LZIP-P2P: LOSSLESS COMPRESSION SUPERCHARGED GPU COMMUNICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The rapid growth of large language models (LLMs) has made GPU communication a critical bottleneck. While prior work reduces communication volume via quantization or lossy compression, these approaches introduce numerical errors that can degrade convergence, accuracy, and stability. We present LZIP-P2P, a design that integrates lossless compression directly into GPU point-to-point (P2P) communication primitives without compromising numerical correctness. LZIP-P2P employs a split-send pipeline that exposes transmissible data early and overlaps compression with communication, while preserving high GPU efficiency by operating on large data blocks. In real workloads, LZIP-P2P accelerates reinforcement learning (RL) weight synchronization by up to 47.5% without application changes.

1 INTRODUCTION

The rapid growth of model size and context length in large language models (LLMs) (Chowdhery et al., 2023; Grattafiori et al., 2024; DeepSeek-AI et al., 2025; Qwen et al., 2025) has made multi-node, multi-GPU execution indispensable for both training and inference. In such settings, large volumes of data, including model parameters, activations, key-value (KV) caches, and intermediate states, are exchanged across GPUs within and across nodes. Point-to-point (P2P) communication provides the flexibility required by irregular and dynamic communication patterns and is widely used in scenarios such as KV cache transfer (UCCL Project, 2025; NVIDIA, 2025a; Licker et al., 2025; Mooncake Project, 2024) and asynchronous weight updates in reinforcement learning (RL) workloads. As a result, P2P communication performance has become a critical bottleneck, directly impacting the efficiency and scalability of LLM training and inference.

To reduce communication overhead, prior work has extensively explored quantization and other lossy compression techniques to reduce data volume (Alistarh et al., 2017; Wang et al., 2023; Li et al., 2024a; Huang et al., 2024a; Xu et al., 2021; Sun et al., 2025). While effective in improving bandwidth utilization, these approaches inevitably introduce numerical errors. These errors can slow convergence during training or degrade model accuracy at inference time, and may also cause training–inference mismatch in RL (He et al., 2025). More recently, lossless compression techniques on GPUs have begun to emerge (Meta AI Research, 2022; Hershcovitch et al., 2025; Hao et al., 2024; Zhang et al., 2025; NVIDIA, 2023a; Azami et al., 2025), demonstrating the potential to reduce data size without sacrificing numerical fidelity. This observation opens up a promising opportunity: applying lossless compression to GPU communication. However, despite its potential, this direction remains largely underexplored for P2P communication, particularly in the context of tightly integrated communication pipelines.

Effectively hiding the overhead of lossless compression within the communication pipeline remains a key challenge. On modern GPUs, compression latency does not scale linearly with input size due to their highly parallel execution model, where throughput is maximized only when sufficient parallelism is exposed to fully utilize hardware resources (NVIDIA, 2023b). As a result, compressing smaller chunks often incurs similar latency to larger ones, undermining conventional fine-grained pipelining strategies that partition data into small blocks and overlap compression with transmission. In such designs, excessive fragmentation can even amplify overheads, leading to degraded end-to-end performance.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Tensor Type	Size (MB)	Data Type	Compression Ratio
Gradient	214	FP32	0.848
Activation	16	BF16	0.679
Weight	107	BF16	0.675

Table 1: Compression ratios of representative tensors collected during GLM4-9B training (GLM et al., 2024). We define the compression ratio as the ratio between the compressed size and the original (uncompressed) size; thus, lower values indicate better compression.

We present LZIP-P2P, a design that integrates lossless compression directly into GPU P2P communication primitives, thereby addressing the fundamental tension between compression overhead and communication efficiency. LZIP-P2P enables bandwidth reduction without modifying user-facing APIs or compromising numerical correctness. Our key observation is that the first stage of a GPU floating-point compressor accounts for only $\sim$14% of the total compression time, yet already finalizes roughly half of a `bfloat16` payload. Exploiting this asymmetry, LZIP-P2P employs a compression-aware *split-send* pipeline that aligns with the structure of floating-point compression (Section 3): it processes data in sufficiently large chunks to maintain high GPU utilization, and decouples compression stages to expose data that can be transmitted early. This allows a significant portion of the payload to be sent over the network before full compression is complete, effectively overlapping communication with compression.

We evaluate LZIP-P2P across a diverse set of GPU clusters, including multi-node Remote Direct Memory Access (RDMA) environments with Elastic Fabric Adapter (EFA) and RDMA over Converged Ethernet (RoCEv2) (Section 5). Our approach consistently improves effective bandwidth, closely approaching the upper bound implied by the achievable compression ratio. In reinforcement learning training, LZIP-P2P improves weight synchronization throughput by up to 47.5% on large tensors, with consistent gains across both dense and mixture-of-experts (MoE) models and stable compression ratios across training steps.

In summary, this paper makes the following contributions:

- A *split-send* communication pipeline that aligns pipeline boundaries with the internal stages of floating-point compression rather than with data size, hiding compression latency inside the transfer while preserving large-block GPU efficiency.
- An open-sourced implementation of LZIP-P2P, integrated into a production P2P communication library and portable across AMD and NVIDIA GPUs and across NVIDIA CX7, Broadcom Thor-2, and AWS EFA NICs, which extends DietGPU (Meta AI Research, 2022) to five floating-point formats including `float8_e4m3fn` and `float8_e5m2`. Across multiple GPU clusters, it accelerates RL weight synchronization by up to 47.5% while preserving bit-exact results.

2 Background and Motivation

2.1 Basic GPU Lossless Compression Design

2.1.1 Statistical Structure of Floating-Point Values

Many existing works (Meta AI Research, 2022; Zhang et al., 2025; Hao et al., 2024; Hershcovitch et al., 2025) exploit the statistical properties of floating-point tensors to enable efficient lossless compression. Floating-point values consist of three fields: sign, exponent, and fraction (mantissa). Among these components, the fraction and sign bits typically exhibit near-uniform distributions and thus provide limited compression opportunities. In contrast, the exponent field often follows a narrow and highly skewed distribution in many machine learning workloads. This skew becomes more pronounced when tensors are normalized to a bounded range. Consider the `bfloat16` format, which contains 1 sign bit, 8 exponent bits, and 7 fraction bits. When a `bfloat16` tensor is normalized, the exponent field frequently collapses to a small number of repeated patterns. Such highly skewed symbol distributions are well suited for entropy coding techniques such as Asymmetric Numeral Systems (ANS) (NVIDIA, 2023a; Meta AI Research, 2022; Azami et al., 2025), enabling high compression ratios. Table 1 reports compression ratios of representative tensors

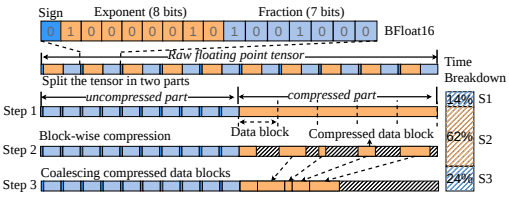
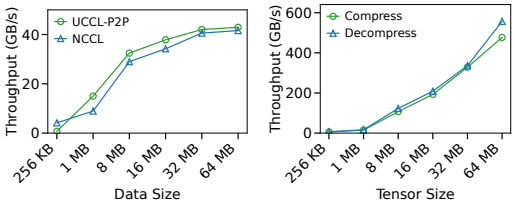


Figure 1: Overview and time breakdown of a typical GPU-based floating-point ANS compression pipeline (S1–S3 denote Steps 1–3).



(a) P2P. (b) Compression.

Figure 2: P2P communication and compression throughput on two AWS p5en.48xlarge nodes connected via EFA.

from GLM4-9B (GLM et al., 2024) training. The ratio remains stable across tensors of the same type: `float32` gradients achieve 0.85, while `bfloat16` tensors consistently achieve around 0.68 across layers and parameter types.

2.1.2 GPU Compression Algorithm for Floating-Point Tensors

Prior work, such as DietGPU (Meta AI Research, 2022), leverages GPUs to accelerate compression of floating-point tensors. Exploiting the statistical structure described above, such approaches decompose each value into two components: the exponent field and the remaining bits (sign and fraction), and compress only the exponent field. This design improves compression efficiency compared to directly compressing raw floating-point representations. Figure 1 illustrates the overall structure of a typical GPU-based ANS compression pipeline, which consists of three steps, each performing a single pass over global memory.

Step 1: Float Splitting and Frequency Table Construction. During the first pass, each value is decomposed into its exponent and remaining bits (sign and fraction) (Figure 1), which are written to two contiguous buffers in global memory: the *compressed part* (exponents) and the *uncompressed part*. This lightweight step accounts for only approximately 14% of the overall compression time.

Step 2: Independent Block-wise Compression. Next, multiple GPU thread blocks perform compression independently (Figure 1). Each thread block reads a chunk of the exponent buffer from global memory, applies ANS encoding locally, and writes the result to a temporary global-memory buffer. Because entropy coding produces variable-length outputs, this step generates non-contiguous compressed data chunks. This is the most time-consuming step, accounting for approximately 62% of the overall compression time.

Step 3: Coalescing Compressed Blocks. In the final step, the variable-length outputs produced by individual thread blocks are coalesced into a single contiguous output buffer (Figure 1). This step accounts for approximately 24% of the overall compression time.

2.2 Challenges of Applying Lossless Compression to GPU Communication

Compression latency must not offset communication savings. The latency of compression can outweigh the savings from reduced data transfer. As we show, conventional approaches to hiding compression latency often yield negligible gains or even degrade performance.

A naive design compresses all data prior to transmission, assuming reduced data volume will improve throughput. Consider a representative example in RL training pipelines, where model weights are transferred between training and rollout nodes via P2P send/recv. A typical tensor size is 16 MB (e.g., `bfloat16` tensors of shape $[4096, 2048]$ in MoE models such as Qwen3.5-35B-A3B (Yang et al., 2025)). As shown in Figure 2b, compressing a 16 MB tensor incurs $\sim$90 μs of latency, while at the P2P throughput of Figure 2a the reduced data volume saves only $\sim$108 μs of transfer time. The net saving is therefore $\sim$18 μs, a marginal throughput improvement of $\sim$4%.

A natural mitigation is to partition tensors into chunks and pipeline compression with communication. However, this strategy is ineffective for GPU-based compression. Splitting a 16 MB tensor into four 4 MB chunks does not alleviate the bottleneck. Each 4 MB chunk still incurs $\sim$70 μs of compression latency, while the per-chunk communication savings shrink to only $\sim$20 μs, resulting in a

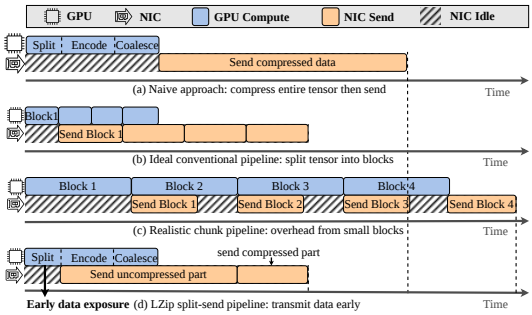


Figure 3: Comparison of pipelining designs. (b)(c) Chunk-based pipelining assumes latency scales with size but is ineffective for GPU compression. (d) Split-send exposes transmissible data early and overlaps communication with remaining compression.

net performance degradation. This asymmetry stems from the massive parallelism of modern GPUs: compression latency does not scale linearly with data size, as GPU parallelism is underutilized for small inputs.

3 DESIGN

LZIP-P2P aims to address the above challenges when applying lossless compression to GPU communication. LZIP-P2P introduces a *split-send* pipeline that overlaps compression with network transmission. Unlike naive pipelining schemes, LZIP-P2P aligns pipeline stages with the internal steps of the compression algorithm. This stage-aligned design operates on large data blocks rather than fine-grained chunks, preserving GPU efficiency while exposing data that can be transmitted early in the compression process. Consequently, network transmission begins earlier and overlaps with subsequent compression steps, enabling faster bandwidth ramp-up and improved network utilization. We first explain why conventional chunk-based pipelining is ineffective for GPU compression (Section 3.1), then identify the opportunity that *split-send* exploits (Section 3.2), and finally present the pipeline design (Section 3.3).

3.1 WHY CHUNK-BASED PIPELINING FAILS

Conventional pipelining splits tensors into independent chunks to reduce the latency of producing the first transmittable byte and to overlap computation with communication. As shown in Figure 3 (b), this approach assumes that compression latency decreases proportionally with data size.

However, this assumption does not hold for GPU compression. Compression involves multiple GPU kernels and memory transformations whose overheads do not scale with input size. Due to these overheads and the massive parallelism of GPUs, the latency of GPU compression increases much more slowly than the input size. Consequently, compressing small chunks may take nearly as long as compressing substantially larger ones. For example, on an NVIDIA H200 GPU, compressing a 16 MB floating-point tensor takes approximately 90 μs, while compressing a 4 MB tensor takes approximately 70 μs. Although the data size is reduced to 1/4, the latency decreases modestly by only 22%. As shown in Figure 3 (c), naive chunking therefore degrades performance in compression–communication pipelines: it significantly reduces overall GPU throughput while barely shortening the time to the first transmittable byte.

3.2 OPPORTUNITY: EARLY AVAILABILITY OF TRANSMISSIBLE DATA

Although chunking cannot shorten compression latency, the compression pipeline itself finalizes part of its output early. The first step of the compression pipeline accounts for 14% of the total compression time but already exposes a large portion of the final representation (Figure 1). This stage performs a structural split of the input tensor into two components: an *uncompressed part* and a *compressed part*. The ratio between the two portions depends on the floating-point format. For example, for `bfloat16` tensors, the uncompressed and compressed portions each account for approximately half of the tensor. For `float32`, the uncompressed portion occupies roughly

three quarters of the data. Once the split stage completes, the uncompressed portion requires no further computation and can therefore be transmitted immediately. Since the compressed portion will be further reduced by the subsequent compression stages (Step 2 and Step 3 in Figure 1), which dominate the overall compression cost, the uncompressed portion constitutes an even larger fraction of the final transmitted data.

3.3 Split-Send Pipeline Design

Based on these observations, we design a *split-send communication pipeline* that overlaps compression with communication while preserving large-block GPU efficiency. Rather than partitioning the tensor by size, LZip-P2P partitions it along the boundary that the first compression stage already produces: the **uncompressed portion** is finalized and can be transmitted immediately, while the **compressed portion** continues through the subsequent compression stages before transmission. As shown in Figure 3 (d), while the GPU processes the remaining compression stages, the network interface concurrently transmits the uncompressed portion. Once compression finishes, the smaller compressed payload is transmitted. This pipeline exposes data early and overlaps compression with network communication without sacrificing GPU throughput.

4 Implementation

We build LZip-P2P by integrating lossless compression into UCCL-P2P (Zhou et al., 2025; UCCL Project, 2025), a high-performance communication library that provides easy-to-use P2P APIs and achieves performance comparable to or exceeding NCCL/RCCL (Jeaugey, 2017; AMD, 2024), NVIDIA NIXL (NVIDIA, 2025a), and Mooncake TE (Mooncake Project, 2024). With approximately 4 K lines of modular C++ code, LZip-P2P incorporates transparent compression into synchronous and asynchronous send, receive, and write operations. It supports both NVIDIA and AMD GPU platforms while preserving the original user-facing interface, requiring no application changes.

We redesign UCCL-P2P's metadata and receiver-side processing to seamlessly integrate compression with one-sided RDMA writes. Because compression changes the transmitted size, LZip-P2P extends UCCL-P2P's fixed-size metadata with the data type and the sizes before and after compression, allowing the sender to issue `write_with_imm` to the correct remote address and the receiver to reconstruct the data. Since compressed data written directly to receiver memory cannot be consumed by the application, LZip-P2P introduces a dedicated receiver-side thread that detects data arrival, decompresses it, and places the result in the user's destination buffer. This design preserves UCCL-P2P's transparent interface while keeping decompression off the RDMA write path.

We extend DietGPU (Meta AI Research, 2022) to support heterogeneous hardware and additional floating-point formats. Combined with UCCL-P2P, LZip-P2P supports NVIDIA and AMD GPUs with diverse RDMA-capable NICs, including EFA and InfiniBand. It supports `float16`, `float32`, `bfloat16`, `float8_e4m3fn`, and `float8_e5m2`. For the `float8` formats, LZip-P2P packs two values into a 16-bit unit and jointly extracts their exponents into an 8-bit stream, enabling byte-granular writes while avoiding misalignment overhead.

5 Evaluation

5.1 Experiment Setup

Testbeds. We evaluate LZip-P2P across diverse GPU–NIC platforms, including two `p5en.48xlarge` instances connected via Elastic Fabric Adapter (EFA), a two-node AMD cluster with AMD Instinct MI355X GPUs connected via RoCEv2. By default, LZip-P2P is evaluated on the `p5en.48xlarge` cluster. Hardware specifications are summarized in Table 2.

Baselines. We use UCCL-P2P (Zhou et al., 2025; UCCL Project, 2025) with its default synchronous `send`/`recv` interface as the baseline, as LZip-P2P is built directly on top of it. We compare LZip-P2P against two designs: (1) a naive *encode-send* scheme that transmits only after full compression, and (2) a conventional *naive pipeline* that partitions data into 8 MB chunks and overlaps compression with communication. We adopt an 8 MB chunk size, as it empirically provides a reasonable

Cluster	AWS EFA	AMD
GPU	8×H200	8×MI355X
CPU	Xeon 8488C	EPYC 9575F
Cores	192	128
NICs	16×EFA	8×DSC SmartNICs
Bandwidth	200 Gbps	400 Gbps

Table 2: Hardware specifications of our experimental platforms.

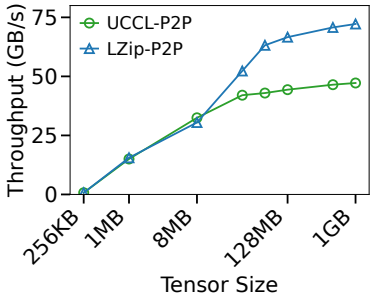


Figure 4: LZIP-P2P throughput for `bfloat16` point-to-point communication across tensor sizes.

balance between compression effectiveness and pipeline overhead. Compression is enabled only for messages larger than 1 MB, as overhead outweighs benefits for smaller messages. We use `bfloat16` as the default data format.

We do not use compression-enabled communication systems as baselines. Libraries such as gZCCL (Huang et al., 2024a), ghZCCL (Huang et al., 2025a), ZCCL (Huang et al., 2025b), C-Coll (Huang et al., 2024b), and compressed MPI libraries (Zhou et al., 2021), as well as lossy gradient-compression frameworks, rely on lossy compression and trade numerical fidelity for bandwidth, whereas LZIP-P2P preserves bit-exact results. Their reported throughput is therefore not directly comparable to ours.

5.2 END-TO-END THROUGHPUT

Figure 4 reports the throughput of our LZIP-P2P compression pipeline across tensor sizes from 256 KB to 1 GB. As tensor size increases, LZIP-P2P delivers progressively higher performance gains over UCCL-P2P.

For medium-to-large tensors (>8 MB), LZIP-P2P consistently outperforms UCCL-P2P. At 1 GB, it improves throughput by up to 52.9% (72.2 GB/s vs. 47.2 GB/s), closely approaching the theoretical upper bound of 73.8 GB/s derived from Amdahl's Law under a compression ratio of 0.64. For smaller tensors, the benefits are more modest (e.g., 8% at 16 MB and 24% at 32 MB), as compression overhead partially offsets bandwidth savings.

We generate `bfloat16` tensors with values uniformly distributed in $[-1, 1]$, achieving a stable compression ratio of approximately 0.64 across all tensor sizes. For small messages (e.g., 4 MB), compression overhead cannot be fully amortized. However, in practical workloads such as RL weight updates and KV cache transfers, tensor sizes typically exceed this threshold, making LZIP-P2P effective in practice.

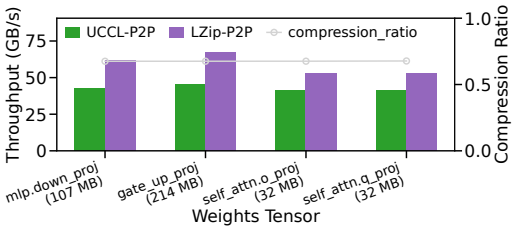
5.3 APPLICATION-LEVEL PERFORMANCE: WEIGHT UPDATES IN REINFORCEMENT LEARNING

We evaluate LZIP-P2P on weight tensors generated during the update phase of LLM reinforcement learning. We consider two representative models: the dense GLM4-9B (9B parameters) and the mixture-of-experts (MoE) model Qwen3.5-35B-A3B (35B parameters). The training pipeline runs on 8 GPUs across two nodes over EFA.

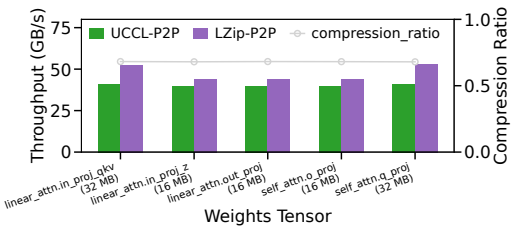
Figure 5 presents the communication throughput across representative weight tensors from different layers and training stages. The x-axis lists tensor names and sizes.

For the dense GLM4-9B model (Figure 5a), weight tensors are relatively large, enabling substantial communication reduction. In particular, LZIP-P2P achieves up to 47.5% higher throughput for the `gate_up_proj` tensor (214 MB).

For the MoE model Qwen3.5-35B-A3B (Figure 5b), tensors are smaller due to sparse expert activation. Despite this, LZIP-P2P still delivers up to 28.8% improvement for the `self_attn.q_proj.weight` tensor (32 MB). For smaller tensors (e.g., 16 MB), LZIP-P2P



(a) Weight update throughput for GLM4-9B.



(b) Weight update throughput for Qwen3.5-35B-A3B.

Figure 5: Application-level evaluation of LZIP-P2P on `bfloat16` weight tensors during RL training: (a) GLM4-9B and (b) Qwen3.5-35B-A3B.

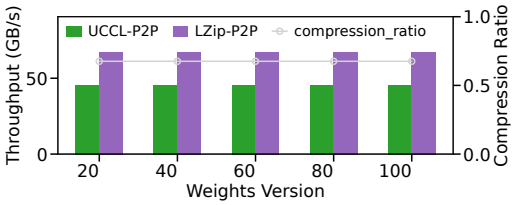
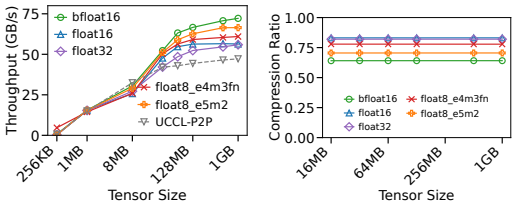


Figure 6: Communication throughput for different versions of the `gate_up_proj` weight tensor (214 MB) during GLM4-9B RL training.



(a) LZIP-P2P throughput.

(b) Compression ratio.

Figure 7: Throughput and compression ratios of LZIP-P2P across floating-point types.

maintains around 10% improvement. Finally, the compression ratio remains stable across models, layers, and training stages. As it directly determines communication reduction, this stability indicates that our lossless design generalizes well to diverse LLM training workloads.

5.4 Ablation Studies

5.4.1 Transmission Throughput and Compression Ratio Across RL Steps

We examine compression behavior during reinforcement learning training by measuring throughput and compression ratio for different versions of the `gate_up_proj` weight tensor (214 MB) from GLM4-9B. As shown in Figure 6, the compression ratio remains stable across training checkpoints, close to that of randomly generated tensors. Correspondingly, LZIP-P2P consistently outperforms UCCL-P2P, achieving stable throughput gains throughout training.

5.4.2 Performance Across Different Floating-Point Data Types

We evaluate LZIP-P2P across different floating-point formats, including `bfloat16`, `float16`, `float32`, `float8_e4m3fn`, and `float8_e5m2` (Figure 7). The achievable compression ratio is primarily determined by the floating-point exponent field, as our scheme encodes only exponent values. In IEEE formats, `float16` uses 5 exponent bits out of 16 bits, while `float32` uses 8 exponent bits out of 32 bits. Consequently, both formats offer limited compression opportunities compared to `bfloat16`, which retains an 8-bit exponent within a 16-bit representation. This trend is reflected in Figure 7b, where the compression ratios for `float16`, `float32`, `bfloat16`, `float8_e4m3fn`, and `float8_e5m2` are approximately 0.83, 0.82, 0.64, 0.76, and 0.70, respectively.

These differences translate into performance gains in Figure 7a. LZIP-P2P achieves up to 41.9% throughput improvement for `float8_e5m2`, compared to 30.2% for `float8_e4m3fn` over UCCL-P2P. The highest performance benefits are observed for `bfloat16` tensors. Notably, `bfloat16` has become the dominant numerical format for large-scale LLM training due to its larger dynamic range and improved numerical stability (Micikevicius et al., 2018; Narayanan et al., 2021). These results demonstrate that LZIP-P2P delivers consistent performance benefits across the floating-point formats commonly used in modern LLM systems.

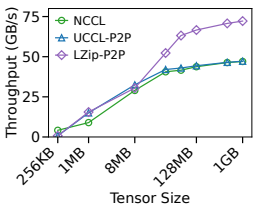


Figure 8: LZIP-P2P throughput versus NCCL `send/recv`.

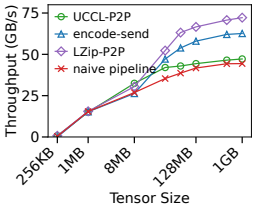
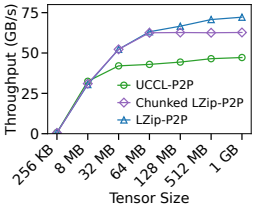
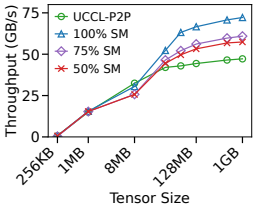


Figure 9: Compression–communication integration strategies.



(a) GPU memory constraints.



(b) Streaming multiprocessor (SM) availability constraints.

Figure 10: Throughput of LZIP-P2P under constrained GPU resources.

5.4.3 COMPARISON WITH NCCL `SEND/RECV`

We further compare baseline UCCL-P2P implementation and LZIP-P2P against the NCCL `send/recv` interface. The UCCL-P2P implementation achieves performance comparable to NCCL for tensors larger than 8 MB and slightly outperforms NCCL for smaller messages (Figure 8). By incorporating compression, LZIP-P2P further improves throughput and consistently outperforms NCCL across the entire range of tensor sizes.

5.4.4 COMPARISON WITH NAIVE METHODS

We compare LZIP-P2P against two alternative designs for integrating compression with communication. The first, *encode-send*, performs transmission only after completing the entire compression pipeline. The second, *naive pipeline*, partitions data into 8 MB chunks and overlaps compression with communication. Figure 9 shows that LZIP-P2P consistently outperforms both alternatives across tensor sizes. For small tensors (e.g., 8 MB), compression overhead dominates, and *encode-send* degrades throughput by 18% relative to UCCL-P2P, while LZIP-P2P reduces this to 6% via partial overlap. For larger tensors, LZIP-P2P outperforms *encode-send* by effectively overlapping compression and communication. The *naive pipeline* slightly underperforms UCCL-P2P, as fine-grained chunking incurs excess compression overhead that outweighs pipelining benefits.

5.4.5 PERFORMANCE UNDER RESOURCE CONSTRAINTS

We evaluate the robustness of LZIP-P2P under constrained GPU memory and limited SM availability. The results show that LZIP-P2P maintains substantial performance gains even under tight resource budgets.

Memory Footprint. LZIP-P2P incurs additional GPU memory overhead for staging compression buffers, scaling with block granularity. To reduce memory usage, large tensors can be partitioned into smaller chunks at the cost of extra kernel launches. Despite this trade-off, LZIP-P2P remains effective under tight memory constraints: with a 164 MB buffer per GPU (approximately 0.1% of the 141 GB high-bandwidth memory (HBM) on H200), chunked LZIP-P2P still achieves up to 41.2% throughput improvement for 128 MB tensors (Figure 10a).

SM Utilization. We evaluate LZIP-P2P under restricted SM availability using CUDA Green Context (NVIDIA, 2025b). As shown in Figure 10b, LZIP-P2P achieves up to 20.4% improvement even with 50% of SMs. LZIP-P2P leverages otherwise idle SMs during communication, as dependencies often prevent LLM communication from fully overlapping with computation.

5.4.6 LZIP-P2P THROUGHPUT ON AMD GPUS AND NICS

We evaluate LZIP-P2P on an AMD cluster with MI355X GPUs and Pensando DSC SmartNICs. Figure 11 shows that LZIP-P2P consistently improves end-to-end P2P throughput across tensor sizes, achieving up to 45.3% higher throughput at large sizes. These results demonstrate that the benefits of LZIP-P2P extend beyond a specific hardware stack and remain effective across both NVIDIA/EFA and AMD/Pensando platforms.

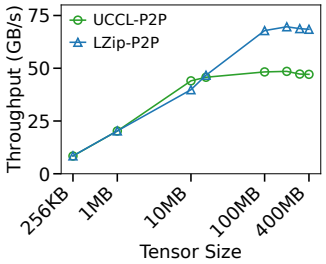


Figure 11: LZIP-P2P throughput on AMD MI355X over RoCEv2.

6 RELATED WORK

6.1 GPU P2P COMMUNICATION LIBRARY

Prior work has improved the efficiency of GPU P2P communication. Systems such as NIXL (NVIDIA, 2025a), Mooncake (Mooncake Project, 2024), pplx-garden (Licker et al., 2025), UCCL-P2P (Zhou et al., 2025; UCCL Project, 2025), NVSHMEM (Langer et al., 2022), NCCL (Jeaugey, 2017), and RCCL (AMD, 2024) provide high-performance P2P communication abstractions. However, none of these systems natively integrate compression into the communication pipeline.

6.2 LOSSLESS COMPRESSION ALGORITHMS ON GPUS

Existing work explores a variety of lossless GPU compression techniques. ZipNN (Hershcovitch et al., 2025) and DFloat11 (Zhang et al., 2025) employ Huffman-based entropy coding, while NeuZip (Hao et al., 2024), TDT (Jamalidinan & Cheshmi, 2025), and DietGPU (Meta AI Research, 2022) adopt ANS-based designs optimized for GPU execution. MPC (Yang et al., 2015) targets massively parallel architectures, and ZipServ (Fan et al., 2026) proposes hardware-aware compression with fixed-length encoding. Earlier efforts such as Deep Compression (Han et al., 2016) apply Huffman coding to quantized models, while nvCOMP (NVIDIA, 2023a) provides general-purpose GPU compression. However, these approaches optimize compression in isolation, without integrating it into GPU communication. Concurrent work ZipCCL (Lin et al., 2026) also couples lossless compression with GPU communication, but targets collective operations in LLM training and centers on codec design, whereas LZIP-P2P targets P2P communication and hides compression latency within the transfer itself.

6.3 GPU COMMUNICATION LIBRARY WITH COMPRESSION

Some prior work reduces communication volume in GPU communication via quantization or other lossy compression techniques. Representative systems, including C-Coll (Huang et al., 2024b), ZCCL (Huang et al., 2025b), gZCCL (Huang et al., 2024a), OmniReduce (Fei et al., 2021), and ghZCCL (Huang et al., 2025a), leverage error-bounded lossy compression to accelerate collective communication on GPU clusters. At the system level, Zhou et al. (2021) extend the MVAPICH2 MPI library with both lossless and lossy compression, primarily targeting HPC workloads rather than modern LLM training. In addition, a broad class of frameworks (Li et al., 2024a; Sun et al., 2025; Chen et al., 2025; 2023; Xu et al., 2021; Li et al., 2024b; Takezawa et al., 2023; Wang et al., 2023; Alistarh et al., 2017) applies lossy compression to accelerate communication during training or inference. While these approaches improve communication efficiency, they introduce approximation error, which can hinder convergence or degrade model accuracy.

7 CONCLUSION

The rapid scaling of large language models has made GPU communication a key performance bottleneck. In this work, we introduced LZIP-P2P, an approach that embeds lossless compression into GPU communication primitives. LZIP-P2P supports point-to-point communication without sacrificing numerical fidelity. Extensive evaluation across multiple GPU clusters shows that LZIP-P2P consistently improves communication throughput.

AI USE STATEMENT

In this work, we used generative AI tools for literature analysis, including searching for and summarizing related work, and for checking bibliographic records against their original sources. We additionally used generative AI tools to draft and edit portions of the manuscript text. The synthetic data used in our experiments, the conceptual framework of LZIP-P2P, and the claims made in this paper were produced mainly by the authors. The design of LZIP-P2P, its implementation, and the experiments reported in Section 5 were carried out by the authors; generative AI tools provided partial assistance in analyzing and interpreting the resulting measurements, and every such analysis was checked against the raw measurements by the authors. We have reviewed all AI-assisted work: every AI-suggested reference was verified against the original publication, and all AI-assisted text was read and revised by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The design of LZIP-P2P and its *split-send* pipeline are described in Section 3, and Section 4 details how LZIP-P2P is integrated into UCCL-P2P, including the metadata and receiver-side changes and the extension of DietGPU to additional floating-point formats. Section 5 specifies the hardware platforms, interconnects, baselines, and tensor workloads used in every experiment, together with the GPU resource limits under which LZIP-P2P is evaluated. An anonymized implementation of LZIP-P2P, sufficient to reproduce the reported measurements, is available at `https://anonymous.4open.science/r/iclr27-artifact/`.

REFERENCES

Dan Alistarh, Demjan Grubic, Jerry Li, et al. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, NeurIPS '17, 2017.

AMD. RCCL: AMD ROCm collective communication library. `https://github.com/ROCm/rccl`, 2024. Accessed: 2026-09-14.

Noushin Azami, Alex Fallin, and Martin Burtscher. Efficient lossless compression of scientific floating-point data on CPUs and GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, 2025. doi: 10.1145/3669940.3707280.

Chen-Chun Chen, Yu-Min Chou, and Jerry Chou. PHY: A performance-driven hybrid communication compression method for distributed training. *Journal of Parallel and Distributed Computing*, 180:104719, 2023. doi: 10.1016/j.jpdc.2023.104719.

Chuyan Chen, Yutong He, Pengrui Li, et al. Greedy low-rank gradient compression for distributed learning with convergence guarantees, 2025. arXiv:2507.08784.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. PaLM: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(1), 2023.

DeepSeek-AI, Aixin Liu, Bei Feng, et al. DeepSeek-V3 technical report, 2025. arXiv:2412.19437.

Ruibo Fan, Xiangrui Yu, Xinglin Pan, et al. ZipServ: Fast and memory-efficient LLM inference with hardware-aware lossless compression. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 2264–2280, New York, NY, USA, 2026. Association for Computing Machinery. doi: 10.1145/3779212.3790250.

Jiawei Fei, Chen-Yu Ho, Atal N. Sahu, et al. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proceedings of the 2021 ACM SIGCOMM Conference*, SIGCOMM '21, pp. 676–691, 2021. doi: 10.1145/3452296.3472904.

Team GLM, Aohan Zeng, Bin Xu, et al. ChatGLM: A family of large language models from GLM-130B to GLM-4 all tools, 2024. arXiv:2406.12793.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models, 2024. arXiv:2407.21783.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016*, 2016. arXiv:1510.00149.

Yongchang Hao, Yanshuai Cao, and Lili Mou. NeuZip: Memory-efficient training and inference with dynamic compression of neural networks, 2024. arXiv:2410.20650.

Horace He et al. Defeating nondeterminism in LLM inference. `https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/`, 2025. Thinking Machines Lab: Connectionism.

Moshik Hershcovitch, Andrew Wood, Leshem Choshen, et al. ZipNN: Lossless compression for AI models. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pp. 186–198, 2025. doi: 10.1109/CLOUD67622.2025.00028.

Jiajun Huang, Sheng Di, Xiaodong Yu, et al. gZCCL: Compression-accelerated collective communication framework for GPU clusters. In *Proceedings of the 38th ACM International Conference on Supercomputing*, ICS '24, pp. 437–448, 2024a. doi: 10.1145/3650200.3656636.

Jiajun Huang, Sheng Di, Xiaodong Yu, et al. An optimized error-controlled MPI collective framework integrated with lossy compression. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 752–764, 2024b. doi: 10.1109/IPDPS57955.2024.00072.

Jiajun Huang, Sheng Di, Yafan Huang, et al. ghZCCL: Advancing GPU-aware collective communications with homomorphic compression. In *Proceedings of the 39th ACM International Conference on Supercomputing*, ICS '25, pp. 43–56, 2025a. doi: 10.1145/3721145.3733642.

Jiajun Huang, Sheng Di, Xiaodong Yu, et al. ZCCL: Significantly improving collective communication with error-bounded lossy compression, 2025b. arXiv:2502.18554.

Samirasadat Jamalidinan and Kazem Cheshmi. Floating-point data transformation for lossless compression, 2025. arXiv:2506.18062.

Sylvain Jeaugey. NCCL: Optimized primitives for collective multi-GPU communication. `https://developer.nvidia.com/nccl`, 2017.

Akhil Langer, Seth Howell, Sreeram Potluri, et al. Dynamic symmetric heap allocation in NVSHMEM. In *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Exascale and Smart Networks (OpenSHMEM 2021)*, volume 13159 of *Lecture Notes in Computer Science*, pp. 187–198. Springer, 2022. doi: 10.1007/978-3-031-04888-3_12.

Minghao Li, Ran Ben Basat, Shay Vargaftik, et al. THC: Accelerating distributed deep learning using tensor homomorphic compression. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, NSDI '24, pp. 1191–1211. USENIX Association, 2024a.

Xue Li, Cheng Guo, Kun Qian, et al. Near-lossless gradient compression for data-parallel distributed DNN training. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '24, pp. 977–994, 2024b. doi: 10.1145/3698038.3698541.

Nandor Licker, Kevin Hu, Vladimir Zaytsev, et al. fabric-lib: RDMA point-to-point communication for LLM systems, 2025. arXiv:2510.27656.

Wenxiang Lin, Xinglin Pan, Ruibo Fan, et al. ZipCCL: Efficient lossless data compression of communication collectives for accelerating LLM training, 2026. arXiv:2604.27844.

Meta AI Research. DietGPU. `https://github.com/facebookresearch/dietgpu`, 2022. GitHub repository. Accessed: 2026-03-07.

Paulius Micikevicius, Sharan Narang, Jonah Alben, et al. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

Mooncake Project. Mooncake transfer engine. https://github.com/kvcache-ai/Mooncake, 2024. Accessed: 2026-09-14.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, et al. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021. doi: 10.1145/3458817.3476209.

NVIDIA. nvCOMP: NVIDIA GPU data compression library. https://github.com/NVIDIA/nvcomp, 2023a. Accessed: 2023-07-31.

NVIDIA. NVIDIA CUDA C programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/, 2023b.

NVIDIA. NIXL: NVIDIA inference xfer library. https://github.com/ai-dynamo/nixl, 2025a.

NVIDIA. CUDA green contexts. https://docs.nvidia.com/cuda/cuda-programming-guide/04-special-topics/green-contexts.html, 2025b. CUDA Programming Guide, Section 4.6. Accessed: 2026-09-14.

Qwen, An Yang, Baosong Yang, et al. Qwen2.5 technical report, 2025. arXiv:2412.15115.

Baixi Sun, Weijin Liu, J. Gregory Pauloski, et al. COMPSO: Optimizing gradient compression for distributed training with second-order optimizers. In *Proceedings of the 30th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '25, pp. 212–224, 2025. doi: 10.1145/3710848.3710852.

Yuki Takezawa, Kenta Niwa, and Makoto Yamada. Communication compression for decentralized learning with operator splitting methods. *IEEE Transactions on Signal and Information Processing over Networks*, 9:581–595, 2023. doi: 10.1109/TSIPN.2023.3307894.

UCCL Project. Everything you want to know about KV cache transfer engine. https://uccl-project.github.io/posts/kv-transfer-engine/, 2025. Blog post, August 13, 2025. Accessed: 2026-09-14.

Guanhua Wang, Heyang Qin, Sam Ade Jacobs, et al. ZeRO++: Extremely efficient collective communication for giant model training, 2023. arXiv:2306.10209.

Hang Xu, Chen-Yu Ho, Ahmed M. Abdelmoniem, et al. GRACE: A compressed communication framework for distributed machine learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, ICDCS '21, pp. 561–572, 2021. doi: 10.1109/ICDCS51616.2021.00060.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. arXiv:2505.09388.

Annie Yang, Hari Mukka, Farbod Hesaaraki, et al. MPC: A massively parallel compression algorithm for scientific data. In *2015 IEEE International Conference on Cluster Computing*, pp. 381–389, 2015. doi: 10.1109/CLUSTER.2015.59.

Tianyi Zhang, Mohsen Hariri, Shaochen Zhong, et al. 70% size, 100% accuracy: Lossless LLM compression for efficient GPU inference via dynamic-length float (DFloat11). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Qinghua Zhou, Ching-Hsiang Chu, N. S. Senthil Kumar, et al. Designing high-performance MPI libraries with on-the-fly compression for modern GPU clusters. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS '21, pp. 444–453, 2021. doi: 10.1109/IPDPS49936.2021.00053.

Yang Zhou, Zhongjie Chen, Ziming Mao, et al. An extensible software transport layer for GPU networking, 2025. arXiv:2504.17307.