HeteroSim: Towards High-Fidelity Heterogeneous LLM Training Simulation on GPUs

Xiaofei Yue*
School of Software Technology,
Zhejiang University, Ningbo, China
Beijing Institute of Technology
Beijing, China
xfyue1203@gmail.com

Fangming Zhao*
School of Software Technology,
Zhejiang University
Ningbo, China
zfm@zju.edu.cn

Fulun Ye
School of Software Technology,
Zhejiang University
Ningbo, China
ashenye@zju.edu.cn

Jiongchi Yu
Singapore Management University
Singapore
jcyu.2022@phdcs.smu.edu.sg

Zhaoxuan Li
Institute of Information Engineering,
CAS, Beijing, China
lizhaoxuan@iie.ac.cn

Tingting Li†
School of Software Technology,
Zhejiang University, Ningbo, China
litt2020@zju.edu.cn

Ziming Zhao†
School of Software Technology,
Zhejiang University, Ningbo, China
zhaoziming@zju.edu.cn

Jianwei Yin†
School of Software Technology,
Zhejiang University, Ningbo, China
zjuyjw@cs.zju.edu.cn

Abstract

Modern Large Language Model (LLM) training clusters increasingly mix heterogeneous GPUs, diverse intra-node fabrics, and inter-node interconnects, combined with varied parallelism strategies. Exploring this massive design space, further amplified by heterogeneity, through real deployments is prohibitively slow and costly. Existing simulators, which are primarily designed and tuned for homogeneous clusters, either trade fidelity for speed or require heavyweight workflows with non-negligible overhead. We propose HeteroSim, a high-fidelity simulation framework for heterogeneous LLM training systems. It introduces: (i) a LLM training workload compiler that captures realistic training graphs, microbatching schedules, and compute-communication overlap; (ii) a heterogeneity-aware computation planner using roofline-style scaling across GPU generations; (iii) a collective communication planner that reproduces NCCL-like behaviors with per-link models, message channelization, and configurable routing. Across a wide range of heterogeneity levels, experimental results show that HeteroSim achieves near-real simulation accuracy while keeping low overhead.

CCS Concepts

• **Computer systems organization → Parallel architectures**; **Heterogeneous (hybrid) systems**.

Keywords

LLM Training Simulation, Heterogeneous GPUs, NCCL

*Equal contribution.
†Corresponding authors.

ACM Reference Format:
Xiaofei Yue, Fangming Zhao, Fulun Ye, Jiongchi Yu, Zhaoxuan Li, Tingting Li, Ziming Zhao, and Jianwei Yin. 2026. HeteroSim: Towards High-Fidelity Heterogeneous LLM Training Simulation on GPUs. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3774904.3792254

1 Introduction

Training Large Language Models (LLMs) has become a web-scale systems problem. The increasing complexity of large-scale modern training infrastructures is reflected in accelerator generations, inter-node interconnects (*e.g.,* PCIe and NVLink), and network conditions (*e.g.,* link speed and congestion control mechanisms). In addition, they support an increasingly rich set of tensor-, pipeline-, and data-parallelism strategies based on communication frameworks (*e.g.,* NCCL [21]). Even if the scheduler attempts to allocate homogeneous resources to a single job, practical limitations (*e.g.,* fragmentation and availability) can lead to significant performance differences between worker nodes. As illustrated in Figure 1, system designers face recurring cost-performance trade-offs due to hardware diversity and complex topologies [8, 35, 38]. Therefore, exploring these design points using real training executions can be both time-consuming and expensive, with even medium-sized studies ultimately requiring significant amounts of GPU time.

Existing simulators [9, 25, 32, 34] struggle to meet this challenge, as most of them are designed and validated for homogeneous clusters with fewer resource constraints. Lightweight tools can improve performance when exploring alternative configuration settings. However, they tend to use coarse abstractions and fail to consider competing agent interactions via workloads such as collective scheduling, and compute-communication overlap. Alternatively, high-fidelity simulators can capture more details, yet their workflows often contain assumptions or structures that limit their usefulness at scale when accounting for heterogeneous devices and

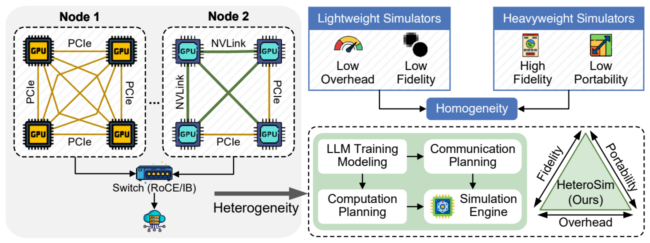


Figure 1: The motivation and positioning of HeteroSim.

mixed fabrics [12, 30]. When conducting hypothesis analysis on modern LLM training deployments, designers face a challenge to consume a faithful LLM framework while providing the practicality and good throughput.

This paper argues that the next-generation simulation framework should possess three characteristics: (i) sufficient system-level performance fidelity for planning or hypothesis analysis; (ii) low overhead for rapid exploration of large design space configurations; and (iii) flexible portability across GPU generations and cluster architectures with lightweight calibration. Achieving these goals requires a unified simulation substrate that integrates realistic LLM training workloads and compute-communication modeling.

We propose HeteroSim, a high-fidelity simulation framework for heterogeneous LLM training that strikes a balance between realism and scalability. HeteroSim is designed to sweep through multiple configurations with low overhead, while providing simulation fidelity across the workload, communications, and topology constraints. It unifies three components: (i) an Intermediate Representation (IR)-based workload compiler that captures compute-communication overlap; (ii) a heterogeneity-aware computation planner that models kernel classes using profile-driven, roofline-inspired scaling across GPU generations; and (iii) an NCCL-assisted communication planner that adheres to fine-grained message planning behaviors, aligning with the collective communication library. Thus, HeteroSim is able to perform end-to-end modeling and simulation in the context of heterogeneous LLM training, while maintaining its practicality for exploring a wide range of design spaces.

In summary, this paper makes the following contributions.

- We present HeteroSim, a high-fidelity simulator for heterogeneous LLM training that scales to large design-space sweeps while capturing compute-communication interactions.
- HeteroSim unifies a workload IR, a heterogeneity-aware computation planner, and an NCCL-assisted communication planner.
- Experimental results show that HeteroSim can achieve lower iteration time prediction errors and simulation overhead under different degrees of cluster heterogeneity.

2 Background and Motivation

We outline the background of LLM training, the limitations of existing simulators, and our motivation under heterogeneous cluster environments.

2.1 LLM Training Pipelines

Modern LLM training performs repeated iterative synchronization loops over mini-batches: forward and backward propagation to compute gradients, and updating the optimizer. To scale to large models and datasets, the system uses various parallel strategies.

(i) *Data Parallelism (DP)* replicates the model across multiple worker nodes, each worker node processing a portion of the data shards and participating in gradient synchronization (*e.g.,* AllReduce) [24]. For the baseline case using dense gradients, the total communication per synchronization step is $\Theta(|\theta|)$, where $|\theta|$ is the size of the model parameters.

(ii) *Tensor/model Parallelism (TP)* partitions tensors within a layer (*e.g.,* attention projection) across devices and synchronizes partial or full chunk results via AllGather and ReduceScatter [12, 19, 30]. The frequency of interconnections and communications reflects the layer structure in the model and typically varies with the size of the activation values and the dimension of the hidden states.

(iii) *Pipeline Parallelism (PP)* divides the layer into multiple stages, with each stage connected by activation and gradient flows [18, 36, 42]. Micro-batch processing can reduce bubble time in the pipeline. However, it introduces inter-stage balancing and inter-layer communication at the PP boundary.

In practice, production systems combine these parallel strategies with activation checkpointing mechanisms, while building unbalanced computate-communication overlap. Topology constraints (*e.g.,* NVLink domains) and implementation methods (*e.g.,* rail pinning) [8, 35] can significantly affect iteration time, particularly for TP- or PP-like configurations. Even when the scheduler attempts to dynamically allocate nodes in the same topology, performance differences between nodes may still occur due to the actual constraints of currently available resources.

2.2 Communication Substrates

Intra-Node. *PCIe* provides a shared switching fabric between GPUs and NICs, but it typically has higher single-message overhead and limited concurrency compared to dedicated GPU interconnects. *NVLink* is a higher bandwidth point-to-point or direct link with lower latency between devices, while *NVSwitch* is a high-bandwidth crossbar between many GPUs themselves [14, 16]. However, communication libraries such as NCCL [21] implement collective communication as algorithmic plans. They select a particular type of collective algorithms and channelization strategies based on message size and topology. These choices directly affect end-to-end performance and the achieved effective bandwidth.

Inter-Node. Large-scale GPU clusters deploy high-speed interconnects over different fabrics, usually implemented using InfiniBand or standard Ethernet/RDMA variants such as RoCE, and regions organized on topologies such as fat-tree topology [7, 26]. Routing is usually set for each flow either by ECMP routing [6, 10], or multi-rail architectures, where multiple NICs can be exposed per host for diversity in load balancing across rail.

A number of congestion control on these fabrics (*e.g.,* DCQCN, TIMELY, and HPCC [15, 17, 43]) will offer metrics for fair share traffic intervals and queuing time behaviors as well [39, 40].

2.3 Limitations of Existing Simulators

Despite the number of advances, existing simulation and design-space exploration tools are still limited along multiple axes, especially since most of them are primarily designed for homogeneous clusters. On the one hand, cycle-accurate or packet-level simulators offer high realism, but the runtime overhead per-simulation is high, making large-scale sweeps challenging. On the other hand, while analytical or coarse-grained models run very fast, they may overlook key effects such as collective scheduling, compute-communication overlap, and competition, which can lead to inaccurate ranking of design points. This trade-off is further amplified in heterogeneous settings due to per-type asymmetries and fabric-specific behaviors.

L1: Parallelism Coverage and Imbalance. Modern LLM training commonly composes parallelism patterns with aggressive micro-batching, which introduces pipeline bubbles, stage imbalance, and fine-grained synchronization traffic between stages. In general, simulators that lack these behaviors may miss the tail effects, straggler-driven overlap changes, and link-hotspot phenomena that impact end-to-end performance.

L2: Heterogeneity and Portability. Many approaches are designed for a special case, addressing homogeneous devices that have the same accelerators and incorporate a simplified intra and inter-node fabric. Moreover, they can revert back to uniform devices and link parameterization, limiting accuracy when modeling deployments with multi-generational accelerators and fabrics.

L3: Routing and Fabric Effects. Assuming shortest-path routing with perfect load balancing (as is common in simulators) fails to capture real-world effects such as ECMP hash granularity, static overrides, and rail affinity. These factors alter traffic distribution and contention [6, 10]. Collective communication library behaviors (*e.g.,* proxying and channelization) and topology-aware transport choices also affect intra- and inter-node traffic [5, 37].

Motivation. These limitations motivate a simulation framework that jointly considers fidelity, overhead, and portability in heterogeneous environments. Specifically, we further clarify these objectives in four requirements. Such a framework would (i) execute experiments efficiently to provide broad coverage of experiment design space; (ii) input a realistic computation graph of the training workload with micro-batch scheduling to capture balancing and tail effects (echoes back to **L1**); (iii) model computation across heterogeneous accelerators with a lightweight calibration path to retarget devices (echoes back to **L2**); (iv) compose intra and inter-node link models to reflect a more realistic fabric path atop the collective communication library (echoes back to **L3**).

Overall, these requirements aim to provide planning and hypothesis analysis for topology, routing, and parallelism choices under real-world constraints in heterogeneous environments.

3 System Overview

HeteroSim integrates workload modeling with heterogeneity-aware planning, and executes the resulting plans generated on a throughput-oriented simulation back end using a single GPU [9]. Figure 2 shows the workflow of HeteroSim, which imports training graphs and proceeds through simulation based on the execution plan.

At a high level, HeteroSim separates three layers: *workload*, *execution plan*, and *engine*. A workload encapsulates a single LLM training configuration, describing a computational graph with parallelism annotations. Also, it includes a configurable description of available devices and communication policies, encasing the heterogeneity of GPU generations and fabric domains.

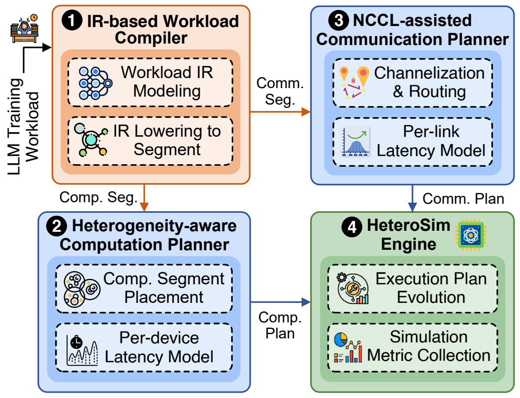


Figure 2: The overall workflow of HeteroSim.

An ❶ IR-based workload compiler lowers these inputs to a *segment graph*: A flattened sequence of compute and communication segments in dependency order, annotated with tensor sizes, kernel classes, and parallelism or collective communication information. Based on the segment graph, the ❷ heterogeneity-aware computation planner and ❸ collective communication planner produce a final *execution plan*. The execution plan instantiates a higher-level message schedule based on a calibrated computation and network model. This explicit message-level representation allows HeteroSim to represent first-order overlap and contention effects caused by dependencies and shared fabrics.

The ❹ core engine synthesizes a series of execution plans, and progresses on the compute and communication segments in accordance with dependency edges and resource constraints. Computation and communication occur independently, albeit synchronized by dependencies, often leading to compute-communication overlap and contention effects. The engine records validated metrics and analysis outputs, including per-segment time, link utilization, and tail iteration statistics. These outputs are useful for diagnosing performance bottlenecks and comparing simulated behaviors against real-system observations where measurements were available.

4 HeteroSim Design

This section presents the design of HeteroSim, which consists of an IR-based workload compiler, a heterogeneity-aware computation planner, and an NCCL-assisted communication planner.

4.1 IR-Based Workload Compiler

HeteroSim models LLM training workloads via a pluggable Intermediate Representation (IR) of the computational graph [4, 13]. The IR encapsulates operator topology, tensor shapes and data types, parallelism annotations, and synchronization barriers, and serves as a common substrate for lowering, planning, and final simulation.

IR Design. The IR is formulated as a directed acyclic graph $G(V, E)$, where each node $v \in V$ holds the following attributes:

- **Operator type** (*e.g.,* `MatMul`, `FlashAttention`, `LayerNorm`, `GELU`) for compute nodes.
- **Primitive type** (*e.g.,* `AllReduce`, `AllGather`, and `Reduce-Scatter`) for collective communication nodes.

Table 1: Kernel classes and corresponding latency models. $\mathrm{FLOPs}(d)$ **and** $\mathrm{HBMBW}(d)$ **refer to the floating-point throughput and high-bandwidth memory** ***bandwidth*** **of device** d**, respectively.** F_v **and** M_v **are per-segment FLOP and memory-byte estimates.** γ, δ **are class-specific fitted parameters.**

Class	Latency model $T_v(d)$
GEMM, ATTNCORE	$\gamma_{k(v)} \cdot \frac{F_v}{\mathrm{FLOPs}(d)} + \delta_{k(v)}$
ELEMWISE, NORM	$\gamma_{k(v)} \cdot \frac{M_v}{\mathrm{HBMBW}(d)} + \delta_{k(v)}$
MISC	piecewise mix (configurable)

- **Tensor metadata** (*i.e.,* shape and data type), and **payload estimates**: FLOPs F_v and estimated memory bytes M_v (for compute), or communication bytes B_v (for communication).
- **Parallelism annotations** (*e.g.,* TP groups, PP stages, and DP shards), with **barrier scopes**.

Edges E encode inter-node dependences with explicit *scope*. For PP, we annotate nodes with stage IDs and expand the schedule into microbatches $m = 1, \ldots, M$ following a one-forward-one-backward (1F1B) pipeline with optional warmup and cooldown. Cross-stage dependences are materialized as explicit P2P communication nodes (send/recv) inserted between stages. By doing so, the IR represents a single training iteration after unrolling (including PP microbatches). Note that repeated iterations are modeled by repeating the resulting segment schedule.

Workload IR Lowering. HeteroSim classifies compute nodes into kernel classes. As shown in Table 1, each compute node v is mapped to a class $k(v) \in \{\{\textsc{Gemm}, \textsc{AttnCore}\}, \{\textsc{Elemwise}, \textsc{Norm}\}, \textsc{Misc}\}$ based on its operator type. These classes correspond to dense matrix multiplications, attention cores, element-wise operations, normalization layers, and miscellaneous operations, respectively. The first two classes are typically compute- or bandwidth-dominated, while MISC covers mixed or configurable behaviors. For a GPU device d, this classification enables profile-driven, roofline-inspired scaling across device families and HBM bandwidth tiers without requiring microarchitecture-specific modeling (see §4.2).

Algorithm 1 outlines the lowering from IR to executable segments. Specifically, the IR is expanded into a step-level schedule with compute segments (lines 3-5), communication segments (lines 6-7), and synchronization barriers (line 11). Note that cross-stage edges for PP become `send/recv` segments carrying activations and gradients (lines 8-9). HeteroSim treats this information uniformly as byte volumes B_v under the same link model. Given the common kernel fusion in vendor libraries, HeteroSim also fuses operator sequences (*e.g.,* `MatMul+Bias+GELU`) into single compute segments with updated (F, M) estimates.

Compute-Communication Overlap. We model compute communication overlap by allowing segments from different resource classes to advance concurrently, subject to capacity constraints and IR barrier scopes. Let $\mathcal{S}_C$ denote the set of ready compute segments and $\mathcal{S}_L$ denote the set of ready communication segments. For a compute segment $i \in \mathcal{S}_C$ and a communication segment $j \in \mathcal{S}_L$, their eligibility for overlap is:

$$\mathrm{eligible}(i, j) = \mathbf{1}[R_{\mathrm{excl}}(i) \cap R_{\mathrm{excl}}(j) = \emptyset] \cdot \mathbf{1}[\text{barrier-free}], \quad (1)$$

Algorithm 1 Lowering IR to time-ordered segments.

```
Input: IR graph $G(V, E)$, parallelism group $\mathcal{G}$
Output: Segment list $\mathcal{S}$
1:  $\mathcal{S} \leftarrow [\ ]$                                  ▷ output segment list
2:  for all topologically ordered nodes $v \in V$ do
3:      if $v$ is a compute node then
4:          classify $k \leftarrow k(v)$ and compute $(F_v, M_v)$;
5:          Append $\textsc{Comp}(k, F_v, M_v, \mathcal{G})$ to $\mathcal{S}$;
6:      else if $v$ is a collective communication node then
7:          Append $\textsc{Comm}(\mathrm{type}(v), B_v, \mathcal{G})$ to $\mathcal{S}$;
8:      else if $v$ is a P2P communication node then
9:          Append $\mathrm{P2P}(\mathrm{src} \rightarrow \mathrm{dst}, B_v)$ to $\mathcal{S}$;
10:     end if
11:     Insert barriers if scope boundaries crossed;
12: end for
13: return $\mathcal{S}$
```

where $R_{\mathrm{excl}}(\cdot)$ returns the set of *exclusive* occupied hardware resources (*e.g.,* copy engines, and per-link tokens), and barrier-free indicates that no synchronization barrier is pending between the two segments under the IR barrier scopes. Link tokens and credit limits used to represent fabric concurrency are defined by the latency model in §4.3. When $\mathrm{eligible}(i, j) = 1$, the segments may still partially contend on *shared* resources (*e.g.,* DRAM bandwidth). We capture this effect with an *overlap coefficient* $0 \le \phi \le 1$ that reduces effective stall under partial contention. The overlapped latency is approximated as:

$$T_{\mathrm{overlap}} \approx \max\{T_i, T_j\} + (1 - \phi) \cdot \min\{T_i, T_j\}, \quad (2)$$

where T_i and T_j are the predicted latencies of segments i and j, respectively. The coefficient ϕ is fitted per device family and message-size regime using microbenchmarks (see §4.3). For multi-channel collective communication, each channel consumes an independent link token and can overlap with compute and other channels subject to fabric concurrency constraints.

4.2 Heterogeneity-Aware Computation Planner

Modern deployments include multiple GPU generations and diverse intra-node fabrics. When a training job spans such resources, performance heterogeneity can dominate iteration time. Given this, we model each GPU device d with a specification tuple:

$$\mathrm{Spec}(d) = \big(\mathrm{FLOPs}(d),\ \mathrm{HBMBW}(d),\ \mathrm{HBMCap}(d),\ \mathrm{NVL}(d),\ \mathrm{PCIE}(d),\ \mathrm{CE}(d)\big). \quad (3)$$

where HBMBW and HBMCap denote HBM bandwidth and capacity, respectively; NVL and PCIE capture intra-node fabric capabilities (*i.e.,* generation, and domain membership); and CE denotes the number of available copy engines, which is used as an exclusive resource for overlap eligibility. Nodes and racks form a hierarchical topology, with each level annotated by *domains* (*e.g.,* NVLink islands) that restrict placement and affect routing eligibility.

Compute Segment Placement. Given parallelism annotations, each compute segment v proceeds via a set of logical concurrent "slots". HeteroSim maps these slots to physical, potentially heterogeneous GPU devices (the parallelism group $\mathcal{G}$) while respecting domain constraints and optimizing predicted performance. For a candidate placement Π, we score it with the following cost model:

$$\mathcal{J}(\Pi) = w_{\text{tp}} \cdot \text{CommCost}_{\text{TP}}(\Pi) + w_{\text{pp}} \cdot \text{EdgeCost}_{\text{PP}}(\Pi) + w_{\text{dp}} \cdot \text{Balance}_{\text{DP}}(\Pi) + w_{\text{het}} \cdot \text{SkewPenalty}(\Pi), \quad (4)$$

where $\text{CommCost}_{\text{TP}}$ estimates TP communication cost using path latencies and bandwidth-normalized message sizes, $\text{EdgeCost}_{\text{PP}}$ estimates PP activation/gradient transfer cost across slower fabrics, $\text{Balance}_{\text{DP}}$ penalizes DP shard clustering on the same rail (when multi-rail NICs exist), and SkewPenalty measures within-group heterogeneity (*e.g.,* variance of predicted per-slot compute latency). Their weights are set to be equal by default.

Constraints. We enforce the following placement constraints. (i) *TP locality*: each TP group preferably resides within a single high-bandwidth domain (*e.g.,* an NVLink domain); otherwise, cross-domain links are minimized; (ii) *PP placement*: we prefer placing each PP stage within a locality region (*e.g.,* a node or fast intra-node domain) when feasible, and minimize gradient transfers over slower links across stage boundaries; (iii) *Rail affinity for DP*: when multi-rail NICs are available, DP ranks are distributed across rails to improve aggregate bisection bandwidth and reduce rail-level hot spots; (iv) *Homogeneity preference*: when heterogeneous GPU models are present, we prefer within-group homogeneity (especially within TP groups) to reduce timing skew in tightly synchronized collective communication; and (v) *Memory capacity*: per-device HBM capacity limits are enforced with reserve margins for optimizer states and activation checkpointing [24, 42].

Algorithm 2 illustrates the heterogeneity-aware placement algorithm of HeteroSim. It initializes a *greedy domain-first* placement by packing TP groups within the largest available NVLink domains and breaking ties by remaining HBM headroom to preserve feasibility. It then places PP stages to reduce cross-node PP transfers, and spreads DP shards across rails to balance bandwidth utilization when applicable. This initialization is fast and yields a feasible placement for large device pools. We then refine the placement using a *local improvement metaheuristic* guided by $\Delta\mathcal{J}$ (lines 2-6). At each iteration, the algorithm proposes a neighbor $\Pi' \leftarrow \mathcal{N}(\Pi)$ using: (i) intra-domain swaps to reduce TP communication cost, (ii) cross-rail swaps to improve DP balance, and (iii) stage reseating to reduce PP transfer costs. A change is accepted if it improves the cost, or with simulated-annealing probability $\min\{1, \exp(-\Delta\mathcal{J}/\tau_t)\}$ with temperature τ_t, to escape local minima.

Even with careful placement, device heterogeneity and pipeline imbalance can induce latency skew within parallelism groups. To study and mitigate this effect, HeteroSim supports two optional knobs at the planning level. The first is *channel-rail mapping with weighted quotas*: for collective communication, we can bias channel assignments toward higher-capacity rails to reduce completion-time skew under heterogeneous fabrics. The second is *stage-aware pipeline scheduling*: we explore PP schedule parameters and partitioning choices (*e.g.,* warmup/cooldown and stage boundaries) that reduce bubbles or hot spots under heterogeneous per-stage costs, without changing model semantics.

Compute Latency Model. For a compute segment v, let S_v be the set of slots induced by parallelism group $\mathcal{G}$. For each slot $s \in S_v$ placed on device $d = \Pi(s)$, we model per-slot latency via a roofline-inspired class model $T_v(d)$ with fitted parameters (see Table 1). To avoid overly optimistic extrapolation across device families, we introduce a *bounded transfer* mechanism that caps the maximum implied speedup when extrapolating from a calibrated reference device, given by,

Algorithm 2 Heterogeneity-aware placement algorithm.

```
Input: GPU device pool $\mathcal{D}$, parallelism groups $\mathcal{G}_{\text{TP}}, \mathcal{G}_{\text{PP}}, \mathcal{G}_{\text{DP}}$
Output: Compute segment placement $\Pi$
1: $\Pi \leftarrow$ GreedyDomainFirst($\mathcal{D}, \mathcal{G}_{\text{TP}}, \mathcal{G}_{\text{PP}}, \mathcal{G}_{\text{DP}}$)
2: for $t = 1$ to $T$ do
3:     propose swap/exchange $\Pi' \leftarrow \mathcal{N}(\Pi)$
4:     $\Delta \leftarrow \mathcal{J}(\Pi') - \mathcal{J}(\Pi)$
5:     if $\Delta < 0$ or rand() $< \min\{1, \exp(-\Delta/\tau_t)\}$ then
6:         $\Pi \leftarrow \Pi'$
7:     end if
8: end for
9: return $\Pi$
```

$$T_v^{\text{bound}}(d) = \max\Big(T_v(d),\ \lambda \cdot T_v(d^\star) \cdot \rho_T(d^\star \rightarrow d)\Big), \quad (5)$$

where $d^\star$ is a reference device with calibrated parameters, $\rho_T(d^\star \rightarrow d)$ is a monotone *time-scaling lower bound* derived from device specification ratios by kernel classes (*e.g.,* FLOPs or HBMBW). It limits how much the predicted latency can decrease under specs-based extrapolation, and $\lambda \geq 1$ is a configurable safety margin. This prevents unrealistically small latency estimates when kernel efficiency drops on newer architectures or under different memory hierarchies. Based on the chosen parallelism strategy, we then aggregate per-slot latencies to obtain the compute-segment latency.

4.3 NCCL-Assisted Communication Planner

LLM training frameworks widely use NVIDIA Collective Communications Library (NCCL) [21] to implement collective communication in production. To align with this, HeteroSim transforms high-level collective communication into executable message plans by capturing NCCL-inspired message-planning behaviors (*e.g.,* channelization, chunking, and proxying), and calibrated key parameters to observed performance.

Message Plan Generation. A collective communication segment $v = (\text{type}, B_v, \mathcal{G})$ is defined by its primitive *type*, payload size B_v, and communicator group $\mathcal{G}$. In practice, NCCL exploits *channels* (*i.e.,* parallel subflows) and *chunking* (*i.e.,* payload partitioning) to pipeline transfers across links.

Channels. Each channel $q \in [0, c)$, where c is the number of channels, instantiates an isomorphic copy of the collective algorithm's step graph, potentially mapped onto disjoint paths. In fact, messages on distinct channels are independent and therefore can overlap subject to concurrency limits.

Chunks. Within each channel, the payload is partitioned into chunks of target size s to enable pipelining and overlap. Each channel carries approximately B_v/c bytes and is further partitioned into $J = \left\lceil \frac{(B_v/c)}{s} \right\rceil = \left\lceil \frac{B_v}{c \cdot s} \right\rceil$ chunks. Especially for ring-like algorithms, chunks are pipelined such that step t of chunk j can overlap with step $t+1$ of chunk $j-1$.

HeteroSim materializes a message plan at *step granularity.* Let $S_{\text{algo}}(|\mathcal{G}|)$ denote the number of *P2P message instances* induced by the chosen collective algorithm for one (channel,chunk) over group $\mathcal{G}$. The total number of message instances is:

$$\mathcal{M}_v = \{ m_k \}_{k=1}^{K}, \quad K = c \times J \times S_{\text{algo}}(|\mathcal{G}|), \tag{6}$$

where each message is

$$m_k \triangleq (src_k,\ dst_k,\ B_k,\ channel_k,\ chunk_k,\ step_k), \tag{7}$$

with src_k and dst_k being the source and destination devices, and $channel_k$, $chunk_k$, and $step_k$ denoting channel, chunk, and step IDs. The payload B_v is partitioned across channels and chunks, yielding a base per-(channel,chunk) volume $\widetilde{B} \approx B_v/(c \times J)$. The per-step message volume B_k depends on the selected algorithm's step partitioning. For ring-like reduce-scatter/allgather-style collective communication, each step transfers a $1/|\mathcal{G}|$ slice of the chunk, hence $B_k = \widetilde{B}/|\mathcal{G}| = B_v/(|\mathcal{G}| \cdot c \cdot J)$. We similarly instantiate B_k for other algorithms using their corresponding partitioning rules.

If proxying (*e.g.,* PXN-like behavior) is enabled for *inter-node* communication, a direct message $(i \rightarrow j)$ that traverses an inter-node link can be rewritten into two legs $(i \rightarrow p)$ and $(p \rightarrow j)$ to stage traffic via a proxy GPU. We select eligible proxies p on the same node, prioritizing GPUs that are NIC-adjacent on the corresponding rail and well-connected (*e.g.,* within the same NVLink domain) for efficient intra-node staging. Each original message m_k is split into two messages, and each leg is subsequently routed and timed by the link model. That is, we model proxying as a two-leg staging abstraction at the message-planning level, rather than reproducing vendor internals bit-for-bit.

Message Routing. For topologies with multiple candidate paths, we precompute a set of R routes and select one using a stable hash over a *message key*: $(src, dst,$ collective ID, channel, chunk) to reduce systematic collisions. For hypothesis exploration, HeteroSim supports weight-based routing policies, where weights can be derived from offline estimates (*e.g.,* oversubscription) or from simulated queue/utilization statistics, without assuming the availability of real-time congestion telemetry in production. Hand-tuned static routing entries can also be specified with the highest priority to enforce deterministic path control. The selected route is materialized via per-link queue enqueues within the HeteroSim engine.

Communication Latency Model. Each message m_k is mapped to a path $\pi_k = (e_1, e_2, \dots)$ over a series of links. Each link $e \in \pi_k$ uses a piecewise latency model over message-size buckets:

$$T_e(B_k) = \alpha_{e,b} + \frac{B_k}{\beta_{e,b}}, \quad B_k \in \text{bucket } b, \tag{8}$$

where $\alpha_{e,b}$ represents fixed overheads (*e.g.,* DMA setup and launch overheads) and $\beta_{e,b}$ is the effective bandwidth for link e in bucket b. For each link type and bucket, (α, β) are empirically fitted using microbenchmarks under controlled concurrency. We distinguish two broad link classes:

- *Intra-node links.* PCIe, and NVLink links are modeled with piecewise (α, β) and explicit *fabric concurrency* via tokens per link domain to represent limited communication resources. We incorporate shared credit pools to approximate contention domains for shared buffers and bandwidth.
- *Inter-node links.* NIC ports and switch uplinks (*e.g.,* over InfiniBand or RoCE-like deployments) are modeled with piecewise (α, β) fits and simplified queuing dynamics. We approximate serialization and store-and-forward queuing delays using FIFO queues that capture bandwidth sharing and head-of-line blocking. Congestion-control effects in RoCE-style deployments (*e.g.,* DCQCN-like behavior [43]) are incorporated by adapting the effective throughput parameter β as persistent backlog builds up, rather than modeling the full feedback-control loop.

Table 2: Five GPU cluster configurations (H0-H4), including one homogeneous baseline (H0) and four heterogeneous settings (H1-H4). Ratios denote the *target mixes* of device and intra-/inter-node fabric types in each level.

Level	GPU mix (H100:A100)	Intra-node fabric	Inter-node fabric
H0 (homog.)	only A100	only N3P4	only RoCE
H1 (device-mix)	50:50	only N3P4	only RoCE
H2 (link-mix)	only H100	N4P5:N3P4 = 50:50	IB:RoCE = 50:50
H3 (balanced)	50:50	N4P5:N3P4 = 50:50	IB:RoCE = 50:50
H4 (all-mix)	25:75	N4P5:N3P4 = 25:75	IB:RoCE = 25:75

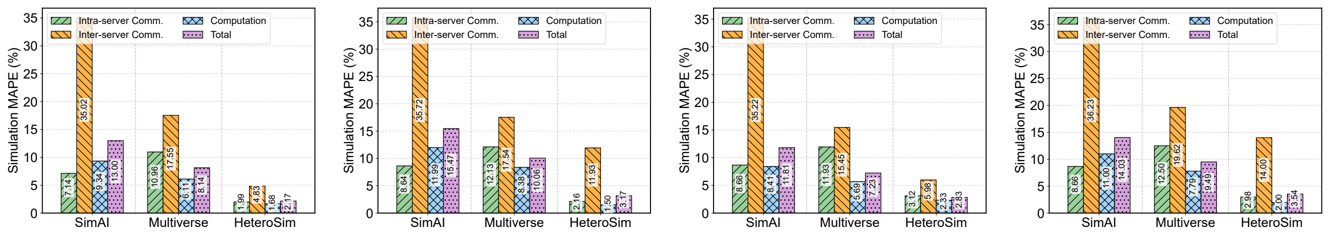
5 Evaluation

5.1 Experimental Setup

Workload and Testbed Setup. All simulations run on a workstation with an NVIDIA H100 GPU, a 64-core Intel CPU@2.10GHz, and 256GB RAM. HeteroSim is implemented as a lightweight control plane and a data-plane simulator. The control plane imports LLM training graphs, lowers them into our workload IR, and generates execution plans by composing the computation planner and the communication planner. The data-plane simulator then executes the generated message-level plans, binds each message flow to calibrated intra- and inter-node fabrics, and records per-segment timing and communication breakdowns for validation and analysis. To validate the accuracy of HeteroSim, we run real LLM training jobs with GPT 13B [3] and LLaMA 30B [29] on a fat-tree GPU cluster. The cluster comprises 32 servers, each equipped with eight GPUs (A100 or H100). A100 nodes use ConnectX-6 NICs with RoCEv2 at 200Gbps and NVLink 3.0 + PCIe Gen 4.0 (N3P4), whereas H100 nodes use ConnectX-7 NICs with InfiniBand at 400Gbps and NVLink 4.0 + PCIe Gen 5.0 (N4P5) [14, 16].

Heterogeneity. We evaluate five GPU-cluster configuration levels (H0-H4) shown in Table 2. The ratios denote the *target* proportions of device and fabric types. For configuration levels exhibiting native heterogeneity (*e.g.,* H3 and H4), we directly map ranks to physical nodes, fully utilizing their native intra-node bandwidths and corresponding network fabrics. Conversely, for emulated configurations (*e.g.,* H1 and H2), we apply traffic shaping when the physical link bandwidth exceeds the target specification (*e.g.,* evaluating H100 nodes within the N3P4-constrained H1 setting) or when creating fine-grained link mixes. Specifically, we throttle the faster physical links to match the calibrated (α, β) parameters of the target lower-bandwidth generation [11]. For each link type, we run lightweight microbenchmarks, ping-pong latency tests, and streaming throughput measurements, to fit calibrated parameters (α, β) per bucket, interpolating between sampled sizes [11]. In addition, different heterogeneity levels reuse these calibrated parameters.

Baselines. We compare HeteroSim against two representative simulators: (i) *SimAI* [32] is a full-stack simulator that instruments training to record fine-grained compute and communication traces



(a) GPT 13B with 64 GPUs. (b) GPT 13B with 128 GPUs. (c) LLaMA 30B with 64 GPUs. (d) LLaMA 30B with 128 GPUs.

Figure 3: The simulated error of training iteration time on heterogeneous GPU clusters under the H4 level.

Table 3: The simulation MAPE (%) at 128 GPUs across heterogeneity levels H0-H4.

Workload	System	H0	H1	H2	H3	H4
GPT 13B	SimAI	5.91	7.12	10.43	13.58	15.47
	Multiverse	1.93	3.04	5.31	9.62	10.06
	HeteroSim	**1.51**	**1.92**	**2.53**	**3.24**	**3.17**
LLaMA 30B	SimAI	3.92	6.27	9.73	11.98	14.03
	Multiverse	2.08	3.37	5.68	8.13	9.49
	HeteroSim	**1.69**	**1.91**	**2.15**	**3.11**	**3.54**

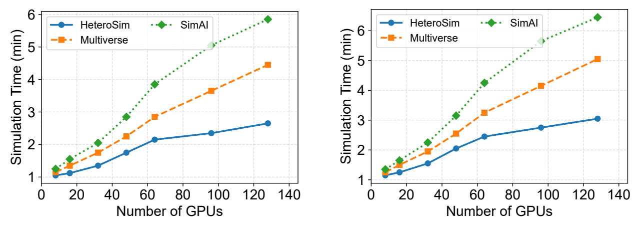
and replays the NCCL-like behaviors. (ii) *Multiverse* [9] is a GPU-accelerated, single-program-multiple-experiments simulator for design-space exploration at scale.

5.2 Overall Performance

Simulation Accuracy. Figure 3 compares the simulated and real iteration time of HeteroSim against the baselines. Across both workloads under the H4 heterogeneity level, HeteroSim achieves the lowest error in per-iteration training time, with total MAPE below 3.54%. On average, HeteroSim reduces errors by 66.4% and 78.4%, respectively, compared to Multiverse and SimAI. Meanwhile, the time breakdown closely follows the ground truth, and the expected rise of the inter-server share at larger scales is preserved.

The baselines underperform under heterogeneity in our configuration. SimAI largely assumes homogeneous or uniformly parameterized links and uses fabric-agnostic queueing without explicit rail locality, which can over-estimate inter-server phases as scale grows. Multiverse improves throughput via GPU execution, but its heterogeneity modeling remains coarser: collective communication is modeled at the segment level rather than per-message, and link selection is not consistently bound to per-type calibration, which makes it harder to reproduce measured overlap. By contrast, HeteroSim binds each NCCL message to a calibrated per-type link model, enforces heterogeneity-aware placement and rail locality, and preserves measured overlap. Thus, it yields lower errors that are important for decision-making in heterogeneous clusters.

Simulation Overhead. As depicted in Figure 4, HeteroSim reduces total simulation time by 28.3%-43.6% compared to the baselines. SimAI is CPU-bound and planner-heavy. Its discrete-event pipeline expands mixed-fabric and mixed-path communication into large queues without explicit rail locality or per-link calibration. This increases per-simulation overhead and triggers super-linear growth as



(a) GPT 13B. (b) LLaMA 30B.

Figure 4: The simulation time for different workloads under the H4 heterogeneity level.

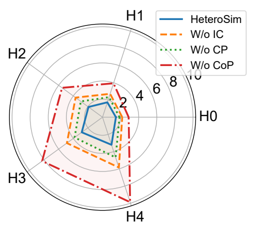
the scale rises. Multiverse improves throughput via GPU execution, but still pays a sizable planning cost in our setting due to frequent collective-plan computation. Moreover, segment-level link selection can interact with algorithm and bucketization changes under H2, leading to additional planning overhead as scale increases. HeteroSim caches heterogeneity-aware placements, collective communication plans, and per-link/device models for reuse across scales, lowering simulation overhead in heterogeneous clusters.

5.3 Scalability

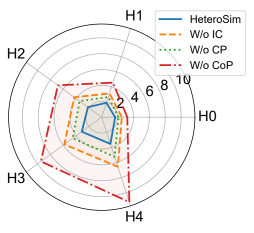
As shown in Table 3, HeteroSim sustains few-percent simulation error for both workloads (1.51%-3.54% MAPE) as heterogeneity varies from H0 to H4. Baseline errors increase with heterogeneity, with larger gaps appearing in PCIe-heavy and RoCE-heavy regimes where cross-node collective communication dominates. In our configuration, SimAI's CPU DES applies coarse, fabric-agnostic queueing that can overstate inter-server phases under mixed link types. Multiverse narrows the gap via GPU execution, yet still reasons with coarser overlap and fabric selection. In contrast, HeteroSim replays NCCL at message granularity, binds flows to calibrated per-type link models, and preserves rail locality without accumulating conservative bias as heterogeneity intensifies.

5.4 Ablation Study

We ablate HeteroSim with three variants: (i) *W/o IR-based Workload Compiler (IC).* The workload graph is simplified by removing fine-grained communication semantics and overlap, and compute and communication segments execute serially with no contention. (ii) *W/o Computation Planner (CP).* The devices are treated as a single GPU generation, and the placement is topology-agnostic. (iii) *W/o*

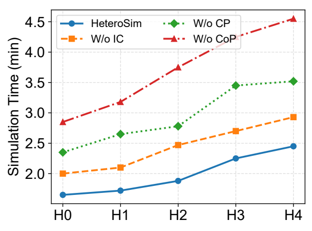


(a) 64 GPUs.

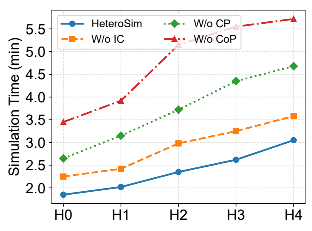


(b) 128 GPUs.

Figure 5: The simulation error (%) of LLaMA 30B with different HeteroSim variants.



(a) 64 GPUs.



(b) 128 GPUs.

Figure 6: The simulation overhead of LLaMA 30B with different HeteroSim variants.

Communication Planner (CoP). The collective communication is modeled at the segment level with a unified link model, without NCCL-like message plans.

As depicted in Figure 5, removing CoP produces the largest fidelity drop, with a simulation MAPE of up to 11.26%. This is because segment-granularity modeling collapses channelization and rail pinning into an average link, blurring IB/RoCE and NVLink/PCIe asymmetries that dominate in mixed fabrics. Disabling IC makes the simulator systematically underestimate latency variance and synchronization overhead by omitting fine-grained overlap effects. Also, the serialized schedule overstates compute stalls and understates communication hiding without overlap. Turning off CP has a moderate effect but consistently increases error on inter-server time, as topology-agnostic placement leaks traffic across slow rails and misaligns TP and PP boundaries with domains.

The ablation results *w.r.t.* simulation overhead are illustrated in Figure 6. Similarly, CoP is most consequential: removing it prevents reuse of cached collective plans and forces coarser-grained communication modeling, increasing runtime and overhead under mixed fabrics. Removing CP introduces mild slowdowns under the H3 and H4 levels due to less stable placement and routing choices. In contrast, IC minimally affects runtime but inflates tail variability in communication-heavy phases.

6 Related Work

LLM Training Simulation. Recent research has improved simulation performance and accelerated DNN/LLM training simulations using GPUs. Multiverse [9] proposes a novel GPU execution model that employs a data-oriented execution to batch process multiple experiments, improving the throughput of design space exploration. SimAI [32] reproduces the behavior of NCCL through detailed calibration and point-to-point playback. ASTRA-sim [25, 34] employs a parsing- and queue-based modeling approach for system-level exploration. However, they use uniform parameterization of devices and links or coarser structural abstractions to describe the environment, which may mask the asymmetry between device types or structures between heterogeneous devices and hybrid structures. HeteroSim extends these efforts by compiling explicit computation and communication segments into message-granular plans and binding messages to each type of calibrated link model. General-purpose simulators (*e.g.,* SimGrid [23]) and microarchitectural simulators (*e.g.,* gem5-gpu [22]) provide different freedom and fidelity-scability trade-offs, which are largely orthogonal to ours.

Calibration and Performance Modeling. Portability performance prediction typically relies on roofline inference [33] for computation and α/β-style models [11] for communication. HeteroSim packages these abstract concepts into reusable calibration kernels and link packets, and combines them through an explicit, NCCL-compliant message schedule. In fact, NCCL [21] dynamically selects the algorithm, channelization method, and proxy path (*e.g.,* PXN [20]). Therefore, a more realistic modeling approach is to expose the channel and path structure, rather than assuming idealized bandwidth. Based on the classic collective analysis [28] and hierarchical decomposition methods [5], HeteroSim compiles collective operations into NCCL-like P2P message instances and binds each message to a calibrated single-link α/β fit [11]. Meanwhile, it applies roofline-style scaling [33] to improve portability across devices. Communication reduction methods (*e.g.,* quantization [1] and optimizer compression [27]) are complementary, mainly manifested in changes in payload size or policy threshold.

Network and Data Center Fabrics. Packet-level simulators (*e.g.,* OMNeT++ [31]) and congestion-control designs (*e.g.,* DCTCP [2], DCQCN [43], TIMELY [17], HPCC [15]) can capture detailed network dynamics and feedback loops. However, their single-experiment overhead can be too large for large-scale hypothesis analysis. On the other hand, HeteroSim uses calibratable link and queue primitives to efficiently combine routing [41], rail affinity, and contention for training simulation, and draws on multipath and load balancing insights from Presto [10] and DRILL [6].

7 Conclusion

This paper presents HeteroSim, a high-fidelity simulator for heterogeneous LLM training. HeteroSim designs a pluggable IR-based workload compiler with broad parallelism coverage. Also, it unifies a heterogeneity-aware computation planning with kernel-wise placement and calibrated scaling, and an NCCL-assisted collective communication planning with calibrated intra- and inter-node link models and configurable routing. Across a wide range of heterogeneity levels, experimental results show that HeteroSim attains accurate simulation against real runs, enabling fast design-space exploration across heterogeneous cluster environments.

Acknowledgments

This research was supported by Zhejiang Provincial Natural Science Foundation of China under Grant No. LQN26F020008 and CAAI-Ant Group Research Fund (2025CAAI-ANT-15).

References

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in neural information processing systems*, 30, 2017.

[2] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM*, pages 63–74, 2010.

[3] Tom Brown, Benjamin Mann, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Tianqi Chen, Thierry Moreau, et al. TVM: An automated End-to-End optimizing compiler for deep learning. In *OSDI*, pages 578–594, 2018.

[5] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. *Proceedings of Machine Learning and Systems*, 1:241–251, 2019.

[6] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *ACM SIGCOMM*, pages 225–238, 2017.

[7] Albert Greenberg, James R Hamilton, et al. Vl2: A scalable and flexible data center network. In *ACM SIGCOMM*, pages 51–62, 2009.

[8] Juncheng Gu, Mosharaf Chowdhury, et al. Tiresias: A GPU cluster manager for distributed deep learning. In *USENIX NSDI*, pages 485–500, 2019.

[9] Fei Gui, Kaihui Gao, Li Chen, et al. Accelerating design space exploration for {LLM} training systems with multi-experiment parallel simulation. In *USENIX NSDI*, pages 473–488, 2025.

[10] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[11] Roger W Hockney. The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel computing*, 20(3):389–398, 1994.

[12] Zhihao Jia et al. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

[13] Chris Lattner, Mehdi Amini, et al. MLIR: Scaling compiler infrastructure for domain specific computation. In *IEEE/ACM International Symposium on Code Generation and Optimization*, pages 2–14, 2021.

[14] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, et al. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):94–110, 2019.

[15] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, et al. Hpcc: High precision congestion control. In *ACM SIGCOMM*, pages 44–58. 2019.

[16] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. Pump up the volume: Processing large data on gpus with fast interconnects. In *ACM SIGMOD*, pages 1633–1649, 2020.

[17] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, et al. TIMELY: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[18] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, et al. Pipedream: Generalized pipeline parallelism for dnn training. In *ACM SOSP*, pages 1–15, 2019.

[19] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pages 1–15, 2021.

[20] NVIDIA. Doubling all2all performance with nvidia collective communication library 2.12. https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/, 2022.

[21] NVIDIA. NVIDIA Collective Communication Library (NCCL) Documentation, 2025. https://docs.nvidia.com/deeplearning/nccl.

[22] Jason Power, Joel Hestness, Marc S Orr, Mark D Hill, and David A Wood. gem5-gpu: A heterogeneous cpu-gpu simulator. *IEEE Computer Architecture Letters*, 14(1):34–36, 2014.

[23] Martin Quinson. Simgrid: a generic framework for large-scale distributed experiments. In *Peer-to-Peer Computing*, pages 95–96, 2009.

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

[25] Saeed Rashidi et al. Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 81–92, 2020.

[26] Ankit Singla et al. Jellyfish: Networking data centers randomly. In *NSDI*, pages 225–238. USENIX Association, 2012.

[27] Hanlin Tang, Shaoduo Gan, et al. 1-bit adam: Communication efficient large-scale training with adam's convergence speed. In *International Conference on Machine Learning*, pages 10118–10129, 2021.

[28] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[29] Hugo Touvron et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[30] Colin Unger, Zhihao Jia, et al. Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 267–284, 2022.

[31] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *International conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, 2008.

[32] Xizheng Wang, Qingxu Li, Yichi Xu, et al. SimAI: Unifying architecture design and performance tuning for large-scale large language model training with scalability and precision. In *USENIX NSDI*, pages 541–558, 2025.

[33] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[34] William Won, Taekyung Heo, et al. Astra-sim2. 0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *IEEE ISPASS*, pages 283–294, 2023.

[35] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, et al. Gandiva: Introspective cluster scheduling for deep learning. In *USENIX OSDI*, pages 595–610, 2018.

[36] Xiaofei Yue, Song Yang, Fan Li, Liehuang Zhu, Xu Wang, Zhen Feng, and Fernando A Kuipers. Hyfaas: Accelerating serverless workflows by unleashing hybrid resource elasticity. *IEEE Transactions on Parallel and Distributed Systems*, 37(1):272–286, 2025.

[37] Xiaofei Yue, Song Yang, Liehuang Zhu, Stojan Trajanovski, and Xiaoming Fu. Demeter: Fine-grained function orchestration for geo-distributed serverless analytics. In *IEEE INFOCOM*, pages 2498–2507, 2024.

[38] Hanyu Zhao, Zhenhua Han, et al. HiveD: Sharing a GPU cluster for deep learning with guarantees. In *USENIX OSDI*, pages 515–532, 2020.

[39] Ziming Zhao, Zhaoxuan Li, Zhuoxue Song, Wenhao Li, and Fan Zhang. Trident: A universal framework for fine-grained and class-incremental unknown traffic detection. In *Proceedings of the ACM Web Conference 2024*, pages 1608–1619, 2024.

[40] Ziming Zhao, Zhaoxuan Li, Zhuoxue Song, Fan Zhang, and Binbin Chen. Rids: Towards advanced ids via rnn model and programmable switches co-designed approaches. In *IEEE INFOCOM*, pages 591–600. IEEE, 2024.

[41] Ziming Zhao, Zhaoxuan Li, Xiaofei Xie, Zhipeng Liu, Tingting Li, Jiongchi Yu, Fan Zhang, and Binbin Chen. Verify all traffic: Towards zero-trust in-network intrusion detection against multipath routing. *IEEE Journal on Selected Areas in Communications*, 2025.

[42] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, et al. Alpa: Automating inter-and intra-operator parallelism for distributed deep learning. In *USENIX OSDI*, pages 559–578, 2022.

[43] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, et al. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.