address this communication bottleneck, the community has introduced a range of optimizations. Systems like NCCL [26], RCCL [5], and MSCCL [33] apply both algorithmic [48, 49, 55] and hardware-aware [19, 30, 46] techniques to accelerate collectives. At the same time, compression methods such as gradient sparsification [15, 45, 58] and quantization [4] reduce bandwidth demands by exploiting the statistical nature of stochastic gradient descent (SGD) [4, 15]. These approaches are built on a key observation: *ML workloads are statistically robust*. They tolerate approximation, noise, and even bounded data loss without compromising final model accuracy [13, 16, 63].

Despite these advances, the underlying transport layer has largely remained general-purpose and overly conservative [35, 52, 57]. In modern ML clusters, RDMA is the dominant communication substrate, typically implemented via RoCE or its derivatives [18, 22]. These transports enforce strict reliability and in-order delivery, relying on retransmissions, packet sequencing, and lossless flow control mechanisms like Priority Flow Control (PFC) to ensure correctness. While appropriate for traditional distributed systems (like key-value stores or databases) [14, 40, 41, 60], these mechanisms are increasingly sub-optimal for ML. Their reliance on complete delivery as a precondition for forward progress introduces latency-critical paths that do not scale. A single packet loss can cascade into Go-Back-N retransmission storms or PFC-induced head-of-line blocking, stalling pipelines across the entire cluster [18, 57].

In response, several recent systems have begun rethinking RDMA transport. IRN [35] removes PFC by enabling in-NIC loss recovery through selective repeat, bitmap tracking, and SACK-based retransmissions. While this design improves cluster scalability, it inflates per-QP state and adds reordering complexity in the NIC. SRNIC [57] simplifies the NIC datapath by removing WQE caching and onloading retransmissions and reordering to host software, improving QP density and reducing NIC memory pressure. UCCL [67] pushes this idea further by onloading the entire transport control plane—including congestion control, flow scheduling, and multipath routing—into software, treating the NIC as a streamlined datapath.

In contrast, Falcon [52] takes the opposite approach: it embraces NIC complexity by integrating fast retransmissions, delay-based congestion control, and multipath routing directly into hardware. While Falcon performs well under loss, it increases the NIC state and vulnerability to hardware faults. At the same time, the Ultra Ethernet Consortium (UEC) proposes a clean-slate design for AI workloads [11, 25], introducing features like packet spraying, hybrid congestion control, and fast loss detection. However, like Falcon and IRN-based approaches, UEC's proposed transport preserves strict reliability semantics—requiring full delivery before forward progress.

The above-mentioned systems represent important steps forward, but they all retain a common assumption: that packet loss is rare and must be recovered before computation can continue [35, 52, 57, 67]. They preserve the long-standing semantic that *forward progress is gated on complete delivery*. At the ML scale, however, this assumption no longer holds. What seems like rare loss at a single node becomes frequent across thousands of workers synchronizing in parallel [56,59]. These losses accumulate at collective barriers, where even a single straggler can stall the entire operation. This is a classic case of "tail at scale" [12], worsened by transport-layer mechanisms that insist on full recovery before making progress.

In this paper, we ask: *If ML workloads can tolerate partial loss and reordering, why enforce strict delivery guarantees at the NIC at all?* If the application is already robust to bounded loss, why not remove the mechanisms that wait for full delivery entirely?

We present OptiNIC, a domain-specific RDMA transport that rethinks reliability and forward progress from the ground up. OptiNIC eliminates retransmissions and in-order delivery from the NIC, forwarding best-effort, out-of-order packets directly to application memory. Crucially, OptiNIC replaces delivery-based progress with a new primitive: *adaptive timeouts*. Rather than waiting for every packet to arrive, the receiver proceeds once a fixed time elapses—even if some data is missing. These timeouts are coordinated across peers and tuned to the collective's structure (e.g., Ring, Tree, BCube), providing consistent, time-bounded semantics for progress in lossy networks.

OptiNIC preserves compatibility with existing RDMA infrastructure. It retains standard congestion control mechanisms (such as DCQCN [68], EQDS [37], or Swift [28]) and maintains IB verbs semantics [8, 36], while keeping the RDMA programming model intact. Rather than recovering from packet loss within the transport, OptiNIC bounds its impact and shifts recovery to the ML stack, where lightweight redundancy mechanisms (such as the Hadamard Transform [59]) can reconstruct missing data efficiently. Timeout tuning, credit/window management, and error handling remain in software, preserving flexibility without adding NIC complexity.

This architectural shift significantly simplifies the NIC. OptiNIC eliminates reorder buffers, retransmission queues, and per-packet sequencing logic, cutting NIC BRAM usage by 2.7× and nearly doubling mean-time-between-failure (MTBF) by removing fault-prone state. It also prevents tail latencies caused by recovery delays at the cluster scale.

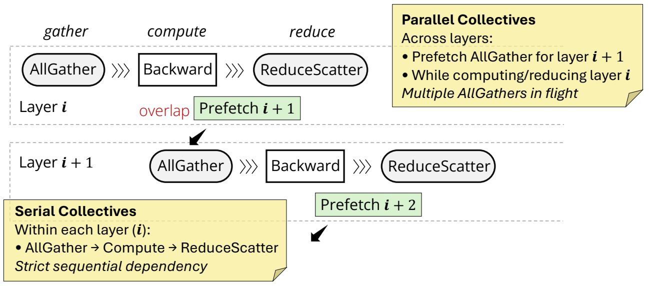


Figure 1: Overlapping intra-layer and inter-layer collective patterns during FSDP backward pass.

We further evaluate OptiNIC across a range of ML workloads and public cloud environments (§5). In clusters running on Hyperstack and CloudLab, OptiNIC delivers 1.8–2.5× speedups for collectives across message sizes and topologies. For end-to-end training, OptiNIC improves time-to-accuracy (TTA) by 2× using ZeRO-3 parallelism, boosts inference throughput by 1.6×, and reduces time-to-first-token (TTFT) tail latency—a key measure of LLM responsiveness—by 3.5×, all while preserving model accuracy.

In short, OptiNIC challenges the long-held assumption that reliable delivery is a necessary precondition for correctness in distributed ML systems. By replacing delivery-based progress with timeout-driven semantics tailored to ML, OptiNIC enables a new class of transport designs: simple, stateless, and resilient—co-designed for the unique demands of modern machine learning.

2 Background and Motivation

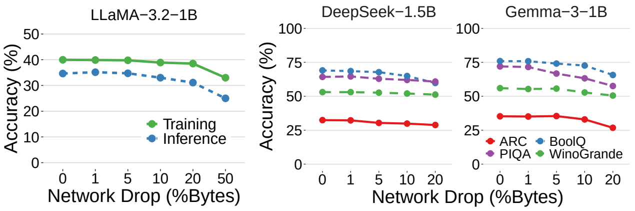
2.1 Communication Bottlenecks in ML Workloads

Distributed ML workloads rely on fine-grained, structured communication between GPUs to synchronize computation across a cluster. These communication patterns are dominated by collectives such as AllToAll (AA), AllReduce (AR), AllGather (AG), and ReduceScatter (RS), which are invoked on every iteration of training or every decoding step during inference. The specific collective topology and frequency are determined by the parallelism strategy employed—data, model, pipeline, tensor, or hybrid [29, 66].

In data parallelism, models are replicated across workers and gradients are synchronized using AR. More advanced variants, such as Fully Sharded Data Parallelism (FSDP) [66] or ZeRO-3 [43], reduce memory usage by partitioning the model state, but introduce additional collectives, AG and RS, within each training step. In model or pipeline parallelism, activations are passed between layer partitions across devices, resulting in fine-grained, latency-sensitive exchanges. Tensor parallelism introduces intra-layer collectives: for example, outputs of split matrix multiplications must be gathered across GPUs to proceed. Inference workloads further introduce complications with cache lookups, sequence-level slicing, and context merging, frequently relying on AA or AG collectives.

Feature	RoCE	IRN [35]	SRNIC [57]	Falcon [52]	UCCL [67]	OptiNIC
Transport Reliability	Go-Back-N (HW)	Selective Repeat (HW)	Selective Repeat (SW)	Selective Repeat (HW)	Selective Repeat (SW)	Best Effort
Packet Reordering	No/Dropped	Buffered in NIC	Software Reordering	Buffered in NIC	Software Reordering	Offset Based
Congestion Control	Hardware	Hardware	Hardware	Hardware	Software	Hardware
Priority Flow Control	Required	Not Required	Not Required	Not Required	Not Required	Not Required
Target Workloads	General RDMA	General RDMA	RDMA + ML	RDMA + ML + HPC	ML Collectives	ML Collectives
Key Focus	High performance	+Network efficiency	+Connection scalability	+Programmable CC	+Programmable transport	+Tail optimality

Table 1: Evolution of RDMA transport designs: from reliability-centric to tail-optimal. OptiNIC eliminates retransmission and reordering machinery, and uses best-effort, offset-based placement to support large-scale ML collectives.



(a) Training and inference on the ARC dataset.

(b) Inference-only accuracy using multiple datasets (ARC, BoolQ, and more).

Figure 2: Training and inference accuracy of all models remains stable under partial network drops ($\leq 5\%$).

Most often, workloads combine these strategies. Figure 1 illustrates the backward pass of an FSDP pipeline, where intra-layer collectives (AG → compute → RS) are chained together, and inter-layer prefetches run concurrently. This mixture of serial and parallel collectives creates intricate synchronization dependencies that define the critical path.

As models scale to hundreds of billions of parameters [31, 59] and clusters scale to thousands of GPUs [18], these collectives increasingly dominate end-to-end performance. Even a single delayed packet in one GPU's AG can stall the entire iteration. Studies show that collectives can account for 50–70% of total runtime in such systems [46,59]. While bandwidth requirements are well-understood, the bottleneck is not throughput but tail latency—specifically, the delay incurred by the slowest GPU in each synchronization round.

This effect is especially pronounced during inference. In multi-query batching or sequence-parallel pipelines, collectives are triggered at sub-millisecond granularity, with little opportunity to amortize communication delays. Time-to-first-token (TTFT) is directly impacted by per-step stalls, even if only a few packets are delayed. Moreover, the data exchanged in these collectives—intermediate tensors, activation fragments, KV cache blocks—is often small, redundant, or transient [17, 65]. Yet today's transports still enforce strict delivery semantics on every packet, waiting for full and in-order delivery before triggering progress. These semantics ignore ML's tolerance to loss (§2.2) and impose tail-latency penalties disproportionate to the impact of missing data.

2.2 ML Workloads are Resilient to Loss

The communication bottleneck in ML arises not only from message frequency but also from a mismatch between transport semantics and application needs. ML workloads are not fragile distributed systems. Rather, they are designed to be robust to approximation, randomness, and partial data [31,56].

Stochastic gradient descent (SGD) inherently absorbs noise across iterations [15]. Prior systems have leveraged this property to reduce bandwidth through quantization [4], gradient sparsification [15], or reduced-precision formats like bfloat16 [4, 15]. In-network aggregation frameworks such as SHARP perform approximate or lossy reductions directly in the dataplane, showing that full-precision delivery is not necessary for convergence [19].

This resilience extends beyond gradients. Activations, attention maps, and routing metadata are often recomputed or subsampled in subsequent steps. In MoE models [42, 50], missing expert outputs may be ignored or replaced via fallback paths. In self-attention [54], partial loss in key/value tensors may have a negligible impact due to the smoothing behavior of softmax layers. Figure 2 shows that across a variety of large-language models (LLMs) and datasets, both training and inference accuracy remain stable even at 5% drop rates.

Even when loss occurs, it is not uniformly damaging. Many tensors are padded, sparse, or partially redundant. From an application perspective, ML pipelines do not require every packet to arrive; rather, they require only enough data to complete the current step. This suggests a different progress model: one that favors bounded, timely delivery over strict reliability. While best-effort delivery may introduce nondeterminism, such variability is already present in large-scale training pipelines, and techniques such as per-step logging or structured redundancy can aid debugging and reproducibility.

2.3 The Cost of Reliable (RDMA) Transports

Despite loss tolerance of ML workloads, the transport layer remains conservative. RDMA transports like RoCE enforce strict reliability by default: Go-Back-N retransmissions, in-order delivery, and PFC to avoid loss [35, 57]. These mechanisms are implemented entirely in hardware. NICs maintain per-connection state, including retry counters, sequence numbers, window logic, and reorder buffers. They use acknowledgment packets to detect loss, timers to trigger recovery, and congestion windows to pace the sender. Even the widely-used NVIDIA NCCL stack uses RDMA's reliable RC queue pairs (QPs) with control/data QPs per peer, tightly coupling transport reliability with application progress [26].

Reliable semantics work well for key-value stores or RPCs, but they scale poorly for ML [56,59]. A single lost packet can block an entire collective. Worse, reliability mechanisms turn rare events into protocol-level delays. PFC-induced backpressure causes head-of-line blocking. Reordering logic inflates NIC memory usage. Retransmissions inject traffic bursts. In large ML jobs, where collectives synchronize thousands of workers, such events are no longer rare—they are expected. What appears as a one-in-a-thousand loss at a single node becomes one per step across the cluster.

Recent designs have tried to reduce this complexity. IRN replaces Go-Back-N with selective repeat, using bitmaps and selective ACKs to recover lost packets more efficiently [35]. SRNIC offloads reordering and retransmissions to software and eliminates the WQE cache, reducing NIC state [57]. UCCL moves the entire transport control plane into software, using the NIC purely as a datapath [67]. Falcon enhances the NIC instead, tightly integrating loss recovery, congestion control, and multipath routing for tail performance under stress [52].

These efforts vary in architecture but share a core assumption: that loss must be detected and corrected before progress. Table 1 compares these designs. All still enforce strict delivery semantics and treat forward progress as a function of full data arrival. This model is fundamentally misaligned with ML workloads, where approximate or delayed recovery is not only acceptable—it is preferable to waiting.

2.4 Reliability Hurts Fault Tolerance

Finally, enforcing reliability reduces the fault tolerance of the NIC. Transport-layer mechanisms—retry logic, sequence tracking, congestion windows—are stored in NIC SRAM, tightly coupled with datapath execution. These stateful elements are vulnerable to soft errors, transient failures, and silent corruption [7, 27]. At a cluster scale, even conservative mean-time-between-failure (MTBF) estimates lead to frequent failures. A stuck timer, corrupted sequence number, or missed completion can stall a QP indefinitely, halting collectives and triggering global backoff.

This fragility is unnecessary. ML workloads can make forward progress without complete delivery. Rather than hardening unreliable machinery inside the NIC, we eliminate it. OptiNIC discards retransmissions, in-order enforcement, and per-packet tracking altogether. Instead, it forwards best-effort packets directly to memory and uses a timeout mechanism to signal progress when no new packets arrive. Recovery, if needed, is handled in software via structured redundancy (e.g., Hadamard Transform [59]), §3.2.

Our architecture reduces per-QP state to just 20 bytes: no retry counters, timers, reorder buffers, or flow windows. Only minimal congestion control metadata remains. As we show in (§5.3), this design reduces NIC BRAM usage by 2.7×, nearly doubles MTBF in hardware fault models, and eliminates the tail stalls caused by reliability logic.

3 Design of OptiNIC

We present OptiNIC, a resilient, tail-optimal RDMA architecture for ML that eliminates retransmissions and in-order delivery at the NIC. Instead of tying progress to reliable delivery, OptiNIC introduces bounded completion semantics: each operation completes within an application-specified timeout, and the NIC signals partial completion to enable timely progress—even when some data is lost.

QP Type	Offloaded Packetization	Reliability	In-Order Delivery	CC
RC	✓	✓	✓	✓
UC	✓	✗	✓	✗
UD	✗	✗	✗	✗
OptiNIC: XP	✓	✗	✗	✓

Table 2: Comparison of RDMA QP types [36] and OptiNIC along key transport features. RC ensures reliability and ordering but incurs high tail latency; UC drops reliability but lacks congestion control (CC); UD offers no hardware support. OptiNIC's XP (eXpress Path) fills the gap: it drops reliability and ordering while retaining connection state, offloaded packetization, and CC.

OptiNIC occupies a new point in the RDMA transport design space (Table 2)—dropping reliability and ordering guarantees while retaining connection state, hardware packetization, and congestion control. These choices reflect its goal: to minimize tail latency and protocol overhead while preserving the RDMA programming model.

We begin by describing the transport architecture of OptiNIC (§3.1), covering its design for data delivery, bounded completion semantics, and congestion control. Next, we describe a software-level recovery mechanism (§3.2), which enables model correctness even when packets are lost or partially delivered. We then present two deployment mappings (§3.3), showing how OptiNIC can be realized with minimal changes to existing RDMA NICs: (1) by modifying SRNIC [57] to remove reliability machinery, and (2) using off-the-shelf RoCE NICs with the UC transport.

3.1 OptiNIC's Transport Architecture for RDMA

OptiNIC reimagines the standard RDMA transport abstractions—data delivery, completion, and congestion control—to support a lossy, time-bounded execution model tailored for ML workloads. While these services exist in all RDMA transports [36], their implementation in OptiNIC is explicitly designed to tolerate packet loss, avoid reordering overheads, and guarantee forward progress within application-defined time bounds. Each transport component is simplified to operate without retransmissions or ordering guarantees, while preserving the core RDMA programming model.

In OptiNIC, senders issue standard RDMA operations, which are fragmented into self-describing packets that can be delivered and placed independently of arrival order (§3.1.1). Completion is decoupled from reliable delivery: each work request entry (WQE) includes a timeout, and the NIC signals completion either when all fragments are received, or the timer expires (§3.1.2). This bounded approach enables timely progress reporting and downstream recovery. Congestion control remains intact: OPTINIC supports existing ECN-, delay-, or credit-based logics by separating rate regulation from reliability, allowing pacing to operate cleanly over a best-effort substrate (§3.1.3).

INFO: Key RDMA Concepts [36].

- RDMA transports use Work Queue Entries (WQEs) to describe operations such as SEND, RECV, or WRITE. Each WQE corresponds to a message and resides on a queue pair (QP). When an operation completes, the NIC posts a Completion Queue Entry (CQE) to notify the application.
- One-sided verbs like WRITE allow a sender to directly place data into the receiver's memory without coordination. Two-sided verbs like SEND/RECV require both sender and receiver to post matching WQEs.
- Packets within a message are usually sequenced using Packet Sequence Numbers (PSNs). In OPTINIC, we instead use a per-message sequence number called wqe_seq to identify operations and support timeout and preemption logic. Remote memory addresses are communicated via the RETH (RDMA Extended Transport Header) in one-sided operations.

3.1.1 Data Delivery Semantics.

The core function of RDMA transport is to move data from source to destination memory via direct memory access (DMA). For large messages, the NIC splits data into MTU-sized packets that must be placed correctly in the destination buffer. Traditional RDMA transports—such as Reliable Connected (RC) and Unreliable Connected (UC), Table 2—rely on strict in-order delivery for correct placement: only the first packet carries the full remote address and offset, while later packets infer their position using implicit packet sequencing. This ordering assumption forces reliable transports to either buffer out-of-order packets (e.g., with selective repeat) or retransmit them (e.g., with Go-Back-N), introducing latency, memory pressure, and tail amplification under loss.

In OPTINIC, we remove these dependencies by treating out-of-order arrival as the common case. We eliminate reordering logic entirely and instead ensure correct placement through self-describing packets, each of which carries sufficient metadata to be placed independently of other packets.

Self-Describing Packets. To place a packet correctly, the receiver must know two things: (1) which message the packet belongs to, and (2) where within the target buffer the payload should be written. Existing transports do not provide this information per packet. For one-sided verbs (like RDMA WRITE), only the first packet includes a RETH header with the virtual address, remote key (rkey), and total length; subsequent packets rely on in-order arrival to infer their offset. Two-sided verbs like SEND/RECV behave similarly: the receiver has the base address from its posted WQE, but packets carry no explicit offset—again relying on strict ordering.

This model breaks under loss or reordering. RC must buffer until gaps are filled; UC simply drops out-of-order packets. Neither can safely place packets that arrive out of order. OPTINIC addresses this by making every packet self-describing. Each fragment carries metadata needed for direct placement:

- For one-sided operations, each packet includes the full RETH header: the virtual address (with offset) and rkey.
- For two-sided SEND/RECV, each packet includes a byte offset into the pre-posted receive buffer.

This allows the receiver to perform in-place DMA on arrival—without buffering, reordering, or inferring offset from PSN. The NIC simply extracts the offset from the packet header and writes the payload to memory. This design supports correct placement regardless of arrival order and applies uniformly to one-sided and two-sided operations.

Out-of-Order Delivery Across Messages. Traditional RDMA semantics also require that messages be delivered in order: packets from a new message cannot be processed until the previous message completes. This works under reliable delivery, where missing fragments are eventually retransmitted. In OPTINIC, however, packets may be lost permanently. Waiting for missing fragments would stall the QP indefinitely.

To allow forward progress without buffering, OPTINIC introduces a single-active-message model. Each packet carries a wqe_seq identifier to indicate which message it belongs to. The receiver maintains a single expected wqe_seq and processes packets as follows:

- If the packet's wqe_seq matches the expected value, it is part of the active message and is placed immediately.
- If the packet's wqe_seq is greater, the sender has moved on. The receiver finalizes the previous message and begins processing the new one.
- If the wqe_seq is less, the packet belongs to a completed (or timed out) message and is dropped.

This keeps receiver state bounded: only one active message is tracked per QP, and no per-message buffering is required. The arrival of a new message acts as an implicit timeout for the previous one, allowing the receiver to progress earlier.

Late Packet Handling. Once a message is completed—either by receiving its final fragment or by timeout—the NIC advances the expected wqe_seq, clears associated state, and posts a CQE. Any packets that arrive afterward with the old sequence number are immediately dropped. This ensures correctness even in the presence of delayed fragments or multipath reordering: late packets cannot corrupt application memory or confuse the completion logic.

3.1.2 (Bounded) Completion Semantics.

Traditional RDMA transports define completion based on reliable delivery: an operation completes when all its fragments are received and acknowledged. This model assumes eventual delivery and ties progress to in-order arrival and retransmissions. In OPTINIC, however, these assumptions no longer hold. There are no retransmissions, and packets may be lost permanently. To ensure forward progress without reliability, OPTINIC introduces a new model: *bounded completion semantics*, where each operation completes within a timeout and reports partial progress if necessary.

OPTINIC preserves the notion of completion familiar to RDMA developers. On the sender side, a WQE is marked complete once all fragments have been transmitted—no acknowledgments are required. On the receiver, normal completion occurs when the NIC observes the last fragment of a message (marked explicitly). Even if earlier packets were lost, receiving the final one signals message completion and triggers a CQE.

When the final fragment never arrives, OPTINIC uses an application-specified timeout to avoid indefinite stalls. Each WQE includes a timeout value that bounds how long it can remain active. If this deadline expires before complete data arrival, the NIC finalizes the WQE and generates a CQE indicating partial progress.

To track this, the NIC maintains a per-WQE byte counter that accumulates the payload size of successfully placed packets. This logic reuses existing DMA metadata and adds only minimal state. Upon timeout, the NIC reports this count to the application, allowing the upper layer (e.g., the collective engine) to proceed with partial data.

Timeouts are managed using per-WQE hardware timers, similar to those already implemented for retry or RNR timeout logic in reliable transports [35,57]. These timers are reused but reinterpreted: instead of triggering retransmissions, they now bound execution time.

Early Completion via Preemption. OPTINIC introduces a form of early timeout via preemption. If the receiver observes a packet from a newer message (with a higher wqe_seq), it immediately finalizes the current message and begins processing the new one. This mechanism ensures timely progress and bounds per-WQE state, even when packets are delayed or reordered. Any subsequent packets from the older message are dropped, ensuring correctness.

Adaptive Timeout Estimation. Choosing a fixed timeout is challenging in distributed ML workloads, where network conditions and collective patterns vary widely [17,65]. To address this, OPTINIC includes an adaptive timeout mechanism that adjusts values over time.

After each collective operation, nodes record two key statistics: the elapsed time and the number of bytes successfully received, including both full and partial completions. These values are exchanged asynchronously across the collective group and used to compute an empirical per-byte transfer cost (e.g., microseconds per kilobyte). Each node then proposes a timeout value for future iterations, derived by multiplying this cost by the message size.

Before the next invocation of the same collective on the same group, nodes aggregate the proposed values to form a group-wide timeout. They compute the median across all peers to reduce the impact of outliers (e.g., nodes experiencing transient loss or congestion). To further avoid oscillation, especially in small collectives, the group applies an exponentially weighted moving average (EWMA) to smooth the update: $T_{\text{new}} = \alpha \cdot T_{\text{median}} + (1-\alpha) \cdot T_{\text{old}}$. We use $\alpha = 0.2$, which balances responsiveness with stability. The resulting value becomes the canonical timeout estimate for future operations of the same collective and group.

If no historical observations are available—such as on the first invocation—OPTINIC initializes the timeout using the measured duration of a warmup collective executed during the bootstrap phase. Specifically, it sets: $T_{\text{initial}} = (1+\gamma) \cdot T_{\text{warmup}} + \delta$, where γ is a multiplicative safety margin (we use 0.25) and δ is a small additive slack (50µs) to absorb short-term variance. This conservative baseline ensures that early iterations proceed reliably while timeout estimation converges.

Timeouts are applied at the granularity of individual RDMA operations. For collective algorithms with multiple phases, the total timeout budget is divided across phases: parallel steps share the same deadline, while sequential steps are assigned proportional slices, ensuring that the entire operation completes within the allotted time.

Finally, small control-plane messages—like handshakes and phase markers—are typically under one MTU and do not impact tail latency or bandwidth. OPTINIC routes them over the pre-existing reliable channel, avoiding unnecessary timeout logic and keeping the data path focused on large transfers.

Timeout Behavior Across Verbs. Timeout behavior follows the standard RDMA model: it applies only to the side that posts a WQE.

- SEND/RECV (two-sided verbs): Both sender and receiver post WQEs and each side attaches its own timeout.
- WRITE (one-sided verb): Only the sender posts a WQE and sets a timeout. The receiver performs DMA but does not track time.
- WRITE_WITH_IMM: Behaves like a hybrid; both sides post WQEs, and timeouts are active on both ends.
- READ: The requester attaches a timeout. To avoid unnecessary transmissions, OPTINIC piggybacks this deadline in the request, allowing the responder to stop sending after the deadline.

3.1.3 Congestion Control Semantics. In traditional RDMA transports, congestion control (CC) is tightly coupled with reliability: packet loss is treated as a congestion signal and triggers retransmission. OPTINIC eliminates retransmissions entirely, decoupling these two mechanisms. In this model, dropped packets no longer imply congestion, and feedback packets from the receiver (e.g., ACKs or CNPs) are interpreted purely as best-effort congestion signals, not as proofs of reliable delivery.

Despite this shift, OptiNIC remains fully compatible with the dominant RDMA CC schemes deployed in practice. ECN-based algorithms like DCQCN [68] rely on explicit switch marks and CNPs generated for packets that do arrive; their control loops operate unchanged. Delay-based controllers such as TIMELY [34] and Swift [28] compute RTT from timestamped feedback packets, which OptiNIC continues to generate for received packets; lost packets yield no feedback. Likewise, telemetry- and credit-based schemes such as HPCC [32] and EQDS [37] depend on in-band telemetry or explicit credit messages—none of which require reliable delivery of every data packet.

3.2 Lightweight Data Recovery & Loss Mitigation

Since OptiNIC operates without retransmissions or in-order delivery, some data loss is expected. Rather than masking this loss via heavyweight transport-layer repair, OptiNIC leverages the inherent robustness of ML workloads and introduces a lightweight software mechanism to mitigate its impact. The key goal is to prevent localized packet drops from introducing correlated corruption in model state or gradient tensors.

Loss Amplification From Spatial Clustering. In a naive design, each packet carries a contiguous slice of a tensor. If this packet is lost, its entire span of values is zeroed during placement. This spatially clustered loss disproportionately affects model quality: adjacent values often correspond to neighboring neurons or channels, and the resulting distortion can destabilize training or degrade inference accuracy. Prior work has shown that structured randomness—such as that introduced by the Hadamard Transform [39]—can spread such errors across a tensor and preserve convergence even under significant loss [31, 56, 59].

INFO: Hadamard Transform [39]. It is an orthogonal linear mixing operation that disperses each input element across all output coefficients. This spreads local errors uniformly and preserves tensor norms, making it effective for mitigating sparse loss.

(a) Block-Wise Encoding for Compute Efficiency. To reduce computational overhead, OptiNIC applies the Hadamard Transform in a block-wise manner. Each tensor is logically divided into B blocks of p elements (typically matching the per-packet MTU size), and each block is transformed independently. Because the transform is linear, encoded tensors can be aggregated or reduced without decoding—a useful property for collectives (like AllReduce). Block-wise encoding significantly lowers GPU compute cost (§5.3) but is not sufficient by itself: if a packet carries an entire encoded block and is lost, all p coefficients from that block are erased, nullifying the transform's resilience benefit.

(b) Stride-based Packet Interleaving to Improve Recovery. To prevent this failure mode, OptiNIC introduces a stride-based layout that interleaves encoded blocks across packets. After transforming each block, packets are constructed by selecting a fraction of coefficients from multiple blocks, rather than all coefficients from a single block. Specifically, a stride parameter S determines how many blocks contribute to a packet: each packet carries p/S elements from each of S blocks, for a total of p elements. This interleaving spreads the impact of a lost packet across many blocks, reducing the distortion experienced by any single one.

This layout is efficiently implemented via Scatter–Gather Entries (SGEs), an RDMA feature [36] that allows non-contiguous memory regions to be transmitted in a single message. When striding is enabled, each packet header includes S and a per-packet offset, allowing the receiver to perform correct placement without coordination or ordering guarantees—extending the self-describing packet abstraction (§3.1.1).

With maximal striding ($S = p$), each packet contains one coefficient from each of p blocks. Losing a packet in this regime zeroes one element per block, converting clustered loss into sparse noise. The inverse Hadamard Transform uniformly distributes this residual error across each affected block. The stride parameter S thus allows OptiNIC to trade off dispersion strength against complexity: higher S improves data recovery, but mixes more block elements per packet. We evaluate the benefits of this design in §5.3, showing that it preserves accuracy even under nontrivial packet loss.

3.3 Realizing OptiNIC on Existing RDMA NICs

Although OptiNIC defines new transport semantics, many of its mechanisms align with capabilities already present in modern RDMA NICs [36]. Realizing OptiNIC therefore requires only modest deltas: removing features that are no longer needed, reusing existing abstractions for new purposes, and introducing small software or header-level extensions. We describe how OptiNIC maps onto two representative platforms—SRNIC and commodity RoCE NICs—while highlighting where these NICs fall short and how OptiNIC bridges those gaps.

SRNIC: Removing Reliability, Reusing Metadata and Timers. SRNIC [57] already contains several building blocks that OptiNIC relies on—per-packet metadata for direct placement (remote address, offset, sequence ID), and an in-place DMA pipeline that bypasses reassembly. These features allow OptiNIC's self-describing delivery semantics to be supported without structural changes to the datapath. Likewise, SRNIC's per-WQE timers, originally intended for retransmissions and RNR backoff, are repurposed to enforce OptiNIC's bounded completion model with byte tracking.

However, SRNIC's original design assumes reliable delivery. Implementing OptiNIC requires removing the full reliability subsystem—including bitmap tracking, outstanding-request tables, and loss-recovery state machines—which no longer serve a purpose and impede tail performance. OptiNIC thus simplifies the NIC by eliminating these components entirely. The only new NIC-visible field is a 2-byte stride parameter to support recovery (§3.2), while congestion control continues to use the existing control queue (CtrlQ) to surface ECN and pacing signals to software. The result is a smaller, faster, and more resilient datapath that preserves compatibility with the RDMA programming model.

RoCE w/ UC: Software Approximation of* OPTINIC *on Fixed-Function Hardware. Commodity ConnectX-class RoCE NICs offer no datapath programmability, and their UC transport enforces in-order delivery for multi-packet messages—a semantics incompatible with OPTINIC's out-of-order, best-effort model. Realizing OPTINIC therefore requires a software approximation that works within these hardware constraints.

The key approximation is forcing every fragment to be a single-packet WRITE. Each MTU-sized block is issued as an independent WRITE_WITH_IMM carrying explicit placement metadata, ensuring that the NIC never triggers its built-in ordering or reassembly logic. This preserves OPTINIC's semantics despite UC's fixed in-order behavior.

Completion and timeout semantics are reimplemented entirely in software, using immediate values to identify fragments and timer queues to enforce deadlines. To prevent corruption after timeout—something commodity NICs cannot guard against—we rely on Memory Windows (MWs) with per-operation rkeys, allowing receivers to revoke write access mid-collective and block late WRITEs. Congestion control is likewise implemented in software. Fragment-level feedback packets guide pacing decisions in lieu of NIC-managed rate control [67]; lost fragments yield no feedback, which integrates cleanly with OPTINIC's best-effort model.

Although this realization incurs modest CPU overhead, it preserves all core OPTINIC semantics—independent placement, bounded completion, and explicit pacing—without requiring firmware or driver changes. This makes OPTINIC immediately deployable on existing RoCE networks while highlighting the limitations of today's NICs and the architectural simplifications OPTINIC enables.

4 Implementation

We implement OPTINIC on two platforms: a software prototype running over commodity RoCE NICs to evaluate end-to-end training and inference, and a hardware prototype on an FPGA-based SmartNIC to assess area, state overhead, and transport-level scalability.

RoCE Software Prototype. We implement OPTINIC as a new transport backend in NVIDIA's NCCL (v2.23.4–1) via the Net plugin interface [26], enabling out-of-the-box compatibility with DeepSpeed (v0.18.2), PyTorch, and vLLM (v0.9.1). The integration adds fewer than 500 lines to NCCL, with the transport logic written in about 8K lines of C++.

All control-plane logic—including completion tracking, timeout management, and congestion control—is implemented in software. A dedicated timer thread manages deadlines, while congestion control uses EQDS [37], with the sender pacing transmissions based on per-fragment ACKs.

Hadamard transforms for error recovery are implemented on GPU using an optimized CUDA kernel from HazyResearch [24], applied block-wise during encoding/decoding.

FPGA Hardware Prototype. The hardware implementation of OPTINIC is built on the AMD Alveo U250 FPGA using Coyote-v2, an open-source RoCEv2-compatible SmartNIC shell [44]. We synthesize the design in Vivado 2022.1 and target 10K QPs to match common transport scalability needs.

We evaluate OPTINIC's hardware resource usage by removing Coyote's built-in reliability mechanisms—retransmission logic, outstanding-request tables, bitmaps, and reorder buffers—and adding minimal per-WQE state for timeouts and byte tracking. The transport pipeline reuses Coyote's existing support for self-describing placement, and we extend the packet header by 2 bytes to support stride placement for recovery.

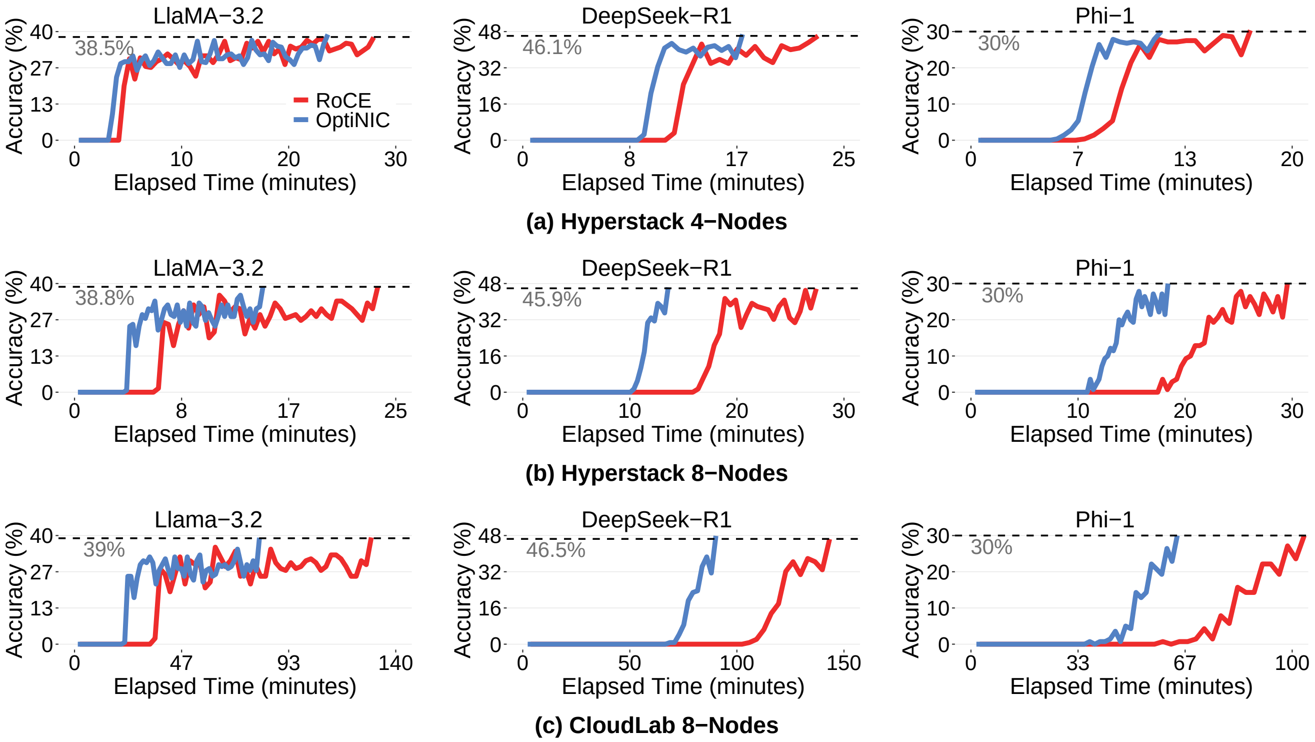
For comparison, we also synthesize three baselines: (a) IRN/Falcon, which uses a 1.2 MB reorder buffer and reconstructed QP state based on prior work. (b) SRNIC, using QP metadata and extensions as described in the original paper. (c) UCCL, which requires no hardware changes and runs atop base RoCE. This setup allows us to directly compare datapath area, path delay, and QP state across all designs in §5.3.

5 Evaluating OPTINIC

We evaluate OPTINIC to validate our central claim: simplifying RDMA transport for ML workloads improves tail latency, reduces hardware overhead, and enhances system resilience. Our evaluation covers three dimensions—latency, efficiency, and fault tolerance—using both microbenchmarks and end-to-end distributed workloads on a real-world Cloud cluster and an FPGA-based prototype.

5.1 Experimental Setup

5.1.1 Test Environments. Our experiments are conducted on three environments from an academic cloud, CloudLab [1], and a commercial cloud vendor, Hyperstack [3]. On CloudLab, we provision an 8-node r7525 [2] cluster, with each machine featuring dual AMD EPYC 7542 CPUs (64 cores, 2.9 GHz), 512 GB DDR4 ECC memory, with an NVIDIA Tesla V100S GPU (32 GB). Networking is provided by dual-port Mellanox ConnectX-5 NICs (PCIe Gen4), connected via a 25 Gbps Ethernet fabric. All nodes run Ubuntu 22.04 and NCCL v2.23.4–1 with default configurations and OPTINIC with NCCL on the SmartNICs (§4). To emulate realistic multi-tenant conditions, we introduce controlled background traffic that reflects RDMA network behavior reported in prior works [22, 35]. On Hyperstack, we provision 4 and 8 node H100-80G-PCIe clusters with each machine featuring 28 CPU cores, 180 GB DRAM, and an Nvidia H100 GPU with 80 GB HBM3 memory and PCIe Gen5 interconnect.



(a) Hyperstack 4−Nodes

(b) Hyperstack 8−Nodes

(c) CloudLab 8−Nodes

Figure 3: End-to-end convergence time-to-accuracy of RoCE and OPTINIC for different models and cluster environments.

5.1.2 Baselines, Workloads, and Cluster Setup. We evaluate three recent open-source LLMs—Llama-3.2-1B [20], Phi-1-1B [21], and DeepSeek-R1-Distill-Qwen-1.5B [23]—chosen to span different architectures and model families. For training, we fine-tune each model on the ARC-Challenge dataset [10] using DeepSpeed with ZeRO-3 parallelism, keeping all other training hyperparameters fixed across transports. For inference, we again use ARC-Challenge prompts and measure end-to-end generation throughput (tokens/sec), latency (TTFT), and accuracy using vLLM with all three models served using Tensor + Pipeline parallelism. For inference, we also serve the Qwen-3-30B MoE model [64] using Tensor+Expert parallelism to exercise MoE-specific communication patterns.

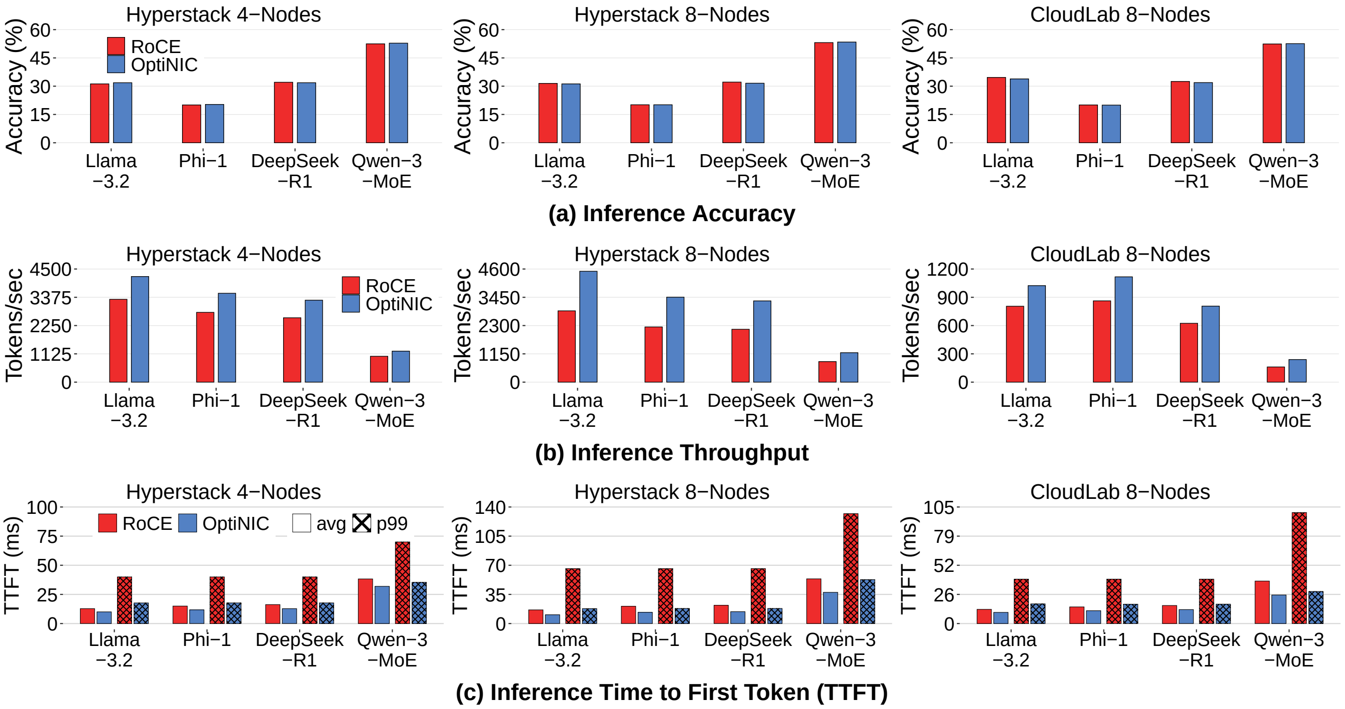
Our primary baseline is RoCEv2 with RC QPs, which enforces retransmissions, in-order delivery, and strict completion semantics at the NIC level—representative of production datacenter RDMA deployments. We also benchmark OPTINIC's performance against IRN [35], SRNIC [57], Falcon [52], UEC [11], and UCCL [67]. Open-source implementations of these are unavailable or incompatible with cloud environments and are therefore excluded from end-to-end performance runs.[1] To assess the resilience of these implementations at scale, we evaluate their soft-error susceptibility using the Xilinx SEU Estimator v2023.1 [6]. Following datacenter deployment guidelines, we model a 15,000-node cluster operating at a junction temperature of 100 °C [38, 47], enabling a comparative analysis of reliability across all transport designs under realistic large-scale conditions.

5.2 End-to-End Performance

5.2.1 Distributed Training. Figure 3 reports convergence trajectories for Llama-3.2-1B, Phi-1-1B, and DeepSeek-R1-Distill-Qwen-1.5B when fine-tuned on ARC-Challenge with ZeRO-3 parallelism. Across all three cloud environments and models, OPTINIC reduces TTA by 1.6× on average relative to RoCE. Larger configurations benefit more: the 8-node setups yield up to 2× improvement, particularly on Hyperstack where H100 GPUs shift the bottleneck toward communication. While CloudLab shows larger raw communication gains, its V100 GPUs limit end-to-end speedups. These gains arise because RoCE's Go-Back-N loss recovery briefly halts progress for all nodes, even when only a single packet is dropped. In contrast, OPTINIC continues making forward progress within each bounded window, avoiding these global pauses. Final accuracy is unchanged across models, and in some cases even slightly improves. For example, DeepSeek-R1 achieves around a 1.2% higher final accuracy with OPTINIC, as the small, bounded perturbations introduced by random and infrequent packet drops act as a mild form of regularization—similar in spirit to noise injection or dropout—and can occasionally enhance generalization.

5.2.2 Distributed Inference. Figure 4 shows inference accuracy, throughput (tokens/sec), and latency (TTFT) across all evaluated models and cloud environments. Interestingly,

[1] We attempted to include UCCL [67] in our end-to-end evaluation, but encountered a bug; the issue has been confirmed by the developers and is currently being fixed.



(a) Inference Accuracy

(b) Inference Throughput

(c) Inference Time to First Token (TTFT)

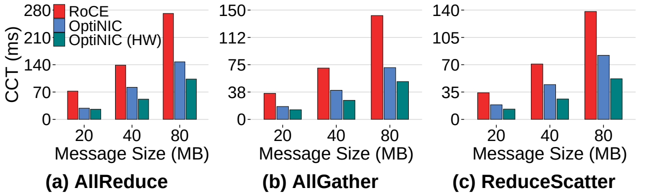
Figure 4: Inference accuracy, throughput, and time-to-first-token (TTFT) across models and cluster environments.

the Qwen-3-30B MoE model also shows a small accuracy increase with OptiNIC. In MoE inference, small activation-level perturbations can change which experts are selected, occasionally producing outputs that score slightly higher. As vLLM serving is far less communication-intensive than ZeRO-3 training, the gains of OptiNIC are correspondingly more moderate but consistently significant. Figure 4a shows that OptiNIC inference accuracy remains effectively unchanged across the board (differences $< 0.2\%$) against tail-inducing reliable RoCE transport across all models and environments. In Figure 4b, across the non-MoE models, OptiNIC improves throughput by roughly 28–60% over RoCE. Finally, average TTFT improves slightly, whereas tail (p99) latency drops sharply across all models (2–3.5×) as shown in Figure 4c, consistent with OptiNIC's tail-optimal design. Notably, the largest gains in both throughput and TTFT tail appear on the Hyperstack 8-node configuration, where the stronger H100 GPUs make communication the dominant bottleneck and OptiNIC's benefits manifest most prominently.

5.3 Microbenchmarks

5.3.1 OptiNIC is up to 2.5× faster than RoCE across all collectives and message sizes.

Figure 5 compares OptiNIC and RoCE across tensor sizes from 20–80 MB for AllReduce, AllGather, and ReduceScatter collectives. RoCE's latency grows steeply with size, reflecting the cumulative cost of retransmissions and completion dependencies. In contrast, OptiNIC scales smoothly: latency increases only moderately, remaining mostly linear, and consistently delivering a 1.6–2.5× speedup over RoCE across the tested sizes and collectives. To emulate a hardware deployment of OptiNIC (HW), we subtract software overheads (segmentation, timers, pacing) from the RoCE-based prototype, isolating the transport's performance contribution. Despite omitting retransmissions, observed loss stays under 1% on average for the tensors. Under a more aggressive timeout for large tensors—accepting 4–5% loss—OptiNIC achieves up to 5× lower latency than RoCE (not shown), highlighting the performance headroom unlocked when applications tolerate slightly higher loss.



(a) AllReduce (b) AllGather (c) ReduceScatter

Figure 5: Collective communication time: comparison across transports, message sizes, and collective types. (Std-dev, RoCE:±35, OptiNIC:±12, and OptiNIC (HW):±3.

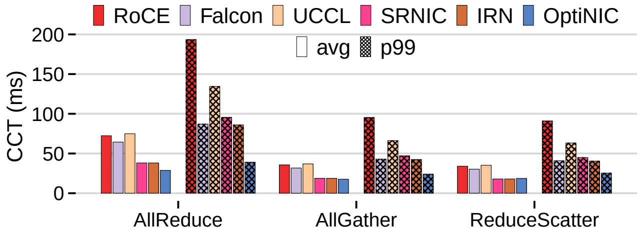
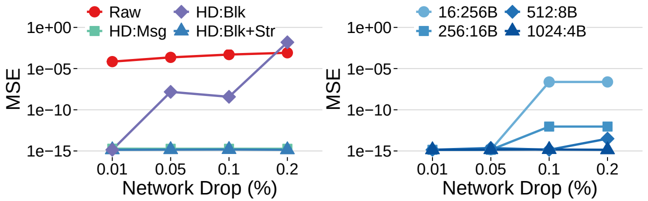


Figure 6: Collective completion time: average and tail comparison for different transports.

5.3.2 OptiNIC delivers the fastest collective completion times across all transports.

As shown in Figure 6, OptiNIC



(a) Encoding Schemes (b) Blocks:Stride

Figure 7: Comparison of MSE with Hadamard on (a) different configurations and (b) stride parameters.

#Splits →	1	4	16	64
Time (ms)	22.1 ± 0.08	19.9 ± 0.04	17.5 ± 0.07	8.4 ± 0.13

Table 3: Mean ± std runtime of Hadamard across different split counts for a 128 MB message.

achieves the lowest collective communication times across all three collectives—AllReduce, AllGather, and ReduceScatter—delivering both the smallest average Collective Completion Time (CCT) and the lowest tail latency (p99) among all transports. In contrast, RoCE, Falcon, and UCCL exhibit similar mean performance, but their tail latencies remain significantly higher: Falcon and UCCL match RoCE on average yet their tail rises to levels comparable to IRN and SRNIC, highlighting persistent head-of-line blocking and retry overheads. IRN and SRNIC modestly reduce mean CCT but still suffer from large p99 spikes, particularly for AllReduce, where tail latencies exceed 100–150 ms. By eliminating retransmissions and reordering entirely, OPTINIC avoids these tail-amplifying effects and consistently delivers both fast and tightly bounded completion times.

5.3.3 Hadamard with stride delivers robust, efficient loss dispersion.

Table 3 shows that a Hadamard transform on raw 128 MB message is the most expensive configuration, while splitting the tensor into 64 blocks reduces runtime by 2.5×, motivating block-level processing. Figure 7a examines the resulting resilience tradeoffs: Raw (no coding) and full-message Hadamard (HD:Msg) behave as expected, with the latter achieving near-ideal MSE at the highest cost. Block-wise Hadamard (HD:Blk) is far cheaper but can catastrophically amplify error because a lost packet removes all encoded coefficients for a block, making recovery impossible. Adding striding (HD:Blk+Str) disperses coefficients across packets so each loss removes only one position per block, producing MSE comparable to full-message Hadamard at much lower overhead. Figure 7b also shows that resilience improves with increasing stride: small strides couple many coefficients per packet and increase MSE, whereas maximal dispersion yields near–ideal reconstruction across all drop rates. Overall, OPTINIC's HD:Blk+Str design matches the robustness of full-message transforms at a fraction of the computational cost, making striding essential for resilient block-wise encoding.

5.3.4 OPTINIC achieves order-of-magnitude higher QP scalability with minimal QP state.

OPTINIC delivers the highest scalability among all evaluated transports, as shown in Table 4, by reducing per-QP NIC state to just 52 B—an order of magnitude smaller than RoCE (407 B), IRN (596 B), SRNIC (242 B), Falcon (350 B), and UCCL (407 B). This dramatic reduction enables OPTINIC to support up to 80K active QPs within the same SRAM budget (4 MB) that limits existing designs to between 8K and 20K QPs. As a result, OPTINIC scales collective training to 40K nodes, far exceeding the limits of RoCE (5K), IRN (4K), and Falcon (6K), and doubling SRNIC's 10K-node scale. UCCL scales even more poorly, as it opens 256 connections per peer—compared to the default 2 for all other schemes—which quickly exhausts NIC resources at large cluster sizes.

Metric	RoCE	IRN	SRNIC	Falcon	UCCL	OPTINIC
NIC State per QP	407 B	596 B	242 B	350 B	407 B	**52 B**
Max. QPs	10K	8K	20K	12K	10K	**80K**
Cluster Size	5K	4K	10K	6K	256	**40K**

Table 4: Comparison of transport protocols across NIC state, QP count, and cluster scalability.

Metric	RoCE	IRN	SRNIC	Falcon	UCCL	OPTINIC
LUT	312.4K	319.6K	304.5K	309.8K	312.4K	**298.4K**
LUTRAM	23.3K	24.2K	22.5K	23.1K	23.3K	**21.7K**
FF	562.1K	573.1K	551.5K	559.2K	562.1K	**543.0K**
BRAM	1.5K	2.2K	0.9K	1.6K	1.5K	**0.5K**
Power (W)	34.7	35.9	33.5	34.3	34.7	**32.5**
MTBF (hrs)	42.8	30.9	57.8	40.5	42.8	**80.5**

Table 5: Comparison of hardware resource utilization and resilience (MTBF) across RDMA NIC architectures.

5.3.5 OPTINIC delivers maximum hardware resilience with the smallest NIC footprint.

Table 5 shows that OPTINIC achieves the smallest hardware footprint and highest resilience among all evaluated NIC transports. Compared to RoCE, IRN, SRNIC, Falcon, and UCCL under a 10K-QP configuration synthesized for AMD Alveo U250, OPTINIC reduces LUT usage to 298.4K, LUTRAMs to 21.7K, and FFs to 543.0K—representing up to 6.6%, 10.2%, and 5.2% savings, respectively. Most notably, OPTINIC cuts BRAM consumption to just 0.5K (a 63–73% reduction versus RoCE and IRN) and lowers power draw to 32.5 W. By eliminating retransmission, reordering, and per-QP window state, OPTINIC maintains only 52 B of per-QP context and thus achieves the highest MTBF of 80.5 hours—nearly 2× better than RoCE and IRN—demonstrating that simplifying transport-layer hardware simultaneously improves efficiency and robustness.

6 Discussion and Future Work

Deployment on DPUs and SmartNICs. While our RoCE prototype executes its software control plane on the host CPU, nothing in OPTINIC's design requires host participation. The timeout manager, per-fragment ACK processing, and software congestion controller all operate on simple, event-driven logic that can run unmodified on modern DPUs and SmartNICs, which already expose programmable ARM clusters or P4/XDP execution units. Offloading these components moves

OPTINIC's control path closer to the NIC datapath, reducing host overhead and enabling tighter pacing loops without altering any transport semantics.

Reproducibility and Sources of Nondeterminism. A known limitation of best-effort transport for distributed training and inference is reduced reproducibility: transient packet losses may lead to nondeterministic training (or inference) dynamics across runs. However, this trade-off is increasingly acceptable in large-scale LLM workloads, where nondeterminism is already prevalent due to numerical instabilities, asynchronous kernel execution, and dynamic parallelism. Even models configured for determinism (e.g., fixed random seeds and zero sampling) often exhibit nontrivial variance in convergence accuracy and output behavior [9]. In practice, this means that strict reproducibility is rarely achieved end-to-end and system-induced variance is one of many contributing factors. Nevertheless, OPTINIC can optionally log missing offsets or byte ranges per step, enabling post hoc debugging or reproduction of loss patterns when needed.

Beyond Distributed Training: Broader Applicability. While our work focuses on distributed training and inference, the same principles can benefit a broader class of applications that value timeliness over strict reliability. Latency-critical and soft real-time systems such as online recommendation services, interactive analytics, and real-time media streaming (e.g., video conferencing) often tolerate minor data loss or approximation in exchange for bounded response times. Applying bounded-loss transport semantics to these domains opens an exciting direction for rethinking communication not as a strictly reliable service, but as a controllable dimension in system design, where performance, efficiency, and accuracy can be balanced according to application goals. OPTINIC demonstrates that reliability is not a universal requirement, but a workload-dependent choice. Relaxing the traditional transport guarantees can yield tangible benefits in tail latency and scalability. We believe this opens up a broader space of domain-specific transport designs that trade reliability for performance, tuned to the needs of the workloads.

7 Related Work

RDMA Transports and NIC Architecture. IRN [35] removes PFC by introducing selective-repeat reliability in the NIC, using bitmap tracking and SACK-based retransmissions to tolerate loss in RoCE clusters. SRNIC [57] simplifies the NIC datapath by eliminating WQE caching and shifting retransmissions and reordering to host software, improving QP density and reducing NIC memory pressure. UCCL [67] continues this trend by offloading transport control—congestion control, flow scheduling, and multipath routing—into software, treating the NIC as a streamlined datapath. Conversely, Falcon [52] embraces NIC complexity with fast retransmissions, delay-based congestion control, and hardware multipath routing, while UEC [11,25] proposes a clean-slate transport for AI workloads with packet spraying and hybrid congestion control. Despite their differences, these designs preserve strict reliability semantics and recover every lost packet. In contrast, OPTINIC removes packet-level recovery entirely and introduces bounded-loss completion semantics in software, yielding a best-effort RDMA transport for ML workloads.

Collective Communication and ML Systems. NCCL [26], MSCCL [33], and UCC [55] accelerate collective operations through topology-aware scheduling, fused kernels, and unified CPU/GPU/DPU interfaces. SHArP [19] performs hierarchical in-network reductions, while SwitchML [46] and OmniReduce [15] offload aggregation into programmable switches or network dataplanes. Recent systems such as MSCCL++ [49], NCCLX [51], and MCCS [62] further improve collective scheduling and overlap on large GPU clusters. These approaches optimize or offload collective algorithms but fundamentally assume a reliable, in-order transport. OPTINIC differs by rethinking the transport itself: it provides a best-effort RDMA system that supports all collective patterns without specialized switches or full packet reliability.

Lossy Transports and Approximate Communication for ML. Approximate communication techniques like Top-k sparsification [53] and quantization methods such as QSGD [4] and TernGrad [61] reduce gradient traffic with error compensation. THC [31] enables homomorphic aggregation over quantized updates, while MLT [56] explores bounded-loss behavior tuned for ML workloads. OptiReduce [59] mitigates tail effects in AllReduce via time-bounded execution with adaptive recovery. These approaches focus on software-level techniques and primarily target AllReduce. In contrast, OPTINIC provides hardware-level loss tolerance in the RDMA datapath, generalizes to all collectives and parallelisms, and supports both training and inference while saturating modern 100–400,Gbps links.

8 Conclusion

As ML systems scale to thousands of GPUs and flows, the assumption that transports must guarantee perfect delivery becomes increasingly misaligned with workload needs. Beyond performance, large NIC state and retransmission logic also reduce resilience and increase fault risk. OPTINIC shows that correctness in distributed ML does not require full reliability, but rather timely, bounded progress. By removing retransmissions and in-order delivery, and introducing adaptive, timeout-driven completion, OPTINIC simplifies the NIC while allowing collectives to proceed without waiting for stragglers. This design yields 1.8–2.5× faster collectives, 2.7× lower BRAM usage, and nearly 2× higher fault resilience. These benefits extend to end-to-end workloads: OPTINIC halves ZeRO-3 time-to-accuracy, improves inference throughput by 1.6×, and reduces tail TTFT by 3.5×, all without affecting accuracy. By prioritizing time-bounded progress over strict reliability, OPTINIC enables a faster, more scalable transport layer for ML.

References

[1] CloudLab. https://www.cloudlab.us, last accessed: 12/10/2025.

[2] CloudLab Hardware. https://docs.cloudlab.us/hardware.html, last accessed: 12/10/2025.

[3] Hyperstack. https://www.hyperstack.cloud, last accessed: 12/10/2025.

[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. *NeurIPS*, 2017.

[5] AMD. ROCm Communication Collectives Library. https://rocmdocs.amd.com/projects/rccl/en/latest/. Last accessed: 12/10/2025.

[6] AMD. AMD's Soft Error Mitigation. https://docs.amd.com/r/en-US/pg187-ultrascale-sem/Understanding-the-Soft-Error-Mitigation-Requirement, last accessed: 12/10/2025.

[7] AMD. SEU and Soft Error Rate Measurements. https://docs.amd.com/r/en-US/ug116/SEU-and-Soft-Error-Rate-Measurements, last accessed: 12/10/2025.

[8] Infiniband Trade Association. InfiniBand Architecture Specification Volume 1. https://www.afs.enea.it/asantoro/V1r1_2_1.Release_12062007.pdf. Last accessed: 12/10/2025.

[9] Berk Atil, Sarp Aykent, Alexa Chittams, Lisheng Fu, Rebecca J Passonneau, Evan Radcliffe, Guru Rajan Rajagopal, Adam Sloan, Tomasz Tudrej, Ferhan Ture, Zhe Wu, Lixinyu Xu, and Breck Baldwin. Non-determinism of "Deterministic" LLM Settings. *arXiv:2408.04667*, 2024.

[10] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457*, 2018.

[11] Ultra Ethernet Consortium. Ultra Ethernet Specification v1.0.1. https://ultraethernet.org/wp-content/uploads/sites/20/2025/10/UE-Specification-1.0.1.pdf. Last accessed: 12/10/2025.

[12] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *CACM*, 2013.

[13] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *NeurIPS*, 2022.

[14] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast Remote Memory. In *USENIX NSDI*, 2014.

[15] Jiawei Fei, Chen-Yu Ho, Atal N Sahu, Marco Canini, and Amedeo Sapio. Efficient Sparse Collective Communication and its Application to Accelerate Distributed Deep Learning. In *ACM SIGCOMM*, 2021.

[16] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers. *ICLR*, 2022.

[17] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *USENIX OSDI*, 2024.

[18] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. RDMA over Ethernet for Distributed Training at Meta Scale. In *ACM SIGCOMM*, 2024.

[19] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim, and Eitan Zahavi. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *IEEE International Workshop on Communication Optimizations in HPC (COMHPC)*, 2016.

[20] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models. *arXiv:2407.21783*, 2024.

[21] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks Are All You Need. *arXiv:2306.11644*, 2023.

[22] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. RDMA over Commodity Ethernet at Scale. In *ACM SIGCOMM*, 2016.

[23] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv:2501.12948*, 2025.

[24] HazyResearch. HazyResearch's Open Source Hadamard CUDA Implementation. https://github.com/HazyResearch/structured-nets. Last accessed: 12/10/2025.

[25] Torsten Hoefler, Karen Schramm, Eric Spada, Keith Underwood, Cedell Alexander, Bob Alverson, Paul Bottorff, Adrian Caulfield, Mark Handley, Cathy Huang, et al. Ultra Ethernet's Design Principles and Architectural Innovations. *arXiv:2508.08906*, 2025.

[26] Sylvain Jeaugey. NCCL 2.0. In *GPU Technology Conference (GTC)*, 2017.

[27] Andrew M. Keller and Michael J. Wirthlin. The Impact of Terrestrial Radiation on FPGAs in Data Centers. *ACM TRETS*, 2021.

[28] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM*, 2020.

[29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings*

of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

[30] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. ATP: In-network Aggregation for Multi-tenant Learning. In *USENIX NSDI*, 2021.

[31] Minghao Li, Ran Ben Basat, Shay Vargaftik, ChonLam Lao, Kevin Xu, Michael Mitzenmacher, and Minlan Yu. THC: Accelerating Distributed Deep Learning Using Tensor Homomorphic Compression. In *USENIX NSDI*, 2024.

[32] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*, 2019.

[33] Microsoft. Microsoft Collective-Communication Library. https://github.com/microsoft/msccl. Last accessed: 12/10/2025.

[34] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-based Congestion Control for the Datacenter. *ACM SIGCOMM CCR*, 2015.

[35] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. Revisiting Network Support for RDMA. In *ACM SIGCOMM*, 2018.

[36] NVIDIA. RDMA Aware Networks Programming User Manual. https://docs.nvidia.com/rdma-aware-networks-programming-user-manual-1-7.pdf. Last accessed: 12/10/2025.

[37] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, Adrian Popa, Cristi Baciu, Mark Silberstein, Georgios Nikolaidis, Mark Handley, and Costin Raiciu. An Edge-queued Datagram Service for All Datacenter Traffic. In *USENIX NSDI*, 2022.

[38] Oracle. Datacenter and Server Thermal Trends and Challenges. https://www.meptec.org/Resources/6%20-%20Oracle.pdf, last accessed: 12/10/2025.

[39] William K Pratt, Julius Kane, and Harry C Andrews. Hadamard Transform Image Coding. *Proceedings of the IEEE*, 1969.

[40] Yilun Qiao, Xiangyao Lu, Hang Zhang, Xiangyu Dong, and Jian Zhan. HERMIT: Low-Latency, High-Throughput, and Transparent Remote Memory. In *USENIX NSDI*, 2023.

[41] Deepti Raghavan, Shreya Ravi, Gina Yuan, Pratiksha Thaker, Sanjari Srivastava, Micah Murray, Pedro Henrique Penna, Amy Ousterhout, Philip Levis, Matei Zaharia, et al. Cornflakes: Zero-Copy Serialization for Microsecond-Scale Networking. In *ACM SOSP*, 2023.

[42] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing MoE Inference and Training to Power Next-Generation AI Scale. In *PMLR*, 2022.

[43] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *IEEE SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[44] Benjamin Ramhorst, Dario Korolija, Maximilian Jakob Heer, Jonas Dann, Luhao Liu, and Gustavo Alonso. Coyote v2: Raising the Level of Abstraction for Data Center FPGAs. *arXiv:2504.21538*, 2025.

[45] Cèdric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. Sparcml: High-performance Sparse Communication for Machine Learning. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[46] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtárik. Scaling Distributed Machine Learning with In-Network Aggregation. In *USENIX NSDI*, 2021.

[47] Scott Schweitzer, CISSP. Power, Heat, Space, and the Move to Double-Wide SmartNICs. https://www.linkedin.com/pulse/power-heat-space-move-double-wide-smartnics-scott-schweitzer-cissp/, last accessed: 12/10/2025.

[48] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *USENIX NSDI*, 2023.

[49] Aashaka Shah, Abhinav Jangda, Binyang Li, Caio Rocha, Changho Hwang, Jithin Jose, Madan Musuvathi, Olli Saarikivi, Peng Cheng, Qinghua Zhou, et al. MSCCL++: Rethinking GPU Communication Abstractions for Cutting-edge AI Applications. *arXiv:2504.09014*, 2025.

[50] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv:1701.06538*, 2017.

[51] Min Si, Pavan Balaji, Yongzhou Chen, Ching-Hsiang Chu, Adi Gangidi, Saif Hasan, Subodh Iyengar, Dan Johnson, Bingzhe Liu, Regina Ren, et al. Collective Communication for 100k+ GPUs. *arXiv preprint arXiv:2510.20171*, 2025.

[52] Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan MG Wassel, Naveen Kr Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, et al. Falcon: A Reliable, Low Latency Hardware Transport. In *ACM SIGCOMM*, 2025.

[53] Sebastian U Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with Memory. *NeurIPS*, 2018.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *NeurIPS*, 2017.

[55] Manjunath Gorentla Venkata, Valentine Petrov, Sergey Lebedev, Devendar Bureddy, Ferrol Aderholdt, Joshua Ladd, Gil Bloch, Mike Dubman, and Gilad Shainer. Unified Collective Communication: A Unified Library for CPU, GPU, and DPU Collectives. In *IEEE HOTI*, 2024.

[56] Hao Wang, Han Tian, Jingrong Chen, Xinchen Wan, Jiacheng Xia, Gaoxiong Zeng, Wei Bai, Junchen Jiang, Yong Wang, and Kai Chen. Towards Domain-Specific Network Transport for Distributed DNN Training. In *USENIX NSDI*, 2024.

[57] Zilong Wang, Layong Luo, Qingsong Ning, Chaoliang Zeng, Wenxue Li, Xinchen Wan, Peng Xie, Tao Feng, Ke Cheng, Xiongfei Geng, Tianhao Wang, Weicheng Ling, Kejia Huo, Pingbo An, Kui Ji, Shideng Zhang, Bin Xu, Ruiqing Feng, Tao Ding, Kai Chen, and Chuanxiong Guo. SRNIC: A Scalable Architecture for RDMA NICs. In *USENIX NSDI*, 2023.

[58] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient Sparsification for Communication-efficient Distributed Optimization. *arXiv:1710.09854*, 2017.

[59] Ertza Warraich, Omer Shabtai, Khalid Manaa, Shay Vargaftik, Yonatan Piasetzky, Matty Kadosh, Lalith Suresh, and Muhammad Shahbaz. OptiReduce: Resilient and Tail-Optimal AllReduce for Distributed Deep Learning in the Cloud. In *USENIX NSDI*, 2025.

[60] Xingda Wei, Rong Chen, and Haibo Chen. Fast RDMA-Based Ordered Key-Value Store Using Remote Learned Cache. In *USENIX OSDI*, 2020.

[61] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning. *NeurIPS*, 2017.

[62] Yongji Wu, Yechen Xu, Jingrong Chen, Zhaodong Wang, Ying Zhang, Matthew Lentz, and Danyang Zhuo. MCCS: A Service-based Approach to Collective Communication for Multi-Tenant Cloud. In *ACM SIGCOMM*, 2024.

[63] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *ICML*, 2023.

[64] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 Technical Report. *arXiv:2505.09388*, 2025.

[65] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference. In *ACM ASPLOS*, 2022.

[66] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *arXiv:2304.11277*, 2023.

[67] Yang Zhou, Zhongjie Chen, Ziming Mao, ChonLam Lao, Shuo Yang, Pravein Govindan Kannan, Jiaqi Gao, Yilong Zhao, Yongji Wu, Kaichao You, Fengyuan Ren, Zhiying Xu, Costin Raiciu, and Ion Stoica. An Extensible Software Transport Layer for GPU Networking. *arXiv:2504.17307*, 2025.

[68] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion Control for Large-Scale RDMA Deployments. In *ACM SIGCOMM*, 2015.