# HetCCL: Efficient LLM Training on Heterogeneous Vendor GPUs

**Heehoon Kim**[1]*† **Chris Jaehwan Lee**[2]* **Taejeoung Kim**[3] **Jongwon Park**[2]
**Jinpyo Kim**[2] **Pyongwon Suh**[3] **Ryan H. Choi**[3] **Sangwoo Lee**[3] **Jaejin Lee**[2,4]

[1]Moreh Inc. [2]Dept. of Computer Science and Engineering, Seoul National University
[3]Samsung Research [4]Graduate School of Data Science, Seoul National University
heehoon.kim@moreh.io, {jaehwan,jongwon.park,jinpyo}@aces.snu.ac.kr
{tj76.kim,pyongwon.suh,ryan.choi,swa.lee}@samsung.com, jaejin@snu.ac.kr

## Abstract

The rapid growth of large language models is driving organizations to expand their GPU clusters, often with GPUs from multiple vendors. However, current deep learning frameworks lack support for communication across heterogeneous vendor GPUs, leading to inefficiency and higher costs. We present HetCCL, a collective communication library that unifies vendor-specific backends and enables RDMA-based communication across GPUs without requiring driver modifications. HetCCL introduces two novel mechanisms that enable cross-vendor communication while leveraging optimized vendor libraries, NVIDIA NCCL and AMD RCCL. Evaluations on a multi-vendor GPU cluster show that HetCCL matches NCCL and RCCL performance in homogeneous setups while extending support to heterogeneous environments, enabling practical and efficient training across NVIDIA and AMD GPUs without changes to existing deep learning applications.

## 1 Introduction

The recent emergence of trillion-scale deep learning (DL) models [1, 13, 14] has enabled solutions to previously intractable problems. A key enabler behind this advancement is the high computational capability of heterogeneous cluster systems equipped with various hardware accelerators (e.g., GPUs, NPUs, and FPGAs), which makes it feasible to train and run inference on such massive models. Among these systems, GPU-based platforms play a dominant role in DL, with NVIDIA and AMD GPUs serving as the de facto standard [30, 47].

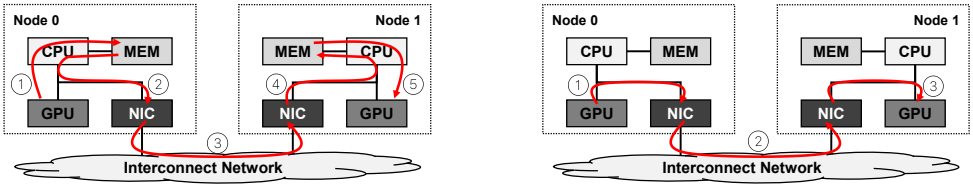
To enable efficient training and inference of DL models across multiple GPUs, a variety of parallelization techniques have been developed. The most representative approaches are data parallelism [54] and model parallelism, including tensor, pipeline, and expert parallelism [22, 61, 62]. Although these techniques differ in how computations and data are distributed across GPUs, they share a common requirement for efficient coordination across devices. This requirement has become increasingly important as modern ML systems rely on parallelization to improve hardware utilization [41, 64].

In multi-GPU parallel training or inference, inter-GPU communication is crucial for sharing computation results and synchronizing model states. Collective communication operations, such as All-Reduce, All-Gather, and Reduce-Scatter, are commonly employed for this purpose. Consequently, most accelerator vendors provide optimized collective communication libraries (CCLs) tailored to their hardware. Examples include NVIDIA NCCL [44], AMD RCCL [9], and Intel oneCCL [23].

Training large-scale models typically requires GPU clusters with sufficient compute capacity. In practice, many academic and research institutions expand their clusters by acquiring GPU systems in a

*Equal contribution. †This work was done while a Ph.D. student at Seoul National University.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).




(a) Without RDMA (b) With RDMA

Figure 1: Communication between GPUs across nodes.

piecemeal fashion [12, 49]. This incremental procurement approach allows continued use of existing resources while integrating newer hardware. As a result, the infrastructure becomes heterogeneous, comprising various types of GPUs, sometimes even from different vendors.

While GPUs from different vendors share similar design philosophies and architectures, they require distinct frameworks and toolchains, making cross-vendor software development challenging. Thanks to ongoing efforts across the DL ecosystem, full-featured DL frameworks are now available for each vendor's GPUs. For instance, PyTorch [50] can be configured to support either NVIDIA or AMD GPUs. However, parallel training across GPUs from different vendors remains infeasible because of the incompatibility of their communication backends. DL frameworks depend on vendor-specific libraries, such as NCCL for NVIDIA GPUs and RCCL for AMD GPUs, which are not interoperable.

As DL workloads scale to larger sizes, the ability to achieve fast and efficient communication across heterogeneous GPUs is becoming a crucial factor for building practical and cost-effective AI infrastructure. Investigating such communication, particularly between NVIDIA and AMD GPUs, is not only timely but also essential for the advancement of the next generation of distributed ML systems. For this purpose, this paper presents HetCCL, a novel CCL that supports GPUs from multiple vendors. To the best of our knowledge, HetCCL is the first work to enable transparent utilization of all GPUs in a multi-vendor heterogeneous cluster. No source code modifications are required at any level, including the driver, runtime, compiler, and application. In particular, it supports NVIDIA and AMD GPUs, the two leading vendors by market share, which together dominate the accelerator market (approximately 88% and 12%, respectively) [30]. By replacing original communication backends (i.e., NCCL and RCCL) with HetCCL, existing distributed training code written in DL frameworks (e.g., PyTorch) can use both vendors' GPUs without further modifications.

The key contributions are summarized as follows:

- We propose HetCCL, the first cross-vendor CCL that enables DL model training on heterogeneous clusters with both NVIDIA and AMD GPUs.
- We show that direct RDMA communication between GPUs from different vendors is possible without modifying GPU drivers or runtimes.
- We describe the design and implementation of efficient collective communication between GPUs from different vendors, with an emphasis on challenges, such as abstracting heterogeneous runtime APIs and integrating vendor-specific code into a unified framework.
- We evaluate the training performance of large language models on a multi-vendor GPU cluster and demonstrate that HetCCL achieves higher aggregate throughput than homogeneous baselines while mitigating straggler effects and preserving model accuracy.

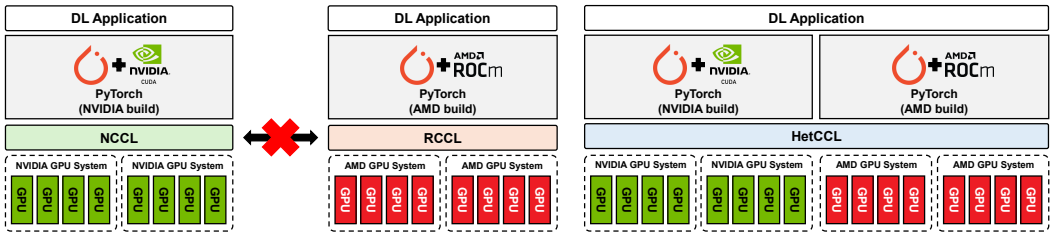
We discuss the limitations of this work and outline directions for future research in Appendix 6.

## 2 Background and Related Work

This section provides an overview of GPU communication mechanisms and reviews relevant prior work. An extended discussion is provided in Appendix A.

### 2.1 Communication Mechanisms of GPUs

**Intra-node communication.** GPUs within the same node communicate via either *peer-to-peer (P2P)* or *shared memory (SHM)* mode, depending on their interconnect and PCIe topology. While




(a) Previous work (b) This work (HetCCL)

Figure 2: Comparison between the previous approach and HetCCL.

P2P enables direct GPU-to-GPU transfers and offers the highest performance, it may be unavailable or inefficient across PCIe root complexes, in which case communication falls back to SHM, which stages data through host memory and provides lower throughput [21].

**Inter-node communication and RDMA.** In a multi-node environment, GPUs located on different nodes cannot directly access each other's memory. The most basic inter-node transfer mechanism relies on the host memory as a staging area at both the sending and receiving ends (Figure 1a). This approach incurs memory copy overhead and suffers from the limited bandwidth of the host memory. For faster inter-node communication between GPUs, modern GPUs support *Remote Direct Memory Access* (RDMA) [57], which allows a network interface card (NIC), such as InfiniBand (IB) [51] and RoCE [26], to access GPU memory directly, bypassing the CPU (Figure 1b). The capability is often marketed under vendor-specific names, such as NVIDIA's GPUDirect [43] and AMD's DirectGMA [8]. Enabling RDMA requires that the device memory be allocated using vendor-specific APIs (e.g., `(cuda/hip)Malloc`) and be registered with the IB Verbs API [35]. The registration enables RDMA-capable NICs to directly access GPU memory regions via Verbs' `send/recv` API.

### 2.2 Existing CCLs for GPUs

**Vendor-provided libraries.** NCCL [44] is the most widely used CCL optimized for NVIDIA GPU environments. RCCL [9], developed by AMD, is a HIP-based implementation that mirrors much of NCCL's design and interface, but is optimized for AMD GPU systems. Intel's oneCCL [23] targets heterogeneous CPU-GPU systems, but it is restricted to Intel hardware. MSCCL++ [39, 59], proposed by Microsoft, is a programmable framework that can be compiled for either NVIDIA or AMD GPUs, but does not support communication across GPUs from different vendors. Meta developed TorchComms [37, 63] to improve the scalability of collective communication within large-scale NVIDIA GPU clusters. These vendor libraries are limited to homogeneous GPU environments and do not allow communication between GPUs from different vendors.

**Other libraries.** MPI [68] is a widely used communication standard in HPC, but is not specifically optimized for the collective communication patterns of modern accelerator-based DL workloads. TCCL [27] is an extension of NCCL that optimizes communication for PCIe-dependent systems by profiling and selecting low-congestion paths; however, it is limited to NVIDIA GPUs. HiCCL [19] builds hierarchical collectives from primitive operations with hierarchy-aware optimizations. UCCL [73] focuses on an extensible transport layer that decouples collective libraries from underlying network mechanisms to improve portability and flexibility. Nonetheless, neither HiCCL nor UCCL supports executing a single collective operation across GPUs from different vendors.

## 3 Motivation

Efficient coordination among heterogeneous GPUs has become increasingly important as large language model (LLM) training compute scales and clusters are expanded to integrate new hardware while retaining existing GPUs. This process results in heterogeneous deployments across GPU generations or capabilities [31, 49]. Unlike hyperscalers that can provision massive homogeneous GPU clusters, academic and institutional mid-scale clusters often rely on incrementally expanded mixed-resource clusters, motivating studies on heterogeneous GPU systems [12, 31, 49, 66, 70] that optimize parallelization and load balancing across diverse accelerators as described in Appendix A.2.

Recently, heterogeneous deployments across GPU vendors are also emerging. Cost and supply constraints of dominant vendors have accelerated the adoption of alternative GPUs from other manufacturers [40]. This trend is evident in real-world infrastructure: major cloud providers [7, 38, 46] and cluster orchestration platforms [18, 29] now offer both NVIDIA and AMD GPUs.

However, prior work on heterogeneous GPU systems primarily targets single-vendor heterogeneity (as shown in Table 1), such as different generations of GPUs from the same vendor, and distributed training across multi-vendor GPUs remains largely unsupported. As illustrated in Figure 2a, DL frameworks like PyTorch rely on vendor-specific backends (e.g., NCCL for NVIDIA and RCCL for AMD), which prevents interoperability between heterogeneous GPUs within a single training job. Thus, the absence of a cross-vendor CCL remains a fundamental barrier to building flexible, cost-effective ML systems. To overcome this limitation, we propose HetCCL, which unifies vendor-specific stacks and enables a single training system to run across NVIDIA and AMD GPUs as illustrated in Figure 2b.

# 4 The HetCCL Design

In this section, we describe the design of HetCCL. We first present four intertwined challenges in designing a cross-vendor CCL, followed by the key insights underlying HetCCL. We then detail HetCCL's approach, which is implemented through two novel mechanisms. Finally, we discuss how HetCCL enables efficient training without code modification or load imbalance.

## 4.1 Challenges

**Near-native performance in homogeneous settings.** Maintaining near-native performance in homogeneous environments is a strict requirement that necessitates the use of vendor-provided CCLs. These libraries are closely aligned with GPU microarchitectures and include extensive vendor-specific optimizations. Although recent MPI implementations [15, 48] provide broad portability, they typically do not achieve the performance levels of NCCL or RCCL for large-scale data transfers [10]. Similarly, most previous CCLs [27, 63] are built on vendor-provided libraries rather than serving as complete replacements. This challenge is intensified by the close integration of NCCL and RCCL within most DL frameworks [50, 56, 62] and by the rapid advancement of vendor libraries, which frequently introduce new hardware features and optimizations [21]. Consequently, although possible, developing a new library from the ground up would be inefficient and inherently limited in performance.

**High performance in heterogeneous settings.** Achieving performance comparable to homogeneous communication in heterogeneous environments remains challenging without high-performance transfer mechanisms. Near-homogeneous performance is typically achievable only through zero-copy RDMA. In contrast, host-mediated relay approaches, such as those implemented in KAITIAN [33], lead to substantial performance degradation. Moreover, RDMA provides only data movement primitives and does not address the scheduling or synchronization of collective operations. As a result, critical coordination challenges persist in heterogeneous collectives.

**Heterogeneity in software stacks across vendors.** Cross-vendor GPU communication is further complicated by fundamental discrepancies in software stacks, including differences in drivers, runtime APIs, and compilation toolchains. Although libraries such as MSCCL++ [39] and TorchComms [37] claim to support both NVIDIA and AMD GPUs, they rely on compile-time selection of a single backend (NCCL or RCCL), which prevents simultaneous cross-vendor execution. This limitation arises because CUDA and HIP expose distinct runtime interfaces, leading these libraries to include only one vendor-specific code path in the generated binary (i.e., host code). Additionally, GPU kernels must be compiled with vendor-specific compilers (e.g., `nvcc` or `hipcc`), producing assembly code compatible only with the corresponding hardware. MSCCL++ and TorchComms employ a single compiler toolchain at build time. As a result, existing designs inherently restrict execution to a single vendor, and separate vendor-specific builds would therefore need a shared control protocol to form a cross-vendor collective.

**Performance disparity and stragglers in heterogeneous collectives.** Even when heterogeneous communication achieves performance close to that of a homogeneous setup, collective operations remain constrained by the slowest device in the group. This performance disparity leads to straggler

effects, intensified by the frequent collective communication operations required in distributed training. It can substantially reduce throughput in large-scale DL workloads such as LLMs.

### 4.2 Observation

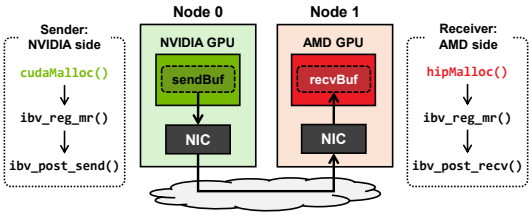


Figure 3: RDMA-based communication between NVIDIA and AMD GPUs in different nodes.

There is a common misconception that RDMA between GPUs from different vendors requires special modifications, such as reimplementing GPU drivers. However, we observe that once a memory region—whether in host memory or device memory managed by different GPU architectures—is properly registered with a NIC, the underlying RDMA stack abstracts away vendor-specific differences. We also find that device memory is interoperable across vendors without requiring any data layout transformation (e.g., endianness conversion). As a result, heterogeneous inter-GPU RDMA is feasible without modifying GPU drivers or runtimes, provided that GPU memory is properly registered with RDMA-capable NICs. Once GPU memory is exposed to the PCIe address space via GPUDirect or DirectGMA, data transfers are handled by the RDMA stack in a vendor-agnostic manner, and the receiver does not need to know the sender's GPU vendor or memory type. Figure 3 illustrates this mechanism with an example where one node is equipped with an NVIDIA GPU and the other with an AMD GPU. Each GPU allocates communication buffers using its own runtime API (i.e., `(cuda/hip)Malloc`), registers them with the local NIC via `ibv_reg_mr`, and performs RDMA `send/recv` operations. Here, standard RDMA setup procedures (e.g., QP creation and completion polling) are omitted for brevity. With modern hardware and driver support, this mechanism can be applied broadly across heterogeneous systems without requiring cross-vendor driver integration. When RDMA is unavailable, the system can gracefully fall back to staging the data through the host memory.

**Goals.** HetCCL aims to enable efficient collective communication across cross-vendor GPUs, pursuing three goals:

- In homogeneous settings, HetCCL must preserve near-native intra-node communication performance comparable to vendor-provided CCLs.
- In heterogeneous settings, HetCCL should achieve performance close to that of homogeneous systems by leveraging RDMA-based inter-node communication.
- HetCCL must maintain backward compatibility with existing DL frameworks, enabling drop-in integration without requiring any code modifications.

**Key Approach.** The key insight behind HetCCL is to decouple cross-vendor collective coordination from vendor-specific execution. Instead of reimplementing a new library, HetCCL adapts NCCL and RCCL collective algorithms and protocols into a common host-side implementation. Each rank loads kernels compiled for its local GPU vendor, allowing ranks from both vendors to participate in one collective without invoking NCCL or RCCL as separate libraries, thereby preserving vendor-native optimizations, remaining robust to future updates, and enabling drop-in integration with existing DL frameworks. To realize this design, HetCCL employs (1) a runtime API abstraction layer across vendors and (2) a platform-specific device code compilation strategy for multi-vendor kernel execution. This design further leverages our observation that RDMA enables efficient data movement across heterogeneous GPUs, allowing HetCCL to achieve near-homogeneous performance in inter-node settings. In addition, HetCCL mitigates performance imbalance across heterogeneous GPUs through GPU-aware micro-batch size adjustment, reducing straggler effects during collective synchronization.

### 4.3 Multi-Vendor Runtime API Abstraction

HetCCL leverages code from both NCCL and RCCL to facilitate high-performance communication across GPUs from different vendors. Although these libraries are developed separately, they share a similar structure and functionality. For example, their analogous high-level structures—specifically,

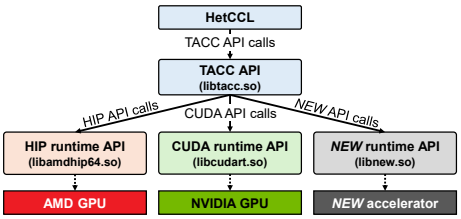



Figure 4: Runtime API abstraction layer.

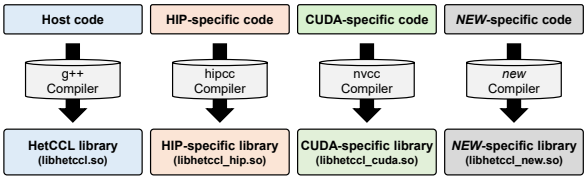



Figure 5: Device code compilation mechanism.

the host-side logic for partitioning and communicating data—facilitate their integration, allowing HetCCL to retain their collective algorithms and protocols in a common host-side control plane. Specifically, HetCCL uses the existing ring and tree algorithms, with each GPU performing reductions using its vendor-specific kernel. Intra-node P2P or SHM hops and inter-node RDMA hops proceed in the same pipeline, with network hops progressed by existing proxy threads. However, the primary challenge is that each library depends on a distinct set of runtime APIs and device kernels. This limits existing CCLs that support multiple vendors, such as MSCCL++ and TorchComms, which rely on compile-time macros to select a single vendor-specific API, thereby including only one of CUDA or HIP in the host-side library build and preventing simultaneous cross-vendor execution. HetCCL tackles these differences through a series of unification mechanisms.

HetCCL introduces a runtime API abstraction layer called *TACC*. TACC unifies the CUDA and HIP runtime APIs under a common platform-independent interface, as shown in Figure 4. Instead of directly calling platform-specific API functions, HetCCL invokes TACC APIs such as `taccMalloc` and `taccMemcpy`. During initialization, TACC automatically detects the platform through `taccSetPlatformAuto` (or selects it based on user input) and configures a platform-specific function table. Subsequent TACC API calls are then forwarded to the corresponding CUDA or HIP runtime APIs (e.g., `cudaMalloc` or `hipMalloc`). We list the TACC APIs in Appendix B.

TACC covers the CUDA and HIP runtime APIs used by NCCL and RCCL, allowing HetCCL to support multiple platforms without modifying the core communication logic. This design also simplifies future extension to other accelerators, since platform-specific changes can be localized within the abstraction layer. Potential feature gaps between NCCL and RCCL are not a fundamental limitation, because RCCL continuously incorporates new NCCL features through its forked codebase [9], while HetCCL relies on native NCCL or RCCL for vendor-local collectives when necessary.

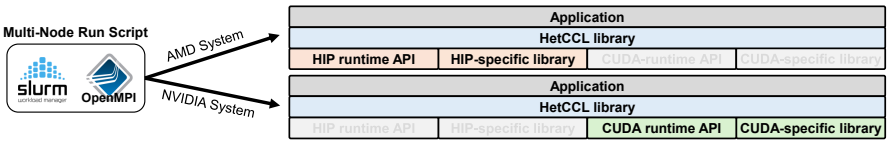



Figure 6: Multi-node execution flow on heterogeneous vendor GPU clusters.

### 4.4 Platform-Specific Device Code Compilation

Unlike runtime API functions, device code (or GPU kernels) cannot be abstracted because it requires compilation with vendor compilers, `nvcc` for NVIDIA and `hipcc` for AMD GPUs. Unlike prior work (e.g., MSCCL++ and TorchComms), which requires single-vendor compilation, HetCCL enables multi-vendor compilation by decoupling device-dependent code into standalone shared libraries independently compiled for each GPU platform. Figure 5 illustrates the compilation workflow of HetCCL. The host-side implementation is shared, while the CUDA and HIP device libraries are compiled separately. At execution time, HetCCL dynamically loads the appropriate backend library, resolves its kernel entry points, and issues kernel launches through a function pointer table initialized once at startup. Each abstracted kernel launch resolves to a single indirect call, incurring negligible overhead compared with microsecond-to-millisecond collective operations. This approach decouples platform-specific compilation from the core communication logic, allowing seamless extension to future accelerators. Only the corresponding kernel library needs to be implemented and loaded for new platforms.

### 4.5 Zero Code Modification

To avoid modifying application code, HetCCL can be deployed using the `LD_PRELOAD` trick [52]. This allows intercepting and overriding NCCL or RCCL symbols with the corresponding HetCCL implementation, enabling seamless integration with existing DL applications. Figure 6 illustrates a transparent, multi-vendor GPU cluster execution scenario. Only the necessary runtime and libraries for the GPU devices present on each node are required. HetCCL dynamically loads backend components (such as those for CUDA or HIP) as needed. HetCCL can be executed across multiple nodes using existing cluster execution utilities like `srun` from SLURM [71] and `mpirun` for MPI [15].

### 4.6 GPU-Aware Workload Balancing

In a cluster of homogeneous GPUs, using the same micro-batch size for each GPU is a common practice. However, our setup involves heterogeneous GPUs, specifically NVIDIA and AMD GPUs that have varying performance capabilities. Simply assigning the same batch size to all GPUs can lead to significant underutilization of the faster devices, as training only continues after all GPUs have processed their respective micro-batches. As a result, the overall throughput is limited by the slowest GPU in the group. To address this issue, we adopt a straightforward yet effective strategy: we adjust the micro-batch size according to the GPU's compute capability [25, 33].

Let $s_i$ denote the processing speed (tokens/s) of GPU $i$, obtained from brief offline profiling, and let $B$ be the total micro-batch size per step. We assign each GPU a micro-batch size $b_i$ proportional to its processing speed, i.e., $b_i = B \cdot \frac{s_i}{\sum_j s_j}$, with integer rounding. This ensures that all GPUs finish their local computation in approximately the same time, since $b_i/s_i$ is equalized across devices.

## 5 Evaluation

We evaluate HetCCL on a cluster comprising nodes equipped with GPUs from different vendors. We examine the communication performance and assess the overall performance of training DL models.

### 5.1 Experimental Setup

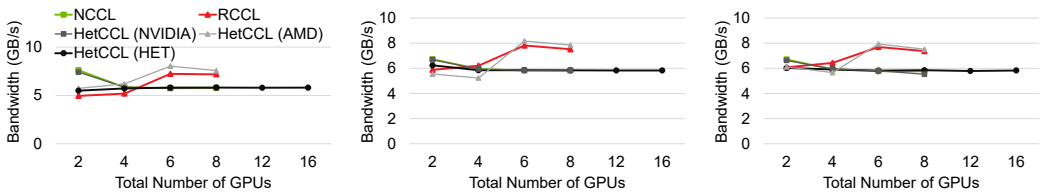
**System configuration.** We conduct experiments on a four-node cluster, consisting of two nodes equipped with four NVIDIA GPUs and two nodes with four AMD GPUs. This configuration allows us to evaluate communication performance across different GPU types (i.e., NVIDIA-to-AMD and vice versa). The detailed specifications of our system configuration are provided in Appendix C.1. Our results characterize cross-vendor training on this 16-GPU cluster, a scale comparable to the previous heterogeneous GPU systems summarized in Table 1.

**Comparison baselines.** We compare HetCCL with state-of-the-art vendor-provided libraries, NCCL [44] and RCCL [9], which also serve as core engines in existing CCLs [19, 59, 63, 73]. To our knowledge, no prior work supports cross-vendor GPU communication; we adopt these as baselines due to their vendor-specific optimizations and seamless integration with modern DL frameworks. We further evaluate HetCCL against MPI in Appendix D.5.

**Models and Implementation.** To evaluate end-to-end training performance with HetCCL, we use LLMs from two widely used model families, GPT [53] and LLaMA [16]. For each family, we evaluate multiple model sizes, covering a wide range of memory and computation intensity scenarios. Experiments are conducted using PyTorch [50] with DeepSpeed [56] for parallelization. We provide the model details in Appendix C.2 and implementation details in Appendix C.3.

### 5.2 Communication Performance

**Point-to-point communication.** Figure 7 shows the RDMA-based point-to-point communication bandwidth across data transfer sizes from 1KB to 1GB. We evaluate three variants of HetCCL: HetCCL (NVIDIA) for NVIDIA-only, HetCCL (AMD) for AMD-only, and HetCCL (HET) for heterogeneous NVIDIA-AMD communication. Vendor-optimized NCCL and RCCL are used as upper-bound baselines for homogeneous settings, where the measured peak bandwidths are bounded by the underlying GPU interconnects (PCIe Gen3 on NVIDIA nodes and PCIe Gen4 on AMD nodes).




(a) All-Reduce (b) All-Gather (c) Reduce-Scatter

Figure 8: Performance of representative collective communication operations.

HetCCL (NVIDIA) achieves performance comparable to NCCL, indicating negligible overhead in homogeneous NVIDIA environments. HetCCL (AMD) slightly outperforms RCCL due to more effective configuration choices; we observed that RCCL's default parameters are suboptimal on our system, and manual tuning (e.g., increasing `NCCL_MAX_NTHREADS`) narrows this gap. By integrating both NCCL and RCCL, HetCCL can selectively adopt configurations that improve overall performance. In heterogeneous cross-vendor communication, HetCCL (HET) achieves performance close to HetCCL (NVIDIA) or NCCL, as overall bandwidth is bounded by the slower endpoint. This demonstrates that HetCCL effectively enables cross-vendor RDMA communication while approaching near-native performance in homogeneous configurations. We discuss the effect of RDMA in Appendix E.1.

**Collective communication.** We evaluate the collective communication performance of NCCL, RCCL, and HetCCL under three configurations: HetCCL (NVIDIA), HetCCL (AMD), and HetCCL (HET) for heterogeneous NVIDIA+AMD setups. We cover all widely used collective operations—All-Reduce, All-Gather, Reduce-Scatter, Reduce, Broadcast, and All-to-All—using a 1GB data size, as large message sizes are representative of training scenarios (e.g., gradient accumulation during backpropagation) [59]. Figure 8 shows the achieved bandwidth of three collective operations; results for the remaining operations are reported in Appendix D.1. The x-axis denotes the total number of GPUs participating in the collective operation. For homogeneous setups, we evaluate up to 8 GPUs across two nodes, using 1-4 GPUs per node, which reflects the maximum homogeneous configuration available in our cluster. In contrast, HetCCL (HET) enables cross-vendor communication and scales beyond this limit, reaching up to 16 GPUs across four nodes.

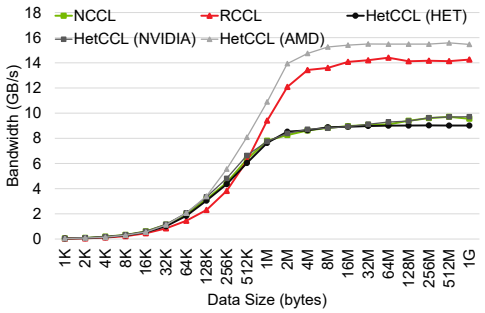
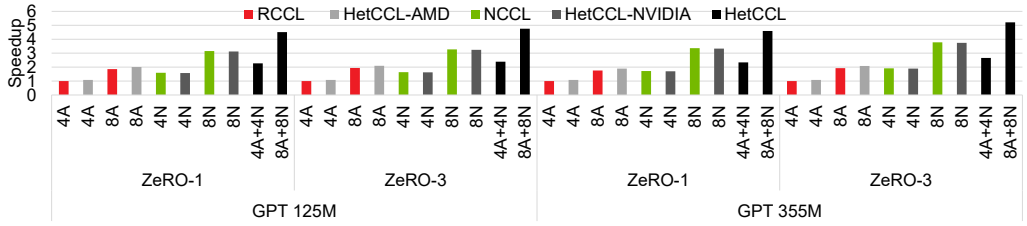



Figure 7: Performance of point-to-point communication with RDMA mechanism.

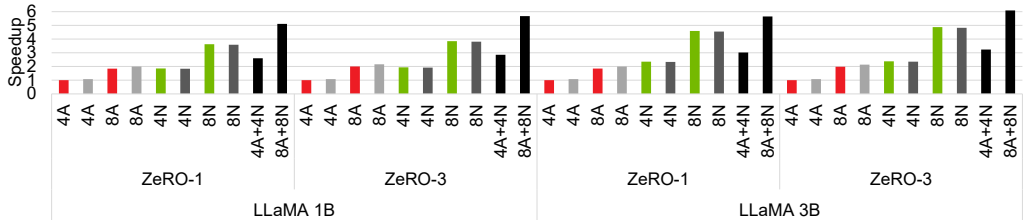
Across all operations, HetCCL (NVIDIA) achieves performance nearly identical to NCCL, indicating negligible overhead in homogeneous NVIDIA environments. HetCCL (AMD) performs comparably to or slightly better than RCCL due to more effective configuration choices. In heterogeneous settings, HetCCL (HET) consistently achieves performance bounded by the slower of the two vendor libraries. Depending on the scale and collective operation, either NCCL or RCCL becomes the bottleneck, which in turn limits HetCCL (HET). This behavior reflects HetCCL's design, which delegates communication to vendor-optimized backends for each GPU type. Moreover, HetCCL (HET) maintains stable bandwidth as the system scales to 12 and 16 GPUs, with 3 and 4 GPUs per node, respectively, demonstrating its efficiency and robustness in mixed-vendor environments.

### 5.3 Training DL Models

For efficient training, we use DeepSpeed ZeRO [54] and evaluate both ZeRO-1 and ZeRO-3, which differ in how model states are partitioned and in the resulting collective communication patterns. We describe the partitioning and communication characteristics of each ZeRO stage in Appendix C.4. Using initial profiling to quantify the NVIDIA-AMD GPU performance gap, HetCCL balances the load by proportionally assigning a larger micro-batch size to the faster GPUs. We discuss the effect of HetCCL's workload balancing on training performance as an ablation in Appendix E.2.




(a) GPT




(b) LLaMA

Figure 9: LLM training speedup. 4N/8N or 4A/8A denotes homogeneous NVIDIA or AMD GPU training, while 4A+4N/8A+8N denotes heterogeneous NVIDIA+AMD GPU training.

**End-to-end Performance.** Figure 9 reports throughput speedup relative to RCCL on AMD GPUs. We evaluate homogeneous configurations using only AMD GPUs (4A, 8A) or only NVIDIA GPUs (4N, 8N), as well as heterogeneous configurations combining both vendors (4A+4N and 8A+8N). These setups span from single-node configurations with 4 GPUs to multi-node configurations with up to 16 GPUs, enabling us to evaluate scalability across both GPU and node counts.

HetCCL remains close to these native baselines on homogeneous GPUs. By jointly utilizing NVIDIA and AMD GPUs, HetCCL achieves average speedups of $1.38\times$ over NVIDIA-only (NCCL) training and $2.69\times$ over AMD-only (RCCL) training, demonstrating effective aggregation of heterogeneous compute resources with minimal overhead and reduced straggler effects. Here, we use a normalized metric, *efficiency*, to quantify how closely heterogeneous GPU execution approaches the performance of homogeneous baselines. We define efficiency as the throughput of HetCCL's mixed-vendor execution (NVIDIA+AMD) normalized by the sum of throughputs from homogeneous NVIDIA and AMD runs. This ensures a fair comparison by comparing heterogeneous execution with homogeneous baselines using the same total number of GPUs. Across all configurations, HetCCL achieves up to 97% efficiency relative to the sum of homogeneous NCCL and RCCL throughputs, with workload balancing raising the efficiency for LLaMA-1B with ZeRO-3 from 89% to this level and improving throughput by $1.09\times$ (Table 10). Comparing ZeRO-1 and ZeRO-3 under identical configurations, we observe negligible differences in efficiency. HetCCL also achieves around 90% efficiency, with no significant difference between GPT and LLaMA.

HetCCL demonstrates near-linear scalability when scaling from 8 GPUs (4A+4N) to 16 GPUs (8A+8N). Across all models, speedup nearly doubles with the number of GPUs, indicating that HetCCL scales effectively in multi-node, mixed-vendor GPU environments. Unlike baselines limited to homogeneous GPUs, HetCCL enables multi-vendor GPUs to participate simultaneously, reducing time-to-solution and GPU-hour consumption, a common proxy for training cost. Additional results demonstrating HetCCL's compatibility with hybrid and expert parallelism are shown in Appendix D.4.

**Model Accuracy.** To examine model output consistency in heterogeneous vendor GPU training, we analyze training convergence using LLaMA-1B, trained on the WikiText-103 [36] dataset for 1K steps. We use BF16 precision and identical training configurations across NCCL (NVIDIA-only), RCCL (AMD-only), and HetCCL (NVIDIA+AMD) in this experiment.

We observe no discrepancies in loss across communication backends. The relative error of the final loss value across all pairwise comparisons among NCCL, RCCL, and HetCCL is below $7 \times 10^{-3}$, well within the numerical tolerance of BF16 (machine epsilon: $7.8125 \times 10^{-3}$). The training loss curves provided in Appendix D.2 demonstrate closely overlapping convergence behavior and identical final model states. These results indicate that heterogeneous GPU training does not harm model integrity, and that HetCCL preserves model accuracy at a level consistent with NCCL and RCCL.

## 6 Limitations and Future Work

While this work demonstrates that efficient training on a multi-vendor GPU cluster is possible with HetCCL, this section discusses the limitations of our study and directions for future work.

**System Scope.** HetCCL currently targets a system setting in which each node contains GPUs from a single vendor, while the overall cluster consists of nodes with different vendors. We chose this scope because intra-node mixed-vendor GPU configurations are uncommon in practice [20, 65, 74], and our current testbed also follows this deployment model. Accordingly, we did not include intra-node mixed-vendor experiments in this work. That said, the core mechanisms of HetCCL are not inherently limited to inter-node heterogeneity. The TACC abstraction and per-vendor device code compilation mechanism remain valid for heterogeneous GPUs within a node. Extending to intra-node multi-vendor environments would additionally require handling cross-vendor P2P/SHM paths, which we plan to explore as future work.

**Dynamic Load Balancing.** HetCCL currently determines micro-batch ratios through an initial offline profiling phase, which is sufficient to achieve stable and effective load balancing, as confirmed in the evaluation results. As a complementary extension, we plan to explore online re-profiling that dynamically updates micro-batch ratios based on per-batch compute times, to further improve robustness against potential performance drift caused by factors such as thermal throttling and resource contention.

## 7 Conclusion

Training infrastructure is increasingly heterogeneous, yet today's DL software ecosystem assumes a single GPU vendor, forcing costly modifications or leaving accelerators underutilized. HetCCL removes this barrier by enabling cross-vendor GPU communication while preserving existing vendor software stacks. Through RDMA-based communication and two key mechanisms—(1) runtime API abstraction and (2) kernel-level integration—HetCCL achieves near-native performance in homogeneous settings and operates efficiently in mixed-vendor environments. By enabling transparent training on heterogeneous vendor GPUs, HetCCL expands the feasible design space for ML practitioners, allowing larger batch sizes, higher training throughput, and more flexible use of available accelerators without modifying application code or compromising model correctness.

## Acknowledgment

This work was partially supported by the National Research Foundation of Korea (NRF) under Grant No. RS-2023-00222663 (Center for Optimizing Hyperscale AI Models and Platforms) and under Grant No. A400-20260031, and by the Institute for Information and Communications Technology Promotion (IITP) under Grant No. 2018-0-00581 (CUDA Programming Environment for FPGA Clusters) and No. RS-2025-02304554 (Efficient and Scalable Framework for AI Heterogeneous Cluster Systems), all funded by the Ministry of Science and ICT (MSIT) of Korea. It was also partially supported by the Korea Health Industry Development Institute (KHIDI) under Grant No. RS-2025-25454559 (Frailty Risk Assessment and Intervention Leveraging Multimodal Intelligence for Networked Deployment in Community Care), funded by the Ministry of Health and Welfare (MOHW) of Korea. Additional support was provided by the BK21 Plus Program for Innovative Data Science Talent Education (Department of Data Science, Seoul National University, No. 5199990914569) and the BK21 FOUR Program for Intelligent Computing (Department of Computer Science and Engineering, Seoul National University, No. 4199990214639), both funded by the Ministry of Education (MOE) of Korea. This work was also partially supported by the Advanced GPU Utilization Support Program, funded by the Ministry of Science and ICT (MSIT) of Korea and operated by the National IT Industry Promotion Agency (NIPA). This work was also supported in part by Samsung Research, Samsung Electronics Co., Ltd. Research facilities were provided by the Institute of Computer Technology (ICT) at Seoul National University.

## References


[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] DeepSpeed AI. Megatron-DeepSpeed, 2026. URL https://github.com/deepspeedai/Megatron-DeepSpeed.

[3] DeepSpeed AI. Megatron-DeepSpeed MoE Examples, 2026. URL https://github.com/deepspeedai/Megatron-DeepSpeed/tree/main/examples_deepspeed/MoE.

[4] Chandrish Ambati and Trung Diep. AMD MI300X GPU performance analysis. *arXiv preprint arXiv:2510.27583*, 2025.

[5] AMD. GPU-aware MPI with ROCm, 2023. URL https://rocm.blogs.amd.com/software-tools-optimization/gpu-aware-mpi/README.html.

[6] AMD. xGMI, 2025. URL https://rocm.blogs.amd.com/software-tools-optimization/mi300x-rccl-xgmi/README.html.

[7] AMD. Advancing the cloud together: How AMD and AWS are powering the next era of enterprise efficiency and innovation, 2025. URL https://www.amd.com/en/solutions/data-center/insights/advancing-the-cloud-together.html.

[8] AMD. DirectGMA, 2026. URL https://github.com/GPUOpen-LibrariesAndSDKs/DirectGMA_P2P.

[9] AMD. RCCL, 2026. URL https://rocm.docs.amd.com/projects/rccl.

[10] Chen-Chun Chen, Kawthar Shafie Khorassani, Pouya Kousha, Qinghua Zhou, Jinghan Yao, Hari Subramoni, and Dhabaleswar K Panda. MPI-xCCL: A portable MPI library over collective communication libraries for various accelerators. In *Proceedings of the SC'23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 847–854, 2023.

[11] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[12] Yifan Ding, Nicholas Botzer, and Tim Weninger. HetSeq: Distributed GPU training on heterogeneous infrastructure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 15432–15438, 2021.

[13] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. GLaM: Efficient scaling of language models with Mixture-of-Experts. In *International conference on machine learning*, pages 5547–5569. PMLR, 2022.

[14] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23 (120):1–39, 2022.

[15] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004.

[16] 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 preprint arXiv:2407.21783*, 2024.

[17] Runsheng Benson Guo, Utkarsh Anand, Arthur Chen, and Khuzaima Daudjee. Cephalo: Harnessing heterogeneous GPU clusters for training Transformer models. *arXiv preprint arXiv:2411.01075*, 2024.

[18] Red Hat. OpenShift Container Platform Hardware Accelerators, 2026. URL https://docs.redhat.com/en/documentation/openshift_container_platform/4.21/html-single/hardware_accelerators.

[19] Mert Hidayetoglu, Simon Garcia de Gonzalo, Elliott Slaughter, Pinku Surana, Wen-mei Hwu, William Gropp, and Alex Aiken. HiCCL: A hierarchical collective communication library. In *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 950–961. IEEE, 2025.

[20] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. Characterization of large language model development in the datacenter. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 709–729, 2024.

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

[22] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. GPipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[23] Intel. OneCCL, 2026. URL https://www.intel.com/content/www/us/en/developer/tools/oneapi/oneccl.html.

[24] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 642–657, 2023.

[25] Xianyan Jia, Le Jiang, Ang Wang, Wencong Xiao, Ziji Shi, Jie Zhang, Xinyuan Li, Langshi Chen, Yong Li, Zhen Zheng, et al. Whale: Efficient giant model training over heterogeneous GPUs. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 673–688, 2022.

[26] Gurkirat Kaur and Manju Bala. Rdma over converged ethernet: A review. *International Journal of Advances in Engineering & Technology*, 6(4):1890, 2013.

[27] Heehoon Kim, Junyeol Ryu, and Jaejin Lee. TCCL: Discovering better communication paths for PCIe GPU clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 999–1015, 2024.

[28] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large Transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

[29] Kubernetes. Schedule GPUs, 2026. URL https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus.

[30] Vili Lehdonvirta, Boxi Wu, Zoe Jay Hawkins, Celine Caira, and Lucia Russo. Measuring domestic public cloud compute availability for artificial intelligence. *OECD Artificial Intelligence Papers*, 2025.

[31] Dacheng Li, Hongyi Wang, Eric Xing, and Hao Zhang. AMP: Automatically finding model parallel strategies with heterogeneity awareness. *Advances in Neural Information Processing Systems*, 35:6630–6639, 2022.

[32] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, 2023.

[33] Jieke Lin, Wanyu Wang, Longxiang Yin, and Yinhe Han. KAITIAN: A unified communication framework for enabling efficient collaboration across heterogeneous accelerators in embodied AI systems. *arXiv preprint arXiv:2505.10183*, 2025.

[34] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[35] Patrick MacArthur, Qian Liu, Robert D Russell, Fabrice Mizero, Malathi Veeraraghavan, and John M Dennis. An integrated tutorial on InfiniBand, Verbs, and MPI. *IEEE Communications Surveys & Tutorials*, 19(4):2894–2926, 2017.

[36] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[37] Meta. TorchComms, 2026. URL https://github.com/meta-pytorch/torchcomms.

[38] Microsoft. Introducing the new Azure ai infrastructure VM series ND MI300X v5, 2024. URL https://techcommunity.microsoft.com/blog/azurehighperformancecomputingblog/introducing-the-new-azure-ai-infrastructure-vm-series-nd-mi300x-v5/4145152.

[39] Microsoft. MSCCL++, 2026. URL https://github.com/microsoft/mscclpp.

[40] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 481–498, 2020.

[41] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, 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*, pages 1–15, 2021.

[42] NVIDIA. An introduction to CUDA-Aware MPI, 2013. URL https://developer.nvidia.com/blog/introduction-cuda-aware-mpi.

[43] NVIDIA. GPUDirect, 2026. URL https://developer.nvidia.com/gpudirect.

[44] NVIDIA. NCCL, 2026. URL https://developer.nvidia.com/nccl.

[45] NVIDIA. NVLink, 2026. URL https://www.nvidia.com/en-us/data-center/nvlink.

[46] Oracle. Oracle and AMD collaborate to help customers deliver breakthrough performance for large-scale ai and agentic workloads, 2025. URL https://www.oracle.com/news/announcement/oracle-and-amd-collaborate-to-help-customers-deliver-breakthrough-performance-for-large-scale-ai-and-agentic-workloads-2025-06-12.

[47] Nathan Otterness and J Anderson. AMD GPUs as an alternative to NVIDIA for supporting real-time workloads. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems*, 2020.

[48] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. The MVAPICH project: Transforming research into high-performance MPI library for hpc community. *Journal of Computational Science*, 52:101208, 2021.

[49] Jay H Park, Gyeongchan Yun, M Yi Chang, Nguyen T Nguyen, Seungmin Lee, Jaesik Choi, Sam H Noh, and Young-ri Choi. HetPipe: Enabling large dnn training on (whimpy) heterogeneous GPU clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 307–321, 2020.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[51] Gregory F Pfister. An introduction to the InfiniBand architecture. *High performance mass storage and parallel I/O*, 42(617-632):102, 2001.

[52] Kevin Pulo. Fun with LD_PRELOAD. In *linux. conf. au*, volume 153, page 103, 2009.

[53] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[54] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[55] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. ZeRO-Infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–14, 2021.

[56] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506, 2020.

[57] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification, 2007.

[58] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing billion-scale model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021.

[59] 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 preprint arXiv:2504.09014*, 2025.

[60] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. UCX: an open source framework for HPC network APIs and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43. IEEE, 2015.

[61] 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. In *International Conference on Learning Representations*, 2017.

[62] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[63] 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.

[64] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatele. A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training. In *Proceedings of the 37th International Conference on Supercomputing*, pages 203–214, 2023.

[65] Ding Tang, Jiecheng Zhou, Jiakai Hu, Shengwei Li, Huihuang Zheng, Zhilin Pei, Hui Wang, and Xingcheng Zhang. H2: Towards efficient large-scale llm training on hyper-heterogeneous cluster over 1,000 chips. *arXiv preprint arXiv:2505.17548*, 2025.

[66] Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. Metis: Fast automatic distributed training on heterogeneous GPUs. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 563–578, 2024.

[67] Manjunath Gorentla Venkata, Valentine Petrov, Sergey Lebedev, Devendar Bureddy, Ferrol Aderholdt, Joshua Ladd, Gil Bloch, Mike Dubman, and Gilad Shainer. Unified collective communication (UCC): An unified library for CPU, GPU, and DPU collectives. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 37–46. IEEE, 2024.

[68] David W Walker and Jack J Dongarra. MPI: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[69] Hao Wang, Sreeram Potluri, Devendar Bureddy, Carlos Rosales, and Dhabaleswar K Panda. Gpu-aware MPI on RDMA-enabled clusters: Design, implementation and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 25(10):2595–2605, 2013.

[70] Xiaodong Yi, Shiwei Zhang, Ziyue Luo, Guoping Long, Lansong Diao, Chuan Wu, Zhen Zheng, Jun Yang, and Wei Lin. Optimizing distributed training deployment in heterogeneous GPU clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 93–107, 2020.

[71] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

[72] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.

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

[74] Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. MegaScale-Infer: Serving mixture-of-experts at scale with disaggregated expert parallelism. *arXiv preprint arXiv:2504.02263*, 2025.

## Appendix

The appendix is organized as follows:

1. Appendix A discusses extended related work.
2. Appendix B includes the runtime API functions implemented in TACC.
3. Appendix C provides experimental details.
4. Appendix D includes additional experimental results.
5. Appendix E provides an ablation study.
6. Appendix F includes the impact statement.

We publicly release the code at: `https://github.com/mcrl/hetccl`.

# A Extended Related Work

Our work relates broadly to various research areas in distributed training with heterogeneous GPUs. In this section, we discuss the communication patterns arising from parallelization in DL (Appendix A.1), and the initiatives aimed at enhancing the performance of DL model training or inference in heterogeneous GPU environments (Appendix A.2).

## A.1 Parallelism and Communication Patterns in DL

**Parallelism in DL.** The rapid growth of DL, particularly LLMs, has driven an unprecedented increase in model scale—from hundreds of billions to trillions of parameters. To meet the corresponding rise in computational and memory demands, state-of-the-art DL frameworks [28, 41, 54, 55, 58, 62] now rely heavily on multi-GPU execution through various forms of parallelism, including data, tensor, pipeline, and expert parallelism [22, 54, 61, 62]. These parallelization strategies inevitably introduce intensive communication between GPUs. Each parallelization strategy requires specific types of collective communication operations.

**Communication patterns in DL.** ***Fully sharded data parallelism (FSDP)*** [54] partitions model parameters across GPUs and performs training by gathering all parameters during the forward pass using All-Gather, and distributing gradients during the backward pass via Reduce-Scatter collective communication. ***Tensor parallelism*** [62] splits individual weight matrices across GPUs. It computes partial results in parallel, which are then aggregated using All-Reduce collective communication to obtain the final result. ***Pipeline parallelism*** [22] involves sequential point-to-point transfers between pipeline stages. ***Expert parallelism*** [14, 32, 61] for Mixture-of-Experts (MoE) architectures demands complex All-to-All collective communication across GPU devices. During inference, ***Prefill-decode disaggregation*** [72] requires low-latency point-to-point communication to minimize user-perceived latency.

As models grow larger and more complex, the demand for GPUs in both training and inference continues to rise. To meet this scaling pressure, many organizations expand their clusters incrementally, often ending up with heterogeneous GPU environments that mix vendors such as NVIDIA and AMD. This trend, however, poses a critical challenge: enabling efficient communication across GPUs from different vendors. Traditional communication libraries and hardware interconnects are optimized for homogeneous setups and cannot support or exploit cross-vendor communication.

## A.2 DL on Heterogeneous GPUs

Modern GPU clusters typically consist of devices with different compute capabilities, memory sizes, and interconnect bandwidths due to upgrades over time or hardware reuse. Effectively utilizing these heterogeneous resources for DL model training has become a significant focus of research. Many existing approaches explore various combinations of parallelism and resource allocation strategies to optimize the training of DL models on these diverse GPUs. These efforts can be broadly categorized into two groups: (1) automatic parallelization, which plans how a single job runs in parallel across different devices, and (2) cluster-level schedulers, which determine how jobs are assigned to various hardware resources over time.

Table 1: Heterogeneity type and evaluation scale of representative related systems.

| System | Vendor Setting | Max #GPUs | GPU Configuration |
|---|---|---|---|
| HeteroG [70] | Single-vendor | 12 | NVIDIA V100, P100, GTX 1080Ti |
| HetSeq [12] | Single-vendor | 16 | NVIDIA Titan Xp, GTX 1080Ti, P100 |
| HetPipe [49] | Single-vendor | 16 | NVIDIA TITAN V, TITAN RTX, RTX 2060, Quadro P4000 |
| AMP [31] | Single-vendor | 16 | NVIDIA V100, T4 |
| Metis [66] | Single-vendor | 32 | NVIDIA P100, T4, V100 |
| HetCCL | Cross-vendor | 16 | NVIDIA V100, AMD Radeon Pro W7800 |

**Automatic parallelization on heterogeneous GPUs.** Several systems are designed to automatically determine parallelism strategies that take into account hardware heterogeneity within a job. HeteroG [70] proposes a Graph Neural Network (GNN)-based optimization framework that selects

hybrid parallelism strategies (i.e., data and model parallelism) along with communication methods for heterogeneous GPUs in a cluster. HetSeq [12] enables parallel training across GPUs with varying performance levels by allowing different batch sizes for each device and synchronizing gradients through weighted averaging, although it focuses exclusively on data parallelism. HetPipe [49] combines pipeline parallelism among virtual workers with data parallelism across them, introducing a synchronization model to ensure convergence even when using lower-tier GPUs. AMP [31] formulates a search space for degrees of parallelism and pipeline assignments, utilizing a heterogeneity-aware cost model to identify efficient strategies through dynamic programming. Metis [66] expands on this approach by incorporating device grouping and asymmetric workload balancing. It employs a heterogeneity-aware search algorithm that prunes inefficient configurations and adjusts intra-stage parallelism based on device capacity.

Table 1 summarizes the types of heterogeneity and evaluation scales for these systems. Consistent with these prior heterogeneous GPU systems, HetCCL targets representative mid-scale clusters formed through incremental GPU integration, while further addressing cross-vendor interoperability between NVIDIA and AMD GPUs. However, these previous studies focus on parallelization strategies for heterogeneous GPUs, whereas we concentrate on implementing a collective communication library for heterogeneous GPUs.

**Heterogeneity-aware cluster scheduling.** In addition to efforts focused on parallelization, another area of research involves job-level scheduling for heterogeneous GPUs in a cluster. Gavel [40] schedules jobs based on throughput profiling across different GPU types, aiming to optimize job completion times while adhering to fairness constraints. However, it does not allow for the splitting of a job across devices of different types. Sia [24] supports elastic and adaptive DL jobs by dynamically scaling their resource usage over time. It also considers heterogeneity during resource allocation, but still limits each job to a single type of device.

```
1  taccGetAvailablePlatforms();
2  taccSetPlatform();
3  taccSetPlatformAuto();
4  taccGetPlatform();
5  taccThreadExchangeStreamCM();
6  taccMalloc();
7  taccMallocHost();
8  taccMallocManaged();
9  taccHostAlloc();
10 taccGetDevice();
11 taccSetDevice();
12 taccGetErrorString();
13 taccGetLastError();
14 taccExtMallocWithFlags();
15 taccDeviceCanAccessPeer();
16 taccFreeHost();
17 taccDeviceGetAttribute();
18 taccStreamCreateWithFlags();
19 taccMemcpy();
20 taccMemcpyAsync();
21 taccFree();
22 taccMemset();
23 taccMemsetAsync();
24 taccStreamQuery();
25 taccStreamSynchronize();
26 taccStreamDestroy();
27 taccEventCreate();
28 taccEventCreateWithFlags();
29 taccEventQuery();
30 taccEventRecord();
31 taccStreamWaitEvent();
32 taccEventDestroy();
33 taccLaunchHostFunc();
34 taccIpcOpenMemHandle();
35 taccIpcCloseMemHandle();
36 taccIpcGetMemHandle();
37 taccHostRegister();
38 taccHostUnregister();
39 taccHostGetDevicePointer();
40 taccDeviceGetPCIBusId();
41 taccGetDeviceProperties();
42 taccDeviceGetByPCIBusId();
43 taccDeviceEnablePeerAccess();
44 taccGetDeviceCount();
45 taccFuncSetAttribute();
46 taccFuncGetAttributes();
47 taccExtLaunchKernel();
48 taccLaunchKernel();
49 taccDeviceSetLimit();
50 taccStreamGetCaptureInfo_v2();
51 taccGraphAddHostNode();
52 taccGraphInstantiate();
53 taccGraphLaunch();
54 taccGraphExecDestroy();
55 taccGraphDestroy();
56 taccDeviceSynchronize();
57 taccStreamBeginCapture();
58 taccStreamEndCapture();
59 taccDeviceReset();
60 taccPointerGetAttributes();
61 taccExtStreamCreateWithCUMask();
62 taccStreamCreate();
63 taccExtGetLinkTypeAndHopCount();
64 taccHostMalloc();
65 taccHostFree();
66 taccEventElapsedTime();
```

Figure 10: TACC API functions.

While these systems acknowledge GPU heterogeneity in terms of architectural generations or performance variability, they generally operate under the assumption that all GPUs come from a single vendor. This is due to the lack of support for collective communication across different vendors. Consequently, the heterogeneity they address is somewhat limited, which helps keep system complexity manageable. Enabling true heterogeneity—where GPUs from different vendors can participate in a single collective communication operation—could open up new research avenues in both parallelization and scheduling.

# B TACC API functions

Figure 10 shows the full list of functions implemented in HetCCL's runtime API abstraction layer, TACC. TACC abstracts the CUDA and HIP runtime APIs used in NCCL and RCCL.

# C Experimental Details

This section describes the experimental details, including the system configuration of our four-node cluster (Appendix C.1), details of DL models used in our experiments (Appendix C.2), implementation details of DL frameworks for LLM training (Appendix C.3), and the communication patterns that arise when training with DeepSpeed ZeRO (Appendix C.4).

## C.1 System Configuration

Table 2: Hardware configuration of the four-node multi-vendor GPU cluster.

| | **NVIDIA (2 nodes)** | **AMD (2 nodes)** |
|---|---|---|
| Mainboard | GIGABYTE MZ52-G40 | ASRock ROMED8-2T |
| CPU | 1 × AMD EPYC 7452 32-Core | 2 × AMD EPYC 7313 16-Core |
| Memory | 8 × DDR4-2666 64GB | 4 × DDR4-3200 16GB |
| GPU | 4 × NVIDIA Tesla V100-PCIe | 4 × AMD Radeon Pro W7800 |
| NIC | Mellanox ConnectX-6 InfiniBand HDR | Mellanox ConnectX-6 InfiniBand HDR |
| GPU PCIe | Gen3 | Gen4 |
| GPU Driver | 550.54.15 (CUDA 12.4) | 6.12.12 (ROCm 6.4.0) |

To comprehensively evaluate HetCCL, we configure a heterogeneous GPU cluster comprising multiple NVIDIA and AMD GPU nodes connected with a uniform InfiniBand interconnect. All nodes use the same InfiniBand NIC and fabric. This setup enables us to compare the results of cross-vendor GPU communication with those from homogeneous GPU environments, where GPUs are housed within the same vendor's nodes. The detailed system configurations are described in Table 2.

Even though the experiments were carried out on a PCIe cluster without high-speed GPU interconnects like NVLink [45] or xGMI [6], the proposed method and findings are still relevant to more advanced environments. This is because vendor libraries like NCCL and RCCL are designed to automatically leverage available hardware features when present. HetCCL is built on top of these libraries and naturally inherits such functionality. To demonstrate that HetCCL remains effective even in the presence of high-bandwidth interconnects, we include an evaluation on a single-node high-end GPU system with high-bandwidth interconnects in Appendix D.6.

## C.2 Models

Table 3: Specifications of models used in the experiments.

| **Model** | **Parameter size** | **Sequence length** | **Vocab size** |
|---|---|---|---|
| GPT | 125M<br>355M | 1024 | 50257 |
| LLaMA | 1B<br>3B | 8192 | 32000 |

Table 3 shows the details of the DL models used in Section 5.3. We follow the remaining hyperparameter configurations (e.g., number of layers, hidden size, attention heads) from the original papers [1, 16].

For each model, we choose the maximum feasible batch size before GPU out-of-memory (OOM) errors. Experiments on larger-scale or MoE model training are in Appendix D.4.

### C.3 Implementation Details

**Tunable Parameters in CCLs.** NCCL and RCCL hard-code a number of tunable parameters whose defaults are not optimal on every system, so some adjustment is expected when porting to new hardware. Two parameters are not aligned between the two: HetCCL (AMD) is built with `NCHANNELS_PER_NET_PEER` raised from 1 to 2 and `MAX_NTHREADS` from 256 to 512, because RCCL's defaults left significant bandwidth unused on our cluster, whereas we ran RCCL as released in order to reproduce it faithfully. However, we did not intend this as a performance claim as written in Section 5.2, and recompiling RCCL with the same two constants closes the gap entirely. All other tunables are aligned across the three implementations: `NCCL_BUFFSIZE`, `NCCL_MIN/MAX_NCHANNELS`, `NCCL_STEPS`, and all `NCCL_IB_*` settings, with `NCCL_ALGO=RING` applied uniformly to every run.

**Training DL Models.** We use PyTorch [50] as our DL framework and employ the AdamW [34] optimizer, incorporating activation checkpointing [11] to reduce memory usage. To maximize computational efficiency and reduce memory footprint, we use FP16 precision for all end-to-end performance evaluations in Section 5.3. For model convergence and numerical consistency analysis, however, we follow the training precision used in the original work [16] to ensure faithful reproduction of model convergence behavior and therefore conduct training using BF16. For parallelization, we adopt DeepSpeed [56], a widely recognized DL optimization library developed by Microsoft. DeepSpeed is designed to be hardware-agnostic and currently supports accelerators from multiple vendors, including Intel, Huawei, NVIDIA, and AMD, which makes it a natural choice for heterogeneous multi-GPU training. We intentionally disable all forms of CPU or NVMe offloading [55, 58] in DeepSpeed to avoid confounding effects from host memory capacity differences in our cluster.

### C.4 Communication Patterns under Different ZeRO Stages

The ZeRO [54] family of optimizations in DeepSpeed [56] includes several stages, each based on different components being partitioned. As a result, different ZeRO stages involve various collective communication operations, as shown in Table 4. We use both ZeRO-1 and ZeRO-3 in our experiments to evaluate various collective communication patterns. ZeRO-1 primarily employs All-Gather for optimizer states and All-Reduce for gradients, while ZeRO-3 utilizes All-Gather for parameters and Reduce-Scatter for gradients.

Table 4: Communication patterns across DeepSpeed ZeRO stages.

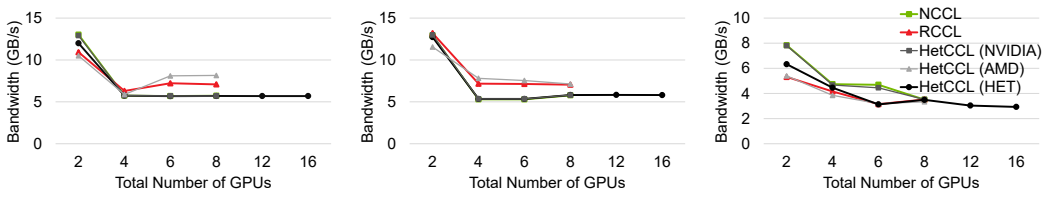
| **ZeRO Stage** | **Stage 1** | **Stage 2** | **Stage 3 (FSDP)** |
|---|---|---|---|
| Optimizer State (OS) | Sharded | Sharded | Sharded |
| Gradient (G) | Replicated | Sharded | Sharded |
| Parameter (P) | Replicated | Replicated | Sharded |
| Communication Ops | All-Gather (OS), All-Reduce (G) | All-Gather (P), Reduce-Scatter (G) | All-Gather (P), Reduce-Scatter (G) |

## D Additional Experimental Results

This section provides additional experimental results, including the performance of other collective communication operations (Appendix D.1), loss convergence of LLM training (Appendix D.2), a comparison with different vendor library versions (Appendix D.3), model training with hybrid and expert parallelism (Appendix D.4), a performance comparison against GPU-aware MPI (Appendix D.5), a performance evaluation on high-end GPU systems (Appendix D.6), and a run-to-run statistical variability (Appendix D.7).

### D.1 Other Collective Communication Operations

We provide the performance of Reduce, Broadcast, and All-to-All collective communication operations in Figure 11. The behavior is consistent across various collective patterns as discussed in

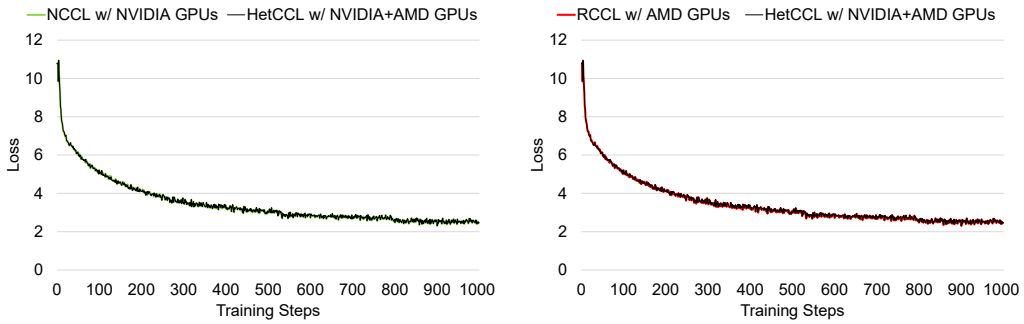



(a) Reduce (b) Broadcast (c) All-to-All

Figure 11: Performance of other representative collective communication operations.

Section 5.2, reflecting HetCCL's RDMA-based communication and internal strategy of delegating communication to vendor libraries: NCCL for NVIDIA GPUs and RCCL for AMD GPUs.

### D.2 LLM Training Convergence




(a) NCCL vs. HetCCL (b) RCCL vs. HetCCL

Figure 12: Training loss convergence of LLaMA-1B on WikiText-103.

We evaluate training convergence to verify that heterogeneous multi-vendor GPU training with HetCCL does not introduce model accuracy degradation or numerical instability. Figure 12 compares the training loss curves of LLaMA-1B across homogeneous and heterogeneous configurations. We observe no noticeable divergence in convergence behavior or final loss values when using HetCCL compared to vendor-specific baselines (i.e., NCCL or RCCL). Table 5 further shows that HetCCL achieves validation and test perplexity comparable to NCCL and RCCL, with only small relative differences (within 1.0%) from NCCL.

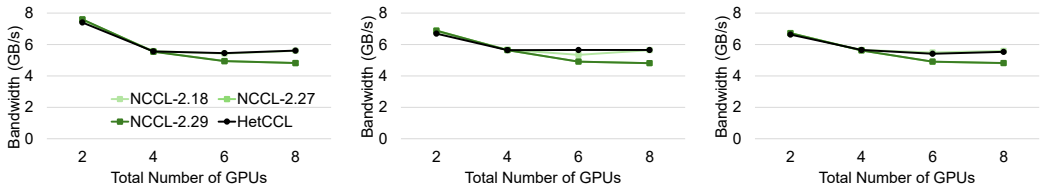
Table 5: Validation and test perplexity of LLaMA-1B on WikiText-103.

| CCL (GPU Vendor) | Validation PPL | Test PPL |
|---|---|---|
| NCCL (NVIDIA GPUs) | 15.70 | 16.30 |
| RCCL (AMD GPUs) | 15.54 | 16.15 |
| HetCCL (NVIDIA + AMD GPUs) | 15.77 | 16.47 |

This is because the implementation and execution behaviors of DL frameworks are consistent across NVIDIA and AMD systems. In addition, NVIDIA CUDA and AMD ROCm kernels exhibit numerically close behavior. Furthermore, NCCL and RCCL collectives are consistent within expected floating-point tolerances. Therefore, despite differences in floating-point implementations and reduction ordering across vendors, under deterministic settings (i.e., identical random seeds and initial weights without non-deterministic operations such as dropout), LLM training on hybrid NVIDIA–AMD systems exhibits stable behavior consistent with homogeneous training.

### D.3 Comparison between Different Versions of Vendor Libraries

We evaluate whether the choice of vendor library version affects the validity of our baseline comparison. In the main evaluation, we use NCCL version 2.18.3 and the corresponding RCCL version 6.0.2




(a) All-Reduce (b) All-Gather (c) Reduce-Scatter

Figure 13: Collective communication performance across different NCCL versions.

as vendor-optimized baselines, since HetCCL has been implemented on top of these versions. To confirm that this does not underestimate recent versions of vendor libraries, we compare HetCCL against newer NCCL releases, including NCCL v2.27.7 and v2.29.7, for both point-to-point and collective communication.

As shown in Figure 13 and Figure 14, the performance differences across NCCL versions are nearly negligible overall. This is consistent with the general development pattern of NCCL, where new releases primarily add support for newer hardware generations (e.g., NVSwitch, Blackwell architectures) and bug fixes, rather than substantially altering performance on existing hardware. We observe a similar trend for RCCL, which is expected because RCCL is maintained as a fork of NCCL and incorporates updates from every NCCL release into its codebase. In some cases, especially for larger message sizes or larger GPU counts, newer NCCL versions are slightly slower than NCCL v2.18.3. We note that the goal of this work is not to analyze version-specific performance differences within NCCL or RCCL. Rather, our focus is to demonstrate that HetCCL can provide cross-vendor communication while preserving near-native performance on homogeneous platforms. We also note that since HetCCL delegates vendor-local communication to NCCL and RCCL through its runtime abstraction layer, newer NCCL and RCCL releases can be integrated with minimal changes.

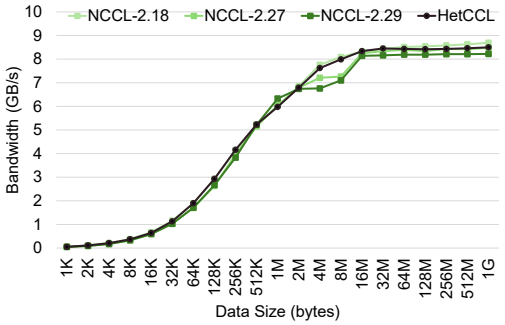



Figure 14: Point-to-point communication performance across different NCCL versions.

### D.4 Training Models with Hybrid and Expert Parallelism

We also evaluate HetCCL on larger-scale models with hybrid parallelism and sparse MoE models with expert parallelism. While the main experiments in Section 5.3 focus on data-parallel-only training to isolate HetCCL's core communication efficiency, this setting limits the maximum model size by per-GPU memory capacity. To evaluate larger models, we use Megatron-DeepSpeed [2], combining ZeRO-based data parallelism [54] with Megatron tensor parallelism [62].

Table 6: LLaMA 7B and 13B model training with Megatron-DeepSpeed.

| CCL (# of GPUs) | LLaMA 7B | | LLaMA 13B | |
|---|---|---|---|---|
| | Throughput (tokens/s) | Speedup | Throughput (tokens/s) | Speedup |
| RCCL (8xAMD) | 691 | 1.00 | 386 | 1.00 |
| NCCL (8xNVIDIA) | 1289 | 1.87 | 716 | 1.85 |
| HetCCL (4xAMD + 4xNVIDIA) | 932 | 1.35 | 537 | 1.39 |
| HetCCL (8xAMD + 8xNVIDIA) | 1806 | 2.61 | 1021 | 2.64 |

For LLaMA 7B and 13B training, we use hybrid parallelism with TP degree 4 and ZeRO-3, where the DP degree is 2 for 8-GPU runs and 4 for 16-GPU runs. Table 6 reports the training throughput of LLaMA 7B/13B models with the same configuration (e.g., sequence length, vocabulary size) as in Table 3. Even at these larger scales, HetCCL maintains high efficiency, achieving 91% efficiency for LLaMA 7B and 93% efficiency for LLaMA 13B. These results show that HetCCL is not limited to

the model sizes evaluated in the main paper and can scale to substantially larger dense LLMs when combined with standard parallel training techniques.

Table 7: Sparse MoE model training with expert parallelism.

| CCL (# of GPUs) | Throughput (tokens/s) | Speedup |
|---|---|---|
| RCCL (8xAMD) | 2792 | 1.00 |
| NCCL (8xNVIDIA) | 4287 | 1.54 |
| HetCCL (4xAMD + 4xNVIDIA) | 3435 | 1.23 |
| HetCCL (8xAMD + 8xNVIDIA) | 6441 | 2.31 |

To train sparse MoE models, we use the MoE example script [3] provided by Megatron-DeepSpeed, which implements GPT-style MoE models with expert parallelism. As shown in Table 7, HetCCL achieves 91% efficiency on GPT-3 XL MoE with 8 experts and 4.1B total parameters. This result further demonstrates that HetCCL is compatible not only with dense LLM training but also with sparse expert-parallel workloads.

We intentionally exclude extremely large models such as LLaMA 70B from this evaluation because they require CPU or NVMe offloading mechanisms [54, 58]. Although technically feasible, such offloading introduces additional system-level factors, making it difficult to isolate the contribution of HetCCL's cross-vendor communication.

### D.5 Performance Comparison against GPU-Aware MPI

Modern MPI implementations support GPU-aware communication [69], allowing GPU device buffers to be passed directly to MPI primitives without explicit host staging. In practice, GPU-aware MPI support is provided through vendor-specific backends, such as CUDA-aware MPI [42] for NVIDIA GPUs and ROCm-aware MPI [5] for AMD GPUs, enabling efficient communication on each respective platform [10, 48].

OpenMPI uses the same host-side MPI library on NVIDIA and AMD nodes and loads the appropriate accelerator plugin on each node. Our benchmark application allocates GPU buffers through CUDA or HIP, so we compile it separately for each vendor. We launch both executables within a single MPI job, mapping ranks to the corresponding nodes. MPI itself is unmodified, and all communication uses standard GPU-aware MPI interfaces.

In this evaluation, we implement heterogeneous GPU-aware MPI using OpenMPI (v5.0.2) [15] with UCX (v1.19.0) [60], one of the most widely used MPI stacks for accelerator-aware communication. We compare the performance of HetCCL against this baseline in both point-to-point and collective communication scenarios to assess the efficiency of the proposed design under heterogeneous GPU configurations. Figure 15 presents the performance of cross-vendor point-to-point communication between NVIDIA and AMD GPUs, while Figure 16 reports the performance of All-Reduce and All-Gather collective operations on 16 GPUs spanning four nodes, where two nodes are equipped with NVIDIA GPUs and the remaining two nodes with AMD GPUs.

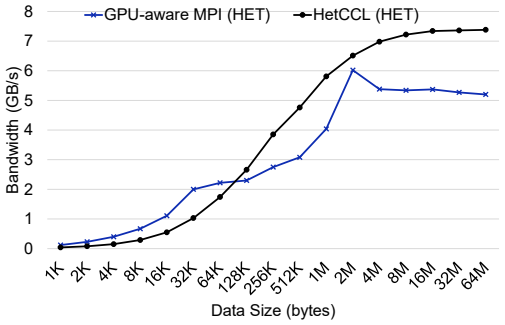
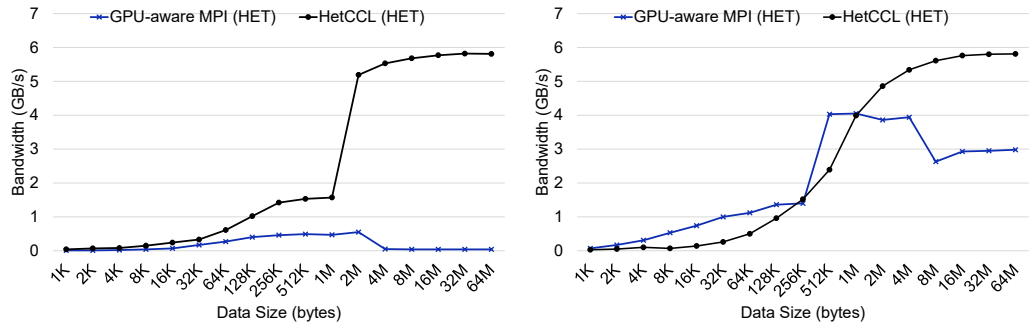



Figure 15: Point-to-point communication performance: GPU-aware MPI vs. HetCCL.

The results show that, consistent with prior studies [10, 33], MPI achieves lower latency for small message sizes, whereas HetCCL delivers higher effective bandwidth for larger messages, making it more suitable for large-scale DL workloads dominated by large tensor transfers. This trend is especially evident in collective operations. Note that HetCCL outperforms GPU-aware MPI in All-Reduce across most of the evaluated message sizes, as MPI implementations typically do not natively provide GPU kernels and often incur additional overhead from host device staging for CPU-based reduction operations. In contrast, HetCCL performs reductions entirely on the GPU, better exploiting the memory hierarchy and interconnect bandwidth of modern accelerators. Motivated by

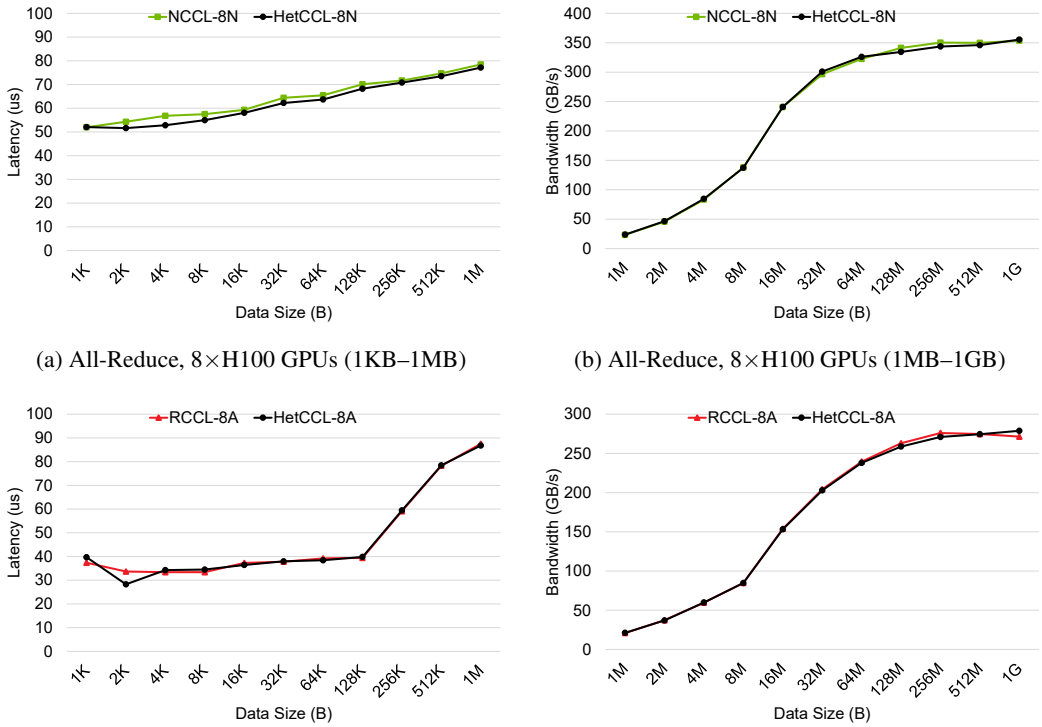



(a) All-Reduce (b) All-Gather

Figure 16: Collective communication performance on 16 GPUs: GPU-aware MPI vs. HetCCL.

these limitations, recent studies [67] have explored the integration of optimized GPU kernels directly into MPI implementations to improve collective performance on accelerator-based systems. For end-to-end LLaMA-1B training on 16 GPUs with ZeRO-1 and a sequence length of 2048, HetCCL achieves an $11.4\times$ speedup over GPU-aware MPI, primarily due to lower communication overhead.

### D.6 Performance on High-End GPU System with High-Bandwidth Interconnect




(a) All-Reduce, 8×H100 GPUs (1KB–1MB) (b) All-Reduce, 8×H100 GPUs (1MB–1GB)

(c) All-Reduce, 8×MI300X GPUs (1KB–1MB) (d) All-Reduce, 8×MI300X GPUs (1MB–1GB)

Figure 17: Single node All-Reduce performance on NVIDIA and AMD high-end GPU systems.

Due to the limited availability of a multi-vendor cluster environment equipped with the latest generation of accelerators, we indirectly validate the performance and extensibility of HetCCL through single-node experiments on homogeneous high-end GPU systems: an 8×NVIDIA H100 system and an 8×AMD MI300X system. Unlike our primary multi-vendor cluster (Table 2), which relies on PCIe-based communication, these systems are equipped with high-bandwidth GPU interconnects, namely NVIDIA NVLink [45] and AMD xGMI [6], providing significantly higher intra-node throughput.

Note that even without a dedicated GPU-to-GPU interconnect (e.g., NVLink, xGMI), GPUs can still perform peer-to-peer (P2P) transfers over PCIe in our system setup.

We conduct single-node experiments to verify whether HetCCL introduces any performance overhead when leveraging vendor-specific high-speed interconnects. We compare HetCCL against NCCL on the H100 system and RCCL on the MI300X system using the All-Reduce collective operation with message sizes ranging from 1KB to 1GB. We separate the range of message sizes into small (1KB–1MB, presented in latency) and large (1MB–1GB, presented in bandwidth) for visibility.

As shown in Figure 17, HetCCL matches the performance of vendor-optimized libraries on both platforms across all message sizes. While these results are obtained in homogeneous settings, they serve as a critical proxy for heterogeneous performance; by proving that HetCCL does not degrade the performance of the fastest available intra-node interconnects, we demonstrate that the library's abstraction layer is lightweight enough to support large-scale heterogeneous clusters without becoming a performance bottleneck.

### D.7 Statistical Variability

We repeat the collective communication configurations reported in Table 8 using three independent job launches and report the mean $\pm$ standard deviation. These measurements use a 1 GB message size, at which bandwidth has saturated.

Table 8: Run-to-run variability in collective communication bandwidth across two nodes. Values are mean $\pm$ standard deviation over three independent runs, in GB/s.

| Operation | NCCL | RCCL | HetCCL (NVIDIA) | HetCCL (AMD) | HetCCL (HET) |
|---|---|---|---|---|---|
| All-Reduce | $5.80 \pm 0.02$ | $7.17 \pm 0.02$ | $5.80 \pm 0.02$ | $7.85 \pm 0.03$ | $5.81 \pm 0.02$ |
| All-Gather | $5.83 \pm 0.01$ | $7.37 \pm 0.02$ | $5.82 \pm 0.03$ | $7.55 \pm 0.04$ | $5.84 \pm 0.02$ |
| Reduce-Scatter | $5.85 \pm 0.02$ | $7.35 \pm 0.03$ | $5.84 \pm 0.02$ | $7.91 \pm 0.03$ | $5.84 \pm 0.02$ |

Table 9: Run-to-run variability in end-to-end training throughput for LLaMA-1B with ZeRO-3 and FP16. Values are mean $\pm$ standard deviation over three independent runs.

| Configuration | Throughput (tokens/s) |
|---|---|
| RCCL (8×AMD) | $10{,}718 \pm 141$ |
| NCCL (8×NVIDIA) | $20{,}889 \pm 20$ |
| HetCCL (8×AMD + 8×NVIDIA) | $30{,}640 \pm 229$ |

Table 9 reports three independent runs for the LLaMA-1B configuration with ZeRO-3 and FP16 from Figure 9. We discard the first 20 warm-up steps and average throughput over the steady-state window in each run. Across both tables, the coefficient of variation is at most 1.31%. The results show small run-to-run variation and preserve the performance trends reported in Sections 5.2 and 5.3.

## E Ablation study

In this section, we provide an ablation study, including the effect of the RDMA mechanism (Appendix E.1), and the effect of the GPU-aware load balancing technique used in HetCCL (Appendix E.2).

### E.1 Effect of RDMA

We include an ablation that isolates the contribution of the GPUDirect RDMA mechanism [8, 43] on point-to-point communication performance. The non-RDMA baseline relies on host-staged communication, where data is explicitly copied from GPU memory to CPU memory before being transferred by the NIC, and then copied back to the destination GPU. This results in a data path of `GPU` $\rightarrow$ `CPU` $\rightarrow$ `NIC` $\rightarrow$ `CPU` $\rightarrow$ `GPU`. In contrast, GPUDirect RDMA enables direct GPU-to-NIC and NIC-to-GPU transfers, resulting in a path of `GPU` $\rightarrow$ `NIC` $\rightarrow$ `GPU`, eliminating host involvement and achieving higher bandwidth, as shown in Figure 18.

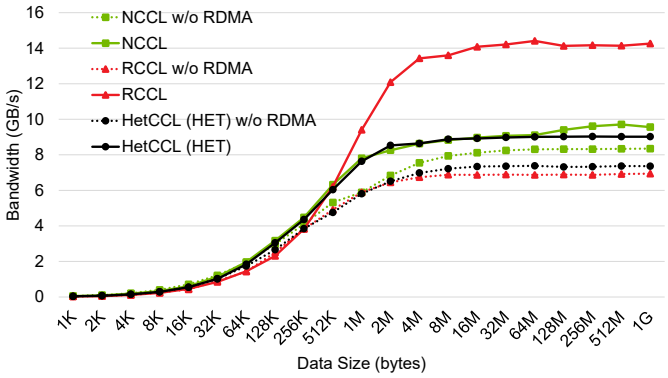



Figure 18: Performance impact of RDMA on point-to-point GPU communication.

### E.2 Effect of GPU-Aware Workload Balancing

We present an ablation study on the GPU-aware workload balancing technique implemented in HetCCL for efficient training with heterogeneous multi-vendor GPUs. HetCCL assigns a larger micro-batch size to faster GPUs to mitigate load imbalance across devices. Specifically, NVIDIA GPUs are assigned approximately twice the micro-batch size of AMD GPUs, reflecting the approximately 2× higher profiled training throughput of NVIDIA V100 GPUs compared to AMD W7800 GPUs in our cluster. Although the theoretical peak FP16 performance is 112 TFLOPS for the NVIDIA V100 and 90.5 TFLOPS for the AMD W7800, we observe substantially lower effective hardware utilization on the AMD GPUs during end-to-end LLM training. This disparity is primarily attributed to AMD's relatively low maturity of the software stack and kernel-level optimizations, a phenomenon consistent with recent performance analysis of AMD's software ecosystem [4].

Consequently, HetCCL prioritizes empirical profiling over naive theoretical metrics to ensure robust workload distribution in heterogeneous environments. The profiling run executes a small number of warm-up training iterations using the same model configuration and parallelization setup as the main training run, and measures the achieved throughput in tokens per second.

Table 10: Performance impact of load balancing on LLM training.

| Model | End-to-end speedup | Profiling overhead (s) |
|---|---|---|
| GPT 125M | 1.22× | 1.4 |
| GPT 355M | 1.19× | 2.9 |
| LLaMA 1B | 1.09× | 26.6 |
| LLaMA 3B | 1.08× | 80.2 |

The experimental results in Table 10 compare uniform micro-batch assignment with HetCCL's GPU-aware workload balancing, where faster GPUs are assigned more work. Given the profiling-derived micro-batch assignment ratio of approximately 1:2, an ideal speedup of 1.5× might be expected. However, the observed throughput improvements are consistently lower than this upper bound. This gap primarily stems from the communication fraction in end-to-end training, which limits the extent to which compute-side load balancing can improve overall throughput.

Table 11: End-to-end training latency breakdown across models.

| Model | Communication (%) | Computation (%) | Synchronization (%) |
|---|---|---|---|
| GPT 125M | 13.26 | 86.72 | 0.02 |
| GPT 355M | 16.96 | 83.03 | 0.01 |
| LLaMA 1B | 22.85 | 77.14 | 0.01 |
| LLaMA 3B | 29.09 | 70.91 | 0.00 |

To better understand this behavior, we break down the end-to-end training latency into communication, computation, and explicit synchronization. Since communication and computation can overlap in practice, we measure the cumulative duration of communication kernels and synchronization

primitives and derive the computation portion by subtracting both from the total end-to-end training time. As shown in Table 11, while computation remains the dominant component, communication already constitutes a substantial fraction of the training step and becomes increasingly important as model size grows. In particular, the communication portion increases from 13.3% for GPT-125M to 29.1% for LLaMA-3B. This indicates that end-to-end throughput is increasingly constrained by communication rather than pure computation, which reduces the achievable computational performance gain from GPU-aware workload balancing. The effect is especially pronounced for larger models, where larger gradients and heavier collective communication lead to diminishing speedups.

All experiments in this section use ZeRO stage 3, which represents the most communication-intensive configuration and therefore serves as a conservative evaluation setting. Although the main experiments were evaluated under data parallelism, this rank-level micro-batch adjustment mechanism is orthogonal to the parallelism strategy and can be applied to other forms of parallelism.

Table 10 also reports the overhead of a short profiling job in HetCCL required to determine appropriate workload ratios. Even for the largest model (LLaMA-3B), the profiling cost is trivial (less than two minutes) and is amortized over long training runs consisting of thousands of training steps. We intentionally adopt a lightweight workload balancing strategy to minimize system complexity and overhead. While more sophisticated balancing approaches [17] are possible, the results show that our profiling-based load balancing approach is sufficient to eliminate major straggler effects in practice.

# F Impact Statement

By enabling communication across heterogeneous multi-vendor GPUs, our work aims to improve resource utilization and reduce the cost of DL model training. We believe this can lower the barrier to training LLMs in diverse computing environments. We do not identify any direct negative societal impacts specific to this work beyond the general risks associated with enabling more efficient training.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the key contributions, proposed mechanisms, and main experimental findings.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of this work and future directions in Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe the system configuration, experimental setup, implementation details, and evaluated workloads in Section 5 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use public models and datasets. We provide the code with detailed README files in the supplementary material and plan to open-source it upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the experimental setup, model configurations, software stack, and parallelization details in Section 5 and Appendix C.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Appendix D.7 reports the mean and standard deviation across three independent job launches for collective communication benchmarks and a representative end-to-end training configuration, together with the measurement protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the hardware and software resources used for the experiments in Section 5, Appendix C, and supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We fully follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: See Appendix F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cite the existing software, datasets, and models used in the paper, and follow their licenses and terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the code with detailed README files in the supplementary material and plan to open-source it upon publication.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.