SimAI: Unifying Architecture Design and Performance Tunning for Large-Scale Large Language Model Training with Scalability and Precision

Xizheng Wang[1,2*], Qingxu Li[1*], Yichi Xu[1*], Gang Lu[1†], Dan Li[2], Li Chen[3], Heyang Zhou[1]
Linkang Zheng[1,4], Sen Zhang[1], Yikai Zhu[1], Yang Liu[1], Pengcheng Zhang[1], Kun Qian[1], Kunling He[1]
Jiaqi Gao[1], Ennan Zhai[1], Dennis Cai[1], Binzhang Fu[1]
[1]Alibaba Cloud, [2]Tsinghua University
[3]Zhongguancun Laboratory, [4]South China University of Technology

Abstract

The large number of GPUs required for a single LLM training significantly hinders the validation of new designs, tunings, and optimizations, calling for the occurrence of efficient simulators. Existing simulators, however, only target a specific granularity of the entire training, intrinsically leading to imprecision. This paper presents SimAI, a unified simulator aiming at precisely and efficiently simulating the LLM training procedure at scale. Through selective and high-fidelity integration of the training frameworks, the kernel computation, and the collective communication library into the simulating procedure, SimAI achieves high precision in simulations. SimAI further conducts multi-thread acceleration and implements lock-free global context-sharing to accelerate the execution speed. The effectiveness of SimAI is validated by its performance results, which show an average of 98.1% alignment to real-world results under various test scenarios and affirm its robustness and adaptability from small-scale labs to large-scale industrial environments. SimAI delivers meaningful guidelines for new host designs and parameter settings, directly benefiting in-production LLM training. We also share experiences and lessons learned during the evolution of SimAI. SimAI is open sourced at https://github.com/aliyun/SimAI.

1 Introduction

As the field of Artificial Intelligence (AI) rapidly advances, particularly with the rise of large language models like OpenAI's GPT-4 [41], the need for scaling AI infrastructure has grown significantly. For instance, training GPT-4 reportedly requires around 25,000 state-of-the-art GPUs [19]. This massive resource demand presents a significant barrier to entry for organizations looking to compete in this high-stakes domain.

In this context, simulators become essential both before and after infrastructure investments. In the planning phase, simulators help organizations estimate the scale and architecture required to achieve performance goals. In the operation phase, they help increase resource utilization, ensuring a return on investment. Thus, simulators are not just tools for improving efficiency but are strategic assets that maximize resource use and ensure that infrastructure investments deliver measurable results.

Currently, it is common practice to use separate simulators for capacity planning and performance tuning. For capacity planning, simulation is typically performed at the flow or job level, ignoring packet-level behavior [32]. In contrast, performance tuning relies on packet-level simulations to analyze network traffic patterns, latency, and packet loss—factors critical for optimizing communication and computation in AI model training and inference.

However, our experience shows that using multiple simulators with different levels of granularity presents three main challenges. First, this approach leads to inaccurate cost-performance analyses, making it difficult to predict load-balancing performance and assess system failures. Second, the poor performance of detailed simulators and the low accuracy of coarse-grained simulators limit the ability to optimize model training in large-scale deployments. Finally, this fragmented approach complicates development and testing, increasing the risk of discrepancies between simulated and real-world performance.

To overcome these challenges, we developed a unified simulator that handles both capacity planning and performance tuning within a single framework. However, this approach introduces several difficulties:

- Generating workloads with high precision to reflect realistic AI training behaviors.
- Simulating computation accurately across various GPU architectures.
- Precisely modeling communication to account for network traffic patterns and latencies.

* Equal Contribution
† Corresponding Author

- Scaling the simulator to support diverse, large-scale AI infrastructure configurations.

In this paper, we share the development and operation experience of SimAI, a unified simulator designed for scalable, high-precision simulations of large-scale LLM training. SimAI addresses these challenges by introducing high-fidelity models across the entire AI training stack. To generate precise workloads for LLM training at any scale, SimAI "hijacks" mainstream training frameworks, such as Megatron [52] and DeepSpeed [46], to run on a single host and create fine-grained workloads (§3.2). Different solutions ensure accurate simulation of both computation and communication. For computation, we break the workload into fine-grained kernels, measuring execution times on existing GPUs and mapping them to other GPU types (§3.3). For communication, we "hijack" the NVIDIA Collective Communications Library (NCCL) [36] to accurately simulate packet-level behavior for collective communication (§3.4). To improve the simulator's efficiency, we implement multi-threaded acceleration and lock-free global context sharing among threads (§3.5).

The performance results of our unified simulator are a testament to its efficacy. In terms of accuracy, it achieves an average deviation of only 1.9% compared to real-world results across various test scenarios. On the scalability front, the simulator handles simulations from small-scale lab environments to large-scale industrial deployments, proving its robustness and adaptability (§4).

Since incorporating this unified simulator into our AI development pipeline, we have served various teams and gained valuable insights. It has improved our infrastructure management and accelerated AI model development and deployment. We share the benefits and contributions of SimAI, including guidelines for new host designs and accurate assessments of scaling benefits (§5). These guidelines have been adopted by engineering teams and incorporated into production deployments. Additionally, we share lessons learned in transforming SimAI from a standalone simulator to a widely used simulation service (§6).

SimAI is a high-precision, full-stack simulator designed to benefit researchers across various domains involved in Large Language Model (LLM) training. This versatile tool caters to multiple levels of the LLM training ecosystem. At the framework level, SimAI enables the exploration of optimal parallel strategies and communication-computation overlap techniques, facilitating parameter tuning to reduce end-to-end training time. For collective communication research, it offers a platform to validate and quantify novel algorithms' performance gains. SimAI's system architecture design allows for experimentation with diverse intra-host and inter-host configurations, helping identify the most cost-effective solutions. From a networking perspective, the simulator supports investigations into the impact of various congestion control algorithms (such as DCQCN [64], HPCC [26], etc.), network protocols (such as RoCEv2 [17], Solar [33], etc.), and routing strategies—including adaptive routing—across different architectural setups. By providing the flexibility to customize and fine-tune different components, SimAI empowers users to conduct multifaceted research accelerating LLM training processes, and is an invaluable tool for scholars and practitioners throughout the LLM development pipeline.

2 Background and Motivation

2.1 AI Training Infrastructure

Large language models (LLMs) require specialized infrastructure, often involving dozens to thousands of GPUs working together to handle pretraining or fine-tuning tasks. For instance, training a GPT-3 model with 175 billion parameters demands 1,024 high-end GPUs running continuously for 34 days [35]. To optimize GPU usage, mainstream training frameworks like Megatron [52] and DeepSpeed [46] offer parallelization techniques such as Data Parallelism (DP), Pipeline Parallelism (PP), and Tensor Parallelism (TP). These methods enable the efficient distribution of training tasks across multiple GPUs. The process relies on collective communication libraries (CCLs) to manage data exchanges, such as using `AllReduce` for gradient synchronization or `AllGather` for parameter sharing. The CCL breaks each collective communication task into a series of peer-to-peer `Send` and `Receive` operations to carry out the necessary data transfers between GPUs.

In Alibaba's clusters, each server contains multiple GPUs. GPUs within the same server are connected through a high-bandwidth intra-host network, such as NVLink or NVLink Switch [40], and each GPU connects to the inter-host RDMA network via network interface cards (NICs). For example, in line with [23], our A100 servers are equipped with eight NVIDIA A100 GPUs [1] and four NVIDIA CX6Dx NICs, each providing 2×100Gbps [6] bandwidth. Each GPU is linked to other GPUs in the same server via a 600GB/s NVLink and to GPUs in other servers via a 100Gbps RDMA network. In production, Megatron and DeepSpeed are the two dominant frameworks, NCCL [36] is the dominant CCL.

2.2 Demands for a Unified Simulator

The rapid evolution of LLMs necessitates advancements in AI training infrastructure and optimization methods. To address these challenges, simulations are crucial for three primary goals:

Comprehensive evaluation of AI infrastructure. To ensure the effective deployment of new hardware and configurations, AI infrastructure must be evaluated from multiple perspectives:

GPU Selection: Before adopting new GPU models, cloud service providers (CSPs) need to evaluate their performance on AI workloads at scale. Simulations allow for a detailed

preview of how different GPUs perform in large-scale deployments, guiding purchasing decisions and scaling strategies.

Network Architecture Design: Once specific GPUs and intra-host interconnects are chosen, the next challenge is optimizing network architecture for scalability. Key questions, such as the ideal network bandwidth per GPU, require high-precision simulations since large-scale experiments are cost-prohibitive.

Host Architecture Design: Evaluating different host configurations for each type of GPU is essential to determine the optimal number of GPUs per host and the best intra-host interconnect. Simulations help answer these questions without the need to physically build and test expensive hardware prototypes.

Cost-effective validation of optimizations. In addition to hardware evaluations, simulations are indispensable for validating new optimization techniques during model development and system upgrades. This involves:

Parameter Tuning: Testing a variety of model parameters and training framework settings is critical to achieving optimal performance. Simulations allow for quick and inexpensive parameter tuning.

Evaluating New Mechanisms: As innovative enhancements—such as new training frameworks, collective communication methods, and network congestion control algorithms—are introduced, simulations provide a low-cost method for evaluating their effectiveness in realistic settings. This ensures that new mechanisms can be thoroughly tested without the expense of large-scale deployments.

Development of a unified simulation framework. Given the diverse needs for simulations across different components and layers of the AI infrastructure, a unified simulation framework is essential. Building multiple specialized simulators for individual parts of the system often results in inconsistent or incomplete insights. For example, a simulator focused solely on GPUs may overlook critical aspects of network performance, leading to inaccurate conclusions.

Our goal is to develop a unified simulator that addresses all these requirements in a single platform. This unified approach will enable consistent, high-precision simulations across different layers of the AI infrastructure, ensuring that teams can validate new designs and optimizations accurately and efficiently.

2.3 Our Goals

Generating workloads that reflect real-world training. To achieve accurate simulation results, realistic input sources—capturing the detailed behaviors of training frameworks—are essential. Simply estimating workload based on the required floating-point operations is too coarse-grained. Some approaches, like Chakra [53], improve this by using trace-driven methods to extract function-level data from PyTorch Execution Trace. However, this only works for LLMs with the same parameters and scale, limiting the ability to simulate new models or configurations.

Goal 1: We need a flexible and precise workload generator that can handle various models, parameters, and scales.

High-fidelity communication simulation. Classical network simulators, such as NS-3 [47] and OMNET++ [55], offer packet-level network behavior simulations but don't address the collective communication used in distributed LLM training. To maximize performance, collective communication libraries (e.g., NCCL) apply various optimizations that affect traffic patterns. Simulating these from scratch can lead to low fidelity.

Goal 2: We need a high-precision collective communication simulator that incorporates key optimizations and enhancements.

High-fidelity computation simulation. Current solutions like GPGPU-Sim [2] simulate GPU kernel computations at a detailed level but are too time-consuming for large-scale LLM simulations. Other approaches, such as ASTRA-sim [45], fail to support different GPUs or lack the necessary precision.

Goal 3: We need an efficient computation simulator that delivers both precision and scalability for large-scale simulations.

Fast simulation speed. Using a combination of current methods (*i.e.*, PyTorch trace generator with ASTRA-sim), simulating a single iteration of GPT-3 training with 128 GPUs can take an entire day, while the same task on real hardware takes just two seconds. Efficiency is critical to scale simulations for practical use.

Goal 4: The simulator must not only meet Goals 1-3 but also be scalable and capable of running large-scale LLM simulations efficiently.

3 The SimAI Simulator

3.1 SimAI Overview

Figure 1 illustrates the key components of SimAI. Each simulation request includes detailed information about the training process, such as the model itself and parameters, training framework configurations, CCL parameters, and the intra/inter-host network topology.

Workload Generator (SimAI-WG) generates realistic workloads for each simulation request (§3.2). The output, called a workload description file, outlines algorithm modules, collective communication operations, and their dependencies.

The workload file is then processed by the **Execution Engine**, which simulates the execution of both computation and communication operations as discrete events. We utilize the **Computation Simulator** (SimAI-CP) and the **Communication Simulator** (SimAI-CM) to simulate computation and

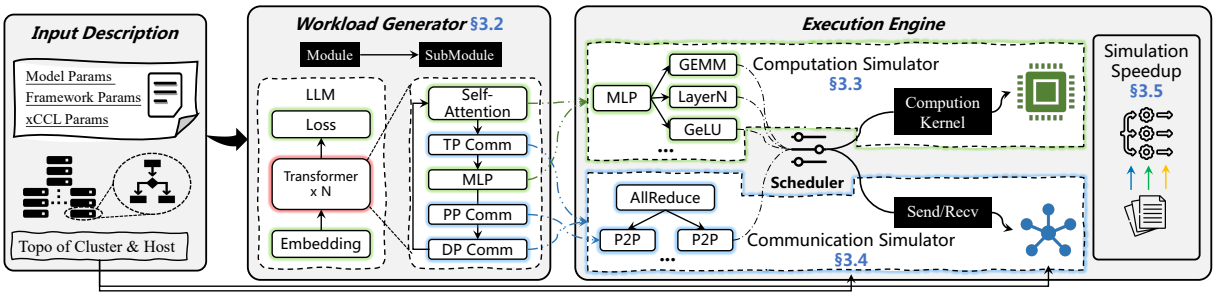


Figure 1: The SimAI architecture

communication tasks, respectively. SimAI-CP transforms submodules into detailed kernels, providing precise computation simulations using a self-built, fine-grained operation library (§3.3). SimAI-CM integrates parts of NCCL, breaking down each collective communication into peer-to-peer operations to deliver accurate communication simulation results (§3.4). Additionally, we implement multi-threaded acceleration and lock-free global context sharing to boost simulation speed further (§3.5).

3.2 Workload Generator

§3.2.1 outlines the process for generating precise workload files for a given model. To further maximize the benefits for real-world training scenarios, selecting a representative benchmark suite is crucial, as discussed in §3.2.2.

3.2.1 Generating workload files

Generating precise workload files is essential, as even small discrepancies between the workload file and actual training execution can lead to low simulation fidelity. While trace-based methods are highly accurate, they require physical clusters to run. SimAI-WG takes a different approach by "hijacking" existing training frameworks to produce workloads identical to real tasks. We have implemented this for Megatron and DeepSpeed, two of the most popular frameworks in our production environment.

Generating workloads using a single host. Without access to a large-scale GPU cluster, the key challenge for this "hijacking" approach is running it on a single host while considering the interactions of multiple peer hosts. To address this, we made two modifications to the framework and NCCL:

- The framework is tricked into believing it runs in a cluster with the target number of GPUs. The inter-host topology is also set to simulate a universal cluster configuration.
- All real communication in NCCL is skipped. For workloads involving pipeline parallelism, SimAI-WG must be configured with the appropriate rank number.

This allows the training framework to generate sequences of computation and communication operations, including algorithm submodules and collective or peer-to-peer communications. Table 1 provides examples of the typical operations used in LLMs. However, these operations lack dependency specifications, which need further specifications.

	Submodules	**Kernels**
Algorithm submodules & kernels	Grad_gather	Grad_gather
	Embedding	Embedding★
	Attention_forward	Layernorm★
		Attention_QKV★
		Core_attention△
		Attention_linear★
	Attention_backward	Attention_backward
	Mlp_forward	Layernorm★
		Mlp_linear_4h△
		GeLU/Swiglu★
		Mlp_linear_h△
	Mlp_backward	Mlp_backward
	Layernorm_post	Layernorm_post★
	Logits_parallel	Logits_parallel△
	Grad_param	Grad_param★
Collective & peer-to-peer communication operations	Allreduce	Write & write_with_imm operations
	Allgather	Write & write_with_imm
	Reducescatter	Write & write_with_imm
	Send	Send
	Recv	Recv

Table 1: List of computation and communication operations. ★ denotes memory-bandwidth-intensive operations and △ denotes computation-intensive operations.

Defining operation dependencies. Since computation and communication operations overlap during execution, we embed dependency information in the workload file to reflect this. Dependencies are determined based on the parallelization framework used and are validated using Nvidia Nsight Systems on a 1,024-GPU cluster. Figure 2 illustrates an example of dependencies when using TP, DP, and PP parallelisms. Solid arrows indicate a strict "happened-before" relationship, while dotted arrows represent overlapping operations where communication begins after the computation has started. The figure demonstrates how Megatron's attention and MLP backward phases benefit from overlapping optimizations. For example, the *MLPbackward* submodule starts only after the *Attentionbackward* submodule and the `AllReduce` operation are complete.

We decoupled the workload from the training framework for independent simulations. Instead of embedding simulation code directly into frameworks like Owl [9], we used SimAI-

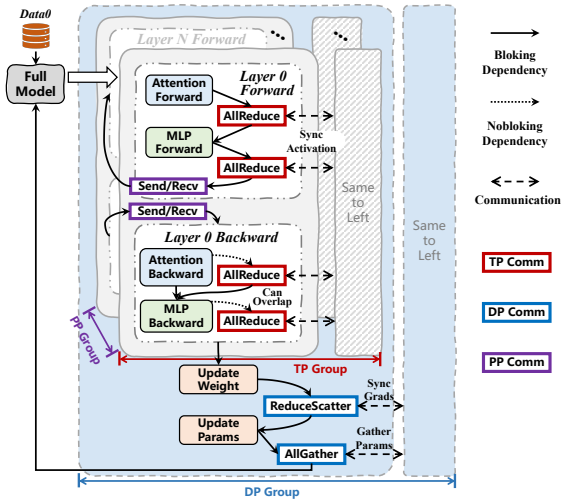


Figure 2: Demonstration of the dependencies among the algorithm submodules and the communication operations.

Size of model parameters	Percentage
100B ~ 500B	41.10%
65B ~ 75B	35.50%
7B ~ 34B	23.40%
Task scale (#GPUs)	**Percentage**
< 512	17.60%
512 ~ 1024	47.05%
> 1024	35.29%

Table 2: Proportions of tasks with different parameter sizes and scales.

WG to extract model workloads and simulate them within the SimAI execution engine. This approach avoids the substantial overhead of modifying training frameworks, which requires extensive changes to monitoring tools, data pipelines, and other critical components. By implementing core algorithms in SimAI-WG, we gain flexibility while understanding how different frameworks generate communication and computation operators.

Communication in large language models. In LLM training, metadata exchanges and barrier operation (under 1 KB) are negligible and not a focus of our simulations. Communication time is primarily introduced by various parallelism techniques, such as TP, PP, EP, and DP. TP, PP, and EP typically have fixed communication patterns and volumes, unaffected by cluster size, involving medium-sized messages ranging from tens to hundreds of megabytes. In contrast, the communication scope of DP expands with the increased cluster size, often involving gigabyte-scale collective communications (such as `AllGather` / `ReduceScatter`) across hundreds to thousands of nodes.

3.2.2 Determining the model-level benchmark suite

As a unified simulator designed to provide comprehensive evaluations of end-to-end performance in real-world LLM training scenarios, it's essential to establish a series of model-level benchmarks, referred to as the benchmark suite.

Model and framework selections. Table 2 shows the distribution of model parameters and task scales over the past six months. The models fall into three categories—small, medium, and large—each making up over 20% of the total. More than 94% of LLMs are variants of GPT-3 or LLaMA, with Megatron and DeepSpeed being the dominant frameworks. Although Megatron usage is on the rise, covering all models is impractical. Instead, we include a selection of open-source LLMs of varying sizes.

Parameter selections. We focus on the parameters that impact workload patterns:

- Model Parameters (e.g., hidden_size, num_layers, seq_len, etc.)
- Framework Parameters (e.g., world size, zero level, reduce_bucket_size/allgather_bucket_size, parallelization strategies like TP, PP, DP, and SP)

Based on the statistics in Table 2, we chose a minimal set of benchmarks that cover typical settings without exploring all possible combinations, as detailed in Table 3. Internal reports indicate that model architects frequently adjust framework parameters to optimize performance or precision. Users can also generate custom benchmarks with SimAI-WG as explained in §3.2.1.

We believe this benchmark suite accurately represents the real LLM training workloads commonly used by our customers. Unless otherwise specified, the benchmarks in this suite will be used for evaluation and discussion throughout the rest of the paper.

3.3 Precise Computation Simulation

Precisely simulating existing GPUs. As explained in § 3.2, by running the mocked framework on a single host, we obtain the detailed submodule workflow. For simulations targeting existing GPUs, SimAI-WG outputs the execution times for all submodules on a host with the corresponding GPU. Since each GPU in practical LLM training is dedicated to a single task, we can accurately simulate the entire computation procedure by following the workload file and filling in the execution times for each submodule. As shown in §4.3, the simulation precision ranges from 96.9% to 99.5%.

We maintain an operation database in SimAI-CP that records the execution times for all submodules in the benchmark suite. Table 4 lists common computation operators and their execution times on various GPUs and configurations. For workloads outside the benchmark suite, specific GPU tests are required to gather additional data.

Fine-grained kernel simulation. While submodule-level simulation works in many cases, it may not be suitable for all scenarios, especially when new parallelization strategies or optimizations reorganize or refine kernels for better performance. In such cases, fine-grained simulation is necessary.

Model		Model_hyperparameter				Framework	Parallel Parameter		Ds_config					
Name	Parameter size	Hidden size	Layers	Sequence length	FFN size	Name	TP	PP	Zero level	Reduce bucket size	Allgather bucket size	Prefetch bucket size	Max live parameters	Param persistence threshold
GPT-3 6.7B	6.7B	4096	32	2048	16384	Megatron	1	1	-	-	-	-	-	-
GPT-3 13B	13B	5120	40	2048	20480	Megatron	2	1	-	-	-	-	-	-
GPT-3 175B	175B	12288	96	2048	49152	Megatron	8	8	-	-	-	-	-	-
LLaMA 65B	65.2B	8192	80	4096	28672	Megatron	8	2	-	-	-	-	-	-
Llama3 405B	405B	16384	126	8192	53248	Megatron	8	16	-	-	-	-	-	-
LLaMA 7B	6.7B	4096	32	4096	11008	Deepspeed	1	-	2	1.00E+09	1.00E+09	-	-	-
LLaMA 65B	65.2B	8192	80	4096	28672	Deepspeed	1	-	3	1.00E+09	-	1.00E+09	6.00E+08	1.00E+06

Table 3: The SimAI benchmark suite v1.0

Operations / GPUs_ConfigParams	Embedding	LayerNorm	...	MLP-Linear
A100_Params1	3425(μs)	715(μs)	...	...
A100_Params2	3157(μs)	695(μs)	...	...
...	...	...	...	...

Table 4: SimAI-CP database format

We have designed a module-kernel converter to break down each submodule into smaller kernels, which are then tested on different GPUs. The third column in Table 1 shows an example of kernels used in different submodules. This further enriches the operation database, enabling precise simulation of advanced optimizations and new features.

Supporting unreleased GPUs. As a simulation service for CSPs, there is strong demand for simulating GPUs that have not yet been released. Decision-makers need to evaluate whether purchasing new GPUs is worth the investment, considering factors like budget, host architecture, and network architecture.

Although physical access to unreleased GPUs is not possible, we may have access to core specifications or a specification sheet. An initial approach might be to estimate computation times by scaling known values from existing GPUs, but this often results in significant inaccuracies—up to 25.1% deviation.

Our analysis shows that different kernels have different performance bottlenecks, typically falling into two categories: computation-intensive or memory-bandwidth-intensive. For example, the Gemm kernel, used for updating the KV cache, is memory-bandwidth-intensive, while flash attention is computation-intensive. Detailed classifications are listed in Table 1.

To improve accuracy, we propose using two equations tailored to these kernel types, based on data from our operation database and the Roofline performance model [58]. These equations, based on measured execution times of compute-bound and memory-bound kernels (subscript Comp_Known and Mem_Known, respectively) in the existing environment, allow us to calculate execution times for these kernels on new GPUs or configurations (subscript Comp_New and Mem_New, respectively):

- For computation-intensive kernels:

$$\text{Time}_{\text{Comp\_New}} = \frac{\text{FLOPS}_{\text{Comp\_New}}}{\text{FLOPS}_{\text{Comp\_Known}}} \times \text{Time}_{\text{Comp\_Known}}$$

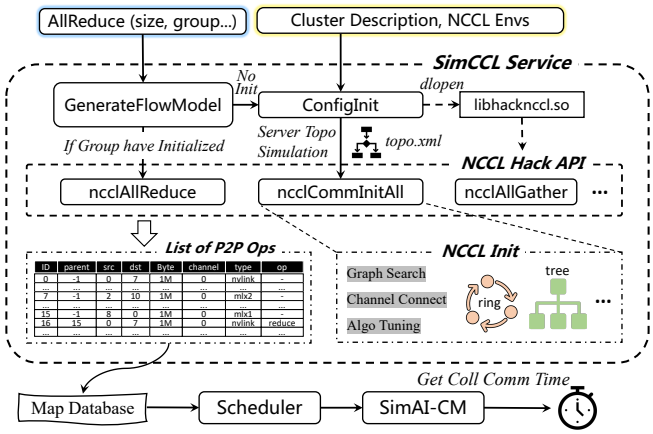


Figure 3: Illustration of how SimAI transforms a collective communication operation to a list of peer-to-peer communication operations.

- For memory-bandwidth-intensive kernels:

$$\text{Time}_{\text{Mem\_New}} = \frac{\text{Bandwidth}_{\text{Mem\_New}}}{\text{Bandwidth}_{\text{Mem\_Known}}} \times \text{Time}_{\text{Mem\_Known}}$$

To ensure accurate results, we recommend using GPUs with similar architectures as baselines, rather than generalizing across different vendors. In our experiments, we use the Nvidia A100 as the baseline for calculating kernel execution times on other Nvidia GPUs. We refer to this method as SimAI-CP-Model.

3.4 Precise Simulation of Communication

Training frameworks rely on collective communication libraries, with NCCL [36] being the most widely used in production. NCCL translates collective communication operations (e.g., `AllReduce`, `AllGather`, `ReduceScatter`) into network-level operations (e.g., `Send`, `Receive`). This complex process involves selecting optimal algorithms based on factors like the number of nodes, message size, and configured parameters. Even small mismatches between practical execution and simulation can cause significant deviations in results. To accurately reproduce these algorithm selections, SimAI integrates key procedures from NCCL directly.

Reproducing NCCL's key procedures. SimAI-CM uses a modified version of NCCL, called SimCCL, to intercept

key operations. SimCCL captures the initialization and core interfaces of collective communication to generate peer-to-peer communication lists.

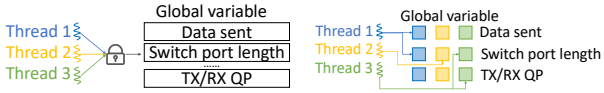
Since simulations typically run on a single host, SimCCL employs a "hijacking" technique to simulate the detailed peer-to-peer operations, similar to the approach discussed in §3.2.1, but at the collective communication level. Here's how SimCCL modifies NCCL's behavior:

1. *NCCL Initialization*: SimCCL intercepts the *NcclCommInitAll* function using the libhacknccl.so in Figure 3, creating virtual communicators for each GPU. This makes the system behave as though it's running in a real multi-GPU cluster, allowing for socket connections and data exchanges during the initialization phase. In the bootstrap and network initialization stages, multiple virtual communicators are created in the same communication group, while only one communicator is actually created.
2. *Topology Discovery*: Instead of searching actual PCIe devices, SimCCL reads a user-specified topology file that defines GPU, NIC, and PCIe switch configurations. Each virtual communicator processes the topology independently and no synchronization is required.
3. *Intra-Host Communication Channel Creation*: SimCCL sets up channels between virtual communicators within the host and stores the details in an isolated information table.
4. *Inter-Host Communication Channel Creation*: SimCCL bypasses gathering information from other GPUs using `AllGather` operations, as it already has information on all hosts. It creates inter-host channels directly.
5. *Collective Communication Transformation*: SimCCL intercepts collective communication calls, reconstructing the operations to trace lower-level communications. It skips actual data transfers and captures inter-GPU communication events, including data size, sender and receiver ranks, and routes, to simulate RDMA-layer behavior.

Supporting all NCCL parameters. SimCCL reflects the communication behavior for the vast majority of NCCL parameters [37]. SimAI-CM has been enhanced to support specific features like PCI × NVLink (PXN). PXN allows a GPU to use a non-local NIC (in the same node) through NVLINK connections for data transfers. This design is typically employed in rail-optimized network topologies, enabling cross-node network traffic to remain on the same rail (single-hop switch) to achieve message aggregation and network traffic optimization. By setting an appropriate NCCL_P2P_PXN_LEVEL, SimCCL can recognize these PXN traffic patterns and reflect them in the output FlowModel. For instance, if rank 1 sends data to rank 8 via rank 0 as an intermediary, the FlowModel represents this as two separate flows: 1->0 and 0->8. Subsequently, SimAI-CM extracts these patterns and simulates them accordingly.

Source file	Description	Lines of code
init.cc	The initialization procedures	364
topo.cc	The topology discovery, search, and synchronization	52
topo.xml	A custmoized topology file	~100
enqueue.cc	Chunk segmentation and task scheduling of collective communication	56

Table 5: Efforts of modifying NCCL.



(a) Multi-thread implementation. (b) Lock-free optimization

Figure 4: Multi-thread implementation and optimization in SimAI.

The SimCCL module is designed to emulate the communication operation processing workflow in NCCL or other CCL. It focuses on transforming collective communication operations into a set of easily interpretable point-to-point communications, without incorporating actual data verification or integrity checks. This approach generally does not affect the end-to-end training process, as it doesn't lead to exceptional workflows due to the absence of real data. In expert parallelism (EP), the gating module's token distribution is influenced by data values. In the simulation, we assume a balanced distribution, which has minimal impact on the simulation results.

Porting efforts. It took us over 10,000 lines of coding efforts for building the blocks shown in Figure 1. The majority of the efforts are intended for repeated use except for the SimCCL module. For example, to support a different version of NCCL or a new CCL, we need to re-adapt the SimCCL module. However, our design does not incur intrusive modifications on the original CCL codebase. As shown in Table 5, only 572 lines of codes (LOC) are essential based on the original NCCL codebase.

3.5 Large Scale Simulation Speedup

At the beginning of SimAI's development, simulating a single iteration of GPT-3 training with 128 GPUs took over 24 hours, compared to just two seconds on a physical GPU cluster. This challenge is similar to AstraSim, which also uses NS-3 for network simulation. To enable simulations of AI infrastructure with more than 1,000 GPUs, we implemented multi-threaded acceleration for SimAI-CM.

Several approaches have been proposed to speed up network simulations. For example, parallel discrete-event simulation (PDES) [10, 18] and UNISON [3] distribute network topologies across multiple logical processes, each running on separate CPU cores. We chose UNISON [3] for three key reasons: (1) It is open-source and builds on NS-3. (2) It automates the partitioning of network topologies and schedules each task to the appropriate thread. (3) It has superior scalability, as demonstrated in their evaluations.

Lock-free sharing of global variables. However, integrating UNISON into SimAI presented a significant challenge. As

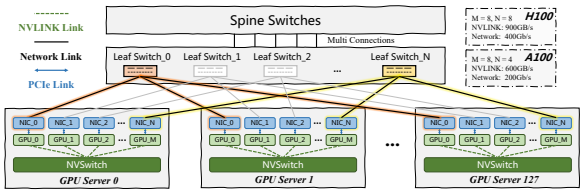


Figure 5: Multi-rail network topology of cluster H100. For cluster A100, two GPUs share a single NIC

the simulation scale increased, a large number of global configurations and contexts were shared across threads. Updating shared data structures with global locks, even atomic ones, caused performance bottlenecks. As shown in Figure 4(a), these global variables track metadata on inter-host communications, such as data volumes between nodes and queue lengths at switches. Atomic operations on these variables slowed down concurrent thread execution.

To solve this, we restructured SimAI to run without global locks. We observed that most global variables were accessed by specific threads. By managing these variables in a thread-independent way, we eliminated the need for global locks. Figure 4(b) illustrates how SimAI separates these variables, enabling lock-free operation. Since each node's simulation runs on a single thread, we isolate global variables in a node ID-indexed table. This allows threads to access the relevant data without locking, reducing the performance issues caused by concurrency. Our lock-free optimization resulted in a 23x speedup compared to the single-thread version and a 15% improvement over the multi-threaded version.

4 Evaluation

In this section, we evaluated SimAI based on the precision and scalability of simulations, using the open-source simulator ASTRA-sim for comparison. Furthermore, results from real-world clusters are used as the ground truth.

4.1 Testbeds and Benchmarks

We utilized two testbeds that share a similar multi-rail, fat-tree, inter-host RoCEv2 network, reminiscent of the Alibaba HPN architecture [44], as illustrated in Figure 5. Cluster A100 consisted of 128 GPU hosts, each equipped with eight NVIDIA A100 GPUs [1] and four Mellanox ConnectX-6 NICs (2×100 Gbps). Cluster H100 included 128 GPU servers, each with eight NVIDIA H100 GPUs [13] and eight Mellanox ConnectX-7 NICs (2×200 Gbps). The intra-host interconnect used NVLink, with bandwidths of 600 GBps for A100 and 900 GBps for H100. Both the simulations and physical clusters shared identical intra-host and inter-host network topologies.

The benchmarks were derived from the SimAI benchmark suite. We enhanced ASTRA-sim with a workload adapter that parsed workload files generated by SimAI-WG. For the physical GPU cluster, the benchmarks involved real LLM tasks that matched the model, framework, and NCCL parameters of the workload files. Our evaluation spanned cluster sizes of 128, 512, and 1024 GPUs. We executed all workloads, but due to space constraints, we only presented performance metrics for three representative models: GPT-3 13B, LLaMA 65B, and GPT-3 175B.

4.2 Precision of Communication Simulation

Figures 6 and 7 showed the performance of intra-host and inter-host collective communication operations across different message sizes (ranging from 8MB to 8GB) and cluster scales. In a multi-rail architecture [59], a collective communication group consists of the ranks on the same rail to minimize the latency across ranks and achieve optimal performance.

From Figure 6, we observed that SimAI was significantly more accurate than ASTRA-sim and closely aligned with the ground truth. The average deviations for SimAI and ASTRA-sim were 3.9% and 74.8% for A100, and 2.3% and 51.7% for H100, respectively. Notably, for `AllGather` and `ReduceScatter` operations on A100, ASTRA-sim achieved its best performance with 8GB messages, yet diverged from reality by over 21.8%, with error rates reaching up to 210.5% under adverse conditions. While there was slight improvement in the `AllGather` operation, ASTRA-sim still showed deviations of 204.3% and 35% for 8MB and 8GB messages, respectively. This discrepancy arose because small message performance is more susceptible to inaccuracies, as we did not simulate the runtime software (e.g., libibverbs) and the NIC pipelines. ASTRA-sim made significant simplifications in practical transmission procedures, leading to poor fidelity, particularly in small message scenarios.

Figure 7 presented the outcomes of collective communication between machines, reinforcing the finding that ASTRA-sim struggled to accurately simulate communications with small message sizes, while SimAI remained consistently close to the ground truth. Furthermore, as communication scaled up, the divergence between ASTRA-sim's simulations and the ground truth widened. For example, in an `AllGather` operation with 8MB messages on a cluster of 128 A100 GPUs, ASTRA-sim exhibited a simulation timing error of 45.9%. This error escalated to 530.2% when the scale increased to 512 GPUs. The results for `AllReduce` and `ReduceScatter` operations displayed similar trends as those observed with the `AllGather` operation.

4.3 Precision of Computation Simulation

We tested the precision of the computation simulation of SimAI and compared it with the ground truth of Nvidia A100, H100, and H20. As shown in Figure 8, SimAI-CP provided highly accurate estimations for all computation kernels. The gaps in total execution times were minimal (0.5%-3.1%) between SimAI-CP and the ground truth. However, for SimAI-CP-Model, the deviations were larger, ranging from 13%-15%, especially for the $Attention-QKV$ and $MLP-linear$ kernels. We recommend that users utilize SimAI-CP-Model only in scenarios where a GPU is not available.

Additionally, we were unable to run all computation kernels

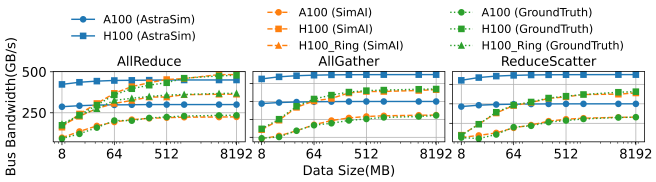


Figure 6: Bus bandwidths of intra-host collective communication operations vary with different message sizes and GPU types. Note that the performance is not related to the cluster scales.

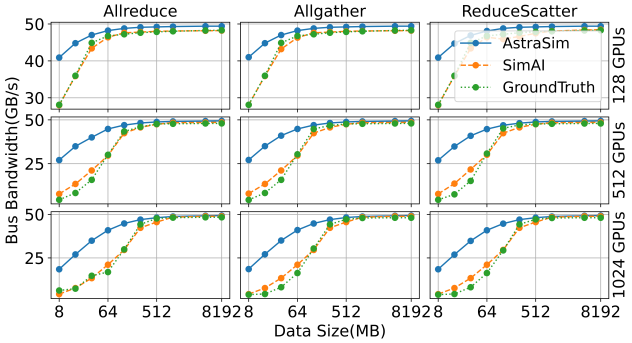


Figure 7: Bus bandwidths of inter-host collective communication operations with different message sizes and cluster scales.

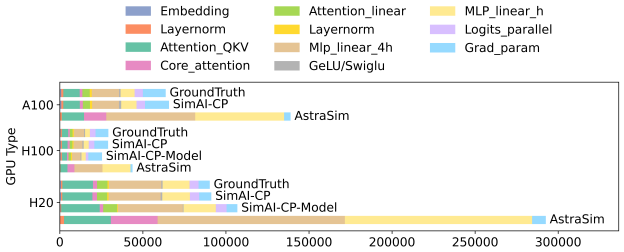


Figure 8: The execution time of the computation kernels of GPT-3 175B on different GPUs.

in SCALE-sim [50], which is used by ASTRA-sim. Only the matrix multiplication kernels ran successfully in SCALE-sim. The numbers for ASTRA-sim in Figure 8 included only these kernels, with execution times multiplied by the effective FLOPS of Nvidia GPUs divided by the effective FLOPS of TPUs. We observed that ASTRA-sim had a significant margin of error in simulating other GPUs, with total errors of 49.8% for H100, 117.9% for A100, and 224% for H20. As discussed in §3.3, the closed internal nature of the GPU computation pipeline caused these methods to fall short in precisely estimating computation times.

Moreover, we attempted to use another computation simulator, Accel-Sim [21], but ultimately failed to run the latest LLM tasks as it only supports PyTorch 0.4, which was released six years ago.

4.4 Precision of End-to-end Simulation

We validated the precision of SimAI in terms of the end-to-end performance on different cluster scales. Figure 9 shows

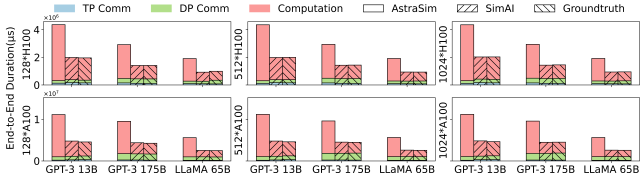


Figure 9: The iteration time of different workloads running on ASTRA-sim, SimAI, and real clusters.

that using SimAI the iteration time of all the workloads is quite close to that of real LLM tasks in real clusters. The gap is less than 4% even under the scale of 1024 GPUs. However, ASTRA-sim is imprecise in the iteration time for both A100 and H100 GPUs. The deviations between SimAI and ground truth are less than 3.9%, 36.1× better than ASTRA-sim. Furthermore, with the increase of the model size (i.e., from GPT-3 13B to GPT-3 175B), the precision of SimAI increases accordingly. It is because the message sizes of the collective communication operations of GPT-3 175B are larger, whose simulation is more precise as validated in §4.2.

5 In Production Benefits

5.1 Guiding Host Design for New GPUs

H100. In 2023, hosts equipped with A100 GPUs were designed to include eight A100 GPUs and four 2×100Gbps NICs (Mellanox-CX6), allowing each GPU to utilize 100Gbps of network bandwidth. With the release of the H100 at GTC 2023 [12], the performance of a single GPU significantly improved. This raises an important question: How much network bandwidth is needed to (1) fully utilize the enhanced performance of the H100 while (2) minimizing overall capital expenditure?

Answering this question is challenging because building multiple engineering prototype hosts with varying configurations is costly and time-consuming. Simply expanding network capacity in proportion to the increase in FLOPS is too simplistic to provide practical guidance.

To inform server design decisions, we utilized SimAI to simulate the performance of a training cluster with 1,024 H100 GPUs and varying network capacities. We tested network equipment configurations of 100Gbps, 200Gbps, and 400Gbps. As shown in Figure 10, the performance of the H100 cluster improves with increased network bandwidth, aligning with our expectations. Upgrading from 200Gbps to 400Gbps results in a 19% performance gain, which remains significant relative to the associated capital expenditure. These simulation results contributed to our final host design decision.

We did not simulate configurations with higher bandwidth than 400Gbps because the PCIe bandwidth of the H100 is 512Gbps (PCIe Gen5×16), indicating that the practical upper limit for a single H100 is approximately 400Gbps.

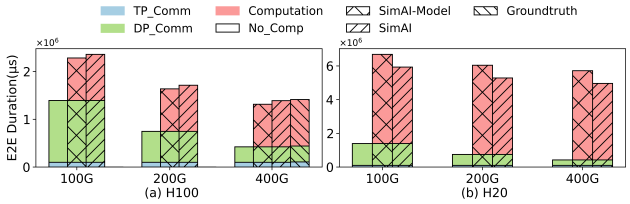


Figure 10: The iteration time of GPT-3 175B with different network bandwidths.

In production, hosts with H100 GPUs are designed to feature eight H100s and eight 2×200Gbps NICs (Mellanox-CX7). Comparatively, the simulation precision using SimAI-CP is over 98%, while SimAI-CP-model shows a precision that is more than 5% lower. Thus, we believe SimAI-CP-model could serve as a viable option when the target GPU type has not yet been released.

H20. With the release of H20 [14] in November 2023, we again used SimAI to guide host design. As shown in Figure 10, since the computational performance of H20 is lower than that of H100, the required network bandwidth also decreases. When network bandwidth is increased from 100Gbps to 200Gbps, end-to-end training performance improves by 11%. However, increasing from 200Gbps to 400Gbps yields only a 6% performance gain.

After thoroughly assessing the investment in networking equipment against performance improvements, we decided to equip each H20 with 200Gbps of network bandwidth (i.e., eight H20 GPUs and eight CX6 NICs per host). Additionally, using SimAI-CP-Model for computation simulation yielded similar conclusions, although with lower precision. This indicates that even for out-of-stock GPU devices where SimAI-CP cannot be used, SimAI can still provide meaningful results.

5.2 Quantifying Scaling-up Benefits

During the training of large language models (LLMs) in production, end-to-end performance is significantly influenced by model and framework parameters, particularly parallelization strategies. Using brute force to explore all parameter combinations in practical training clusters can result in considerable overhead. However, with SimAI, we can automatically identify optimal parameter settings.

In collaboration with the LLM team and the training optimization team, we conducted comprehensive simulations across various parameters. One surprising finding relates to the tensor parallelism (TP) configuration. Conventional wisdom suggests that using more GPUs in a single TP group can reduce the memory and computation load per GPU but may increase communication overhead. Typically, production practices set the TP group size to match the number of GPUs within a host connected via NVSwitch, yet this lacks quantitative analysis.

To understand the trade-offs associated with this parameter setting, we performed experiments within SimAI using GPU counts of 8, 16, 32, and 64, all interconnected with NVSwitch. For each configuration, we ran training tasks on three different models, aiming to identify performance trends and optimal setups. Figure 11 illustrates the following key points:

1. In an 8-GPU host, the optimal TP sizes for GPT-3 13B, LLaMA 65B, and GPT-3 175B are 4, 8, and 8, respectively.
2. For any specific TP size, performance improves with more GPUs in the host.
3. Even with a larger GPU count, performance may decline if the TP size is inappropriate.

These findings offer two valuable guidelines for practical LLM training:

1. The TP parameter should accommodate the entire model layer. However, increasing the TP size can reduce end-to-end throughput; a better strategy is to set up more data parallel (DP) groups for parallel training.
2. GPU host design must consider the evolving nature of LLMs. If a layer's maximum size is known, the number of GPUs within a host should be determined accordingly, prioritizing enhanced scale-out performance.

6 Experience

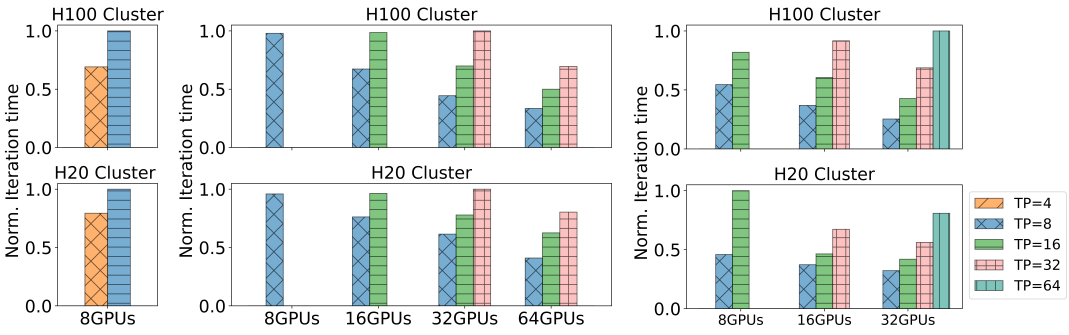
6.1 Evolution of Simulator

Maintaining precision from the very beginning. Initially, we used ASTRA-sim as the foundation for our simulator prototype. Our first challenge was to assess the training performance of emerging large language models (LLMs) like GPT-3 and LLaMA. However, suitable traces or workloads for simulating these models were unavailable, and the built-in DNN [56] workload in ASTRA-sim only provided a rough approximation, leading to significant discrepancies in simulations—ranging from 92.1% to 143.9% across different testbeds. This made it impractical to base our simulation model on the built-in workload.

To ensure accurate LLM workload simulations, we decided to build SimAI-WG from the ground up. We began by analyzing the open-source code of frameworks like Megatron [52] and DeepSpeed [46] to reconstruct their workflows in our simulator. While this improved precision, it had significant drawbacks: (1) It demanded extensive human resources, requiring our engineers to thoroughly understand each framework's components and execution details, which could take months. (2) Keeping our models updated proved challenging due to the rapid evolution of training frameworks, necessitating continuous engineering efforts without any automated update systems.

As a result, we abandoned this approach after our initial internal release and introduced the mock-framework method detailed in this paper. By accurately replicating all sub-modules and their relationships, we achieved substantial improvements in simulation precision.

Measurements from our cluster show that end-to-end training times fluctuate due to factors like network transmission



(a) GPT-3 13B. (b) GPT-3 175B. (c) LLaMA 65B.

Figure 11: Training performance of different hosts and different TP sizes. The x-axis denotes the number of GPUs in a single host. Because of the big model size and limited GPU memory, TP should be larger than 2 for GPT-3 13B and 4 for GPT-3 175B and LLaMA 65B. And we can not run TP = 64 or TP = 48 for GPT-3 175B because the TP size should be the factor of both the number of feedforward neural network (FNN) layers (96) and the world size (1024). We do not let TP exceed the number of GPUs in a single host as the RDMA network is the bottleneck.

and hardware status. Workflow tracking indicates these fluctuations are limited to 5%, with communication time accounting for only 30% of the total. As communication fluctuations contribute negligibly to overall variance, we input averaged parameter values into the SimAI-CP and SimAI-CM components to achieve more consistent performance estimates.

Enhancing computation simulation precision. With a precise workload as input, we still faced a considerable gap between practical execution and simulation results. Early attempts to derive execution times from theoretical models yielded low precision. Additionally, our GPU-acceleration team developed a fine-grained GPU simulator that provides detailed simulation timings for specific operations, but integrating it into our system resulted in excessive simulation latency.

To address this, we focused on the key factors influencing execution times and developed SimAI-CP. Our operation library construction unfolded in two phases. Initially, we extracted key sub-modules from LLMs and measured their execution times, achieving satisfactory precision. However, this approach was limited to specific GPU types. In the second phase, we split sub-modules into finer kernels categorized by computation and memory access intensity. This allowed us to estimate operation times on unavailable GPUs based on performance specifications, enhancing the simulator's applicability.

Improving collective communication simulation precision. We also encountered significant inaccuracies in collective communication simulations when using ASTRA-sim. A critical issue was the inconsistency between the collective communication algorithms in ASTRA-sim and those in NCCL. Even after implementing NCCL's algorithms, results remained unsatisfactory, prompting us to forgo optimizing ASTRA-sim and instead mock key NCCL behaviors from scratch.

The improvements in computation and communication simulations progressed concurrently. When both achieved optimal outcomes, we combined SimAI-CP and SimAI-CM, resulting in a simulator capable of simulating model training with over 93% precision.

Speeding up the entire simulator. After achieving high precision, our final challenge was ensuring efficient simulation execution. Built on NS-3, SimAI underwent a thorough review of existing acceleration solutions, ultimately integrating UNISON [3]. Although we encountered context-access conflicts and crashes during direct operation, implementing global locks to manage inter-thread context access reduced performance. We then analyzed context dependencies and implemented lock-free access, achieving up to a 23× performance improvement, making the simulator practically usable.

6.2 Simulation as a Service

The ultimate goal of developing this simulator is to meet the simulation needs of various teams, transforming it into an easy-to-use simulation service.

Running without GPUs. To achieve high simulation precision, we initially integrated original processing codes (e.g., PyTorch and NCCL) into SimAI. However, this required servers with specific GPUs and environments. To create a genuine simulation service that operates on any host, we decoupled SimAI from the GPU software stack, recognizing and hooking all CUDA-related functions.

Additionally, updating the operation library became an independent procedure. When a new GPU type is available, we conduct a full test of all defined operations on that GPU and update the library as needed.

Task management and scheduling. As a simulation service, SimAI must respond to concurrent simulation requests. To enhance system throughput, we maintain a cluster of hosts. Each request is assigned to an independent simulation task, scheduled on the host with the lightest workload. We use Kubernetes [29] for high scalability.

Parameter tuning. As mentioned in §5, adjusting the TP group size significantly affects end-to-end simulation results. In addition, SimAI allows fine-tuning various parameters influencing model training, including topology, model, and parallel strategy parameters. For example, modifying topology parameters alters the virtual topology within the simulation, impacting network behavior. Adjusting model parameters or parallel strategies affects the computation kernel execution time and communication volume, ultimately impacting training efficiency. Any input supported by SimAI can be fine-tuned to optimize overall performance.

Fine-grained monitoring. For each simulation, we not only generate reports detailing overall training time and throughput but also incorporate extensive monitoring statistics. We developed a front-end system to present the variations of different metrics during simulations.

7 Related work

AI infrastructure simulation. ASTRA-sim [45, 60] is the only comprehensive DNN training simulator designed to simulate the software and hardware co-design stack of distributed training systems. Dally [51] extends ASTRA-sim by incorporating simulations of modern networking hardware (e.g., NVIDIA Quantum [38] and Spectrum switches [39]). SimAI is inspired by ASTRA-sim but introduces significant improvements in workload generation, high-fidelity computation/communication simulation, and scalability.

GPU and computation simulation. Besides the roofline model [58], which provides coarse performance estimates for compute kernels or applications running on a GPU, other works offer instruction-level GPU simulations, such as GPGPU-Sim [2], Accel-Sim [22], SCALE-sim [50], and Gem5-GPU [43]. However, these simulations are very time-consuming for large-scale deployments.

Network simulators and scalability. Existing network simulators fall into three categories: discrete-event simulation (DES) (e.g., NS-3 [47], OMNeT++ [55], OPNET [28]), mathematical model estimation [24, 30, 42], and learning-based approximation (e.g., DeepQueueNet [61], MimicNet [63]). DES simulators offer high-fidelity packet-level simulation, and some works use parallel and distributed event simulation (PDES) approaches [10, 18] to accelerate these simulations. However, PDES often suffers from poor performance and complex configurations. DONS [11] employs a data-oriented design to reduce cache misses in simulation and can be automatically parallelized for intra-host and inter-host scenarios, though it currently lacks support for RDMA transports. UNISON [3] addresses this limitation with fine-grained partitioning and load-adaptive scheduling. SimAI's acceleration is based on UNISON, enhanced with lock-free sharing of global variables. In the future, SimAI will consider simulating network virtualization scenarios to evaluate various virtualization technologies (such as MasQ [15], etc.).

AI benchmarks. The recent surge in machine learning has led to the development of several benchmark suites, but many have limitations in LLM benchmarking. These suites typically include benchmarks for various application domains. MLPerf [31], ML-Bench [27], and AIBench [54] offer comprehensive machine learning benchmarks, while ParaDnn [57], DeepBench [7], and GNNMark [4] focus on deep learning and graph neural networks. However, these benchmarks only cover current applications, which may become obsolete as models evolve rapidly.

Learning-based simulator. Learning-based methods, commonly used as end-to-end performance estimators (EPE) [20, 25, 48, 49, 62, 63], improve scalability by predicting metrics like RTT and packet loss through deep learning models trained on real network traffic. However, these methods require extensive training data, creating significant overhead and limiting their applicability in large-scale simulations. In contrast, SimAI employs discrete event simulators (DES) to mimic the behavior of model training, enabling large-scale simulations and generating valuable data for training learning-based models.

Fault simulation. Fault simulation in model training is a feature worth exploring. Some existing works introduce various types of network faults in the NS-3 simulator to evaluate the impact on the network and the effectiveness of fault tolerance mechanisms [5, 8, 16, 34]. We plan to incorporate simulations for supporting model faults in future work.

8 Conclusion

In this paper, we introduce SimAI, a unified simulator designed to provide accurate and efficient simulations for large language model (LLM) training. SimAI integrates key processes from leading training frameworks and collective communication libraries, achieving an average of 98.1% alignment with real-world results across various test scenarios. To enhance scalability and performance, SimAI incorporates a lock-free, multi-threaded optimization. It is widely utilized in Alibaba Cloud production environments, where it has contributed to the development of valuable operational guidelines.

Acknowledgments

We thank our shepherd Dr. Fang Zhou and the anonymous NSDI reviewers for their constructive comments. This work was supported by the Beijing Outstanding Young Scientist Program (No. JWZQ20240101008).

References

[1] Nvidia a100 tensor core gpu. https://www.nvidia.com/en-us/data-center/a100/. [Online; accessed 4-May-2024].

[2] Bakhoda Ali, George L. Yuan, Wilson W. L. Fung, Wong Henry, and Tor M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, 04 2009.

[3] Songyuan Bai, Hao Zheng, Chen Tian, Xiaoliang Wang, Chang Liu, Xin Jin, Fu Xiao, Qiao Xiang, Wanchun Dou, and Guihai Chen. Unison: A parallel-efficient and user-transparent network simulation kernel. 2024.

[4] Trinayan Baruah, Kaustubh Shivdikar, Shi Dong, Yifan Sun, Saiful A Mojumder, Kihoon Jung, José L Abellán, Yash Ukidave, Ajay Joshi, John Kim, et al. Gnnmark: A benchmark suite to characterize graph neural network training on gpus. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 13–23. IEEE, 2021.

[5] Fedor Chernogorov, Ilmari Repo, Vilho Räisänen, Timo Nihtilä, and Janne Kurjenniemi. Cognitive self-healing system for future mobile networks. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 628–633. IEEE, 2015.

[6] Nvidia mellanox connectx-6 smartnic adapter. https://www.nvidia.com/en-sg/networking/ethernet/connectx-6/. [Online; accessed 4-May-2024].

[7] Deepbench: benchmark operations that are important to deep learning on different hardware platforms. https://github.com/baidu-research/DeepBench. [Online; accessed 4-May-2024].

[8] Emmanuel Effah and Ousmane Thiare. Survey: Faults, fault detection and fault tolerance techniques in wireless sensor networks. *Int. J. Comput. Sci. Inf. Secur.(IJCSIS)*, 16(10):1–14, 2018.

[9] Jason Flinn, Xianzheng Dou, Arushi Aggarwal, Alex Boyko, Francois Richard, Eric Sun, Wendy Tobagus, Nick Wolchko, and Fang Zhou. Owl: Scale and flexibility in distribution of hot content. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 1–15, Carlsbad, CA, July 2022. USENIX Association.

[10] R.M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Series on Parallel and Distributed Computing. Wiley, 2000.

[11] Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Xizheng Wang, Ran Zhang, and Lu Lu. Dons: Fast and affordable discrete event network simulation with automatic parallelization. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 167–181, New York, NY, USA, 2023. Association for Computing Machinery.

[12] Gtc 2023). https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s52226/. [Online; accessed 4-May-2024].

[13] Nvidia h100 tensor core gpu. https://www.nvidia.com/en-us/data-center/h100/. [Online; accessed 4-May-2024].

[14] Nvidia h20 tensor core gpu. https://developer.nvidia.com/blog/accelerating-inference-on-end-to-end-workflows-with-h2o-ai-and-nvidia/. [Online; accessed 4-May-2024].

[15] Zhiqiang He, Dongyang Wang, Binzhang Fu, Kun Tan, Bei Hua, Zhi-Li Zhang, and Kai Zheng. Masq: Rdma for virtual private cloud. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery.

[16] Peng Hu and Jinhuan Zhang. 5g-enabled fault detection and diagnostics: How do we achieve efficiency? *IEEE Internet of Things Journal*, 7(4):3267–3281, 2020.

[17] InfiniBand Trade Association. RoCE v2 Specification. https://cw.infinibandta.org/document/dl/7781, 2014.

[18] Shafagh Jafer, Qi Liu, and Gabriel Wainer. Synchronization methods in parallel and distributed discrete-event simulation. *Simulation Modelling Practice and Theory*, 30:54–73, 01 2013.

[19] Peng Jiang, Christian Sonne, Wangliang Li, Fengqi You, and Siming You. Preventing the immense increase in the life-cycle energy and carbon footprints of llm-powered intelligent chatbots. *Engineering*, 2024.

[20] Charles W. Kazer, João Sedoc, Kelvin K.W. Ng, Vincent Liu, and Lyle H. Ungar. Fast network simulation through approximation or: How blind men can describe elephants. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 141–147, New York, NY, USA, 2018. Association for Computing Machinery.

[21] Mahmoud Khairy, Zhesheng Shen, Tor M Aamodt, and Timothy G Rogers. Accel-sim: An extensible simulation framework for validated gpu modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 473–486. IEEE, 2020.

[22] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. Accel-sim: An extensible simulation framework for validated gpu modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 473–486, 2020.

[23] Xinhao Kong, Yibo Zhu, Huaping Zhou, Zhuo Jiang, Jianxi Ye, Chuanxiong Guo, and Danyang Zhuo. Collie: Finding performance anomalies in RDMA subsystems. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 287–305, Renton, WA, April 2022. USENIX Association.

[24] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050. 06 2004.

[25] Lingda Li, Santosh Pandey, Thomas Flynn, Hang Liu, Noel Wheeler, and Adolfy Hoisie. Simnet: Accurate and high-performance computer architecture simulation using deep learning. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(2), jun 2022.

[26] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. Hpcc: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 44–58, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Yuliang Liu, Xiangru Tang, Zefan Cai, Junjie Lu, Yichi Zhang, Yanjun Shao, Zexuan Deng, Helan Hu, Kaikai An, Ruijun Huang, Shuzheng Si, Sheng Chen, Haozhe Zhao, Liang Chen, Yan Wang, Tianyu Liu, Zhiwei Jiang, Baobao Chang, Yujia Qin, Wangchunshu Zhou, Yilun Zhao, Arman Cohan, and Mark Gerstein. Ml-bench: Evaluating large language models for code generation in repository-level machine learning tasks, 2024.

[28] Z. Lu and H. Yang. *Unlocking the Power of OPNET Modeler*. Unlocking the Power of OPNET Modeler. Cambridge University Press, 2012.

[29] Marko Luksa. *Kubernetes in action*. Simon and Schuster, 2017.

[30] Marco Marsan, Michele Garetto, Paolo Giaccone, Emilio Leonardi, Enrico Schiattarella, and Alessandro Tarello. Using partial differential equations to model tcp mice and elephants in large ip networks. *Networking, IEEE/ACM Transactions on*, 13:1289 – 1301, 01 2006.

[31] Peter Mattson, Christine Cheng, Gregory Diamos, Cody Coleman, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. *Proceedings of Machine Learning and Systems*, 2:336–349, 2020.

[32] Introducing our next generation infrastructure for ai. https://about.fb.com/news/2024/04/introducing-our-next-generation-infrastructure-for-ai/. [Online; accessed 4-May-2024].

[33] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, Rong Liu, Chao Shi, Binzhang Fu, Jiaji Zhu, Jiesheng Wu, Dennis Cai, and Hongqiang Harry Liu. From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 753–766, New York, NY, USA, 2022. Association for Computing Machinery.

[34] Arslan Munir, Joseph Antoon, and Ann Gordon-Ross. Modeling and analysis of fault detection and fault tolerance in wireless sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(1):1–43, 2015.

[35] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[36] Nvidia collective communications library (nccl). https://developer.nvidia.com/nccl. [Online; accessed 4-May-2024].

[37] Environment variables of nvidia collective communications library (nccl). https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html. [Online; accessed 4-May-2024].

[38] Nvidia quantum switch. https://nvdam.widen.net/s/k8sqcr6gzb/infiniband-quantum-2-qm9700-series-datasheet-us-nvidia-1751454-r8-web. [Online; accessed 4-May-2024].

[39] Nvidia spectrum switch. https://nvdam.widen.net/s/mmvbnpk8qk/networking-ethernet-switches-sn5000-datasheet-us. [Online; accessed 4-May-2024].

[40] Nvlink and nvlink switch. https://www.nvidia.com/en-us/data-center/nvlink/. [Online; accessed 4-May-2024].

[41] OpenAI. Gpt-4 technical report, 2024.

[42] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H. Low. Multipath tcp: Analysis, design, and implementation. *IEEE/ACM Transactions on Networking*, 24(1):596–609, 2016.

[43] 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, 2015.

[44] Kun Qian, Yongqing Xi, Jiamin Cao, Jiaqi Gao, Yichi Xu, Yu Guan, Binzhang Fu, Xuemei Shi, Fangbo Zhu, Rui Miao, Chao Wang, Peng Wang, Pengcheng Zhang, Xianlong Zeng, Eddie Ruan, Zhiping Yao, Ennan Zhai, and Dennis Cai. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 691–706, New York, NY, USA, 2024. Association for Computing Machinery.

[45] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 81–92, 2020.

[46] 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*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.

[47] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[48] Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, page 140–151, New York, NY, USA, 2019. Association for Computing Machinery.

[49] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[50] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883*, 2018.

[51] Aakash Sharma, Vivek M. Bhasi, Sonali Singh, George Kesidis, Mahmut T. Kandemir, and Chita R. Das. Gpu cluster scheduling for network-sensitive deep learning, 2024.

[52] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[53] Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, and Tushar Krishna. Chakra: Advancing performance benchmarking and co-design using standardized execution traces, 2023.

[54] Fei Tang, Wanling Gao, Jianfeng Zhan, Chuanxin Lan, Xu Wen, Lei Wang, Chunjie Luo, Zheng Cao, Xingwang Xiong, Zihan Jiang, Tianshu Hao, Fanda Fan, Fan Zhang, Yunyou Huang, Jianan Chen, Mengjia Du, Rui Ren, Chen Zheng, Daoyi Zheng, Haoning Tang, Kunlin Zhan, Biao Wang, Defei Kong, Minghe Yu, Chongkang Tan, Huan Li, Xinhui Tian, Yatao Li, Junchao Shao, Zhenyu Wang, Xiaoyu Wang, Jiahui Dai, and Hainan Ye. Aibench training: Balanced industry-standard ai training benchmarking. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 24–35, 2021.

[55] Andras Varga. *A Practical Introduction to the OMNeT++ Simulation Framework*, pages 3–51. 05 2019.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[57] Yu Wang, Gu-Yeon Wei, and David Brooks. A systematic methodology for analysis of deep learning hardware and software platforms. *Proceedings of Machine Learning and Systems*, 2:30–43, 2020.

[58] 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.

[59] Noah Wolfe, Misbah Mubarak, Nikhil Jain, Jens Domke, Abhinav Bhatele, Christopher D Carothers, and Robert B Ross. Preliminary performance analysis of multi-rail fat-tree networks. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 258–261. IEEE, 2017.

[60] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. Astra-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 283–294, 2023.

[61] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. Deepqueuenet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 441–457, New York, NY, USA, 2022. Association for Computing Machinery.

[62] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. Deepqueuenet: towards scalable and generalized network performance estimation with packet-level visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 441–457, New York, NY, USA, 2022. Association for Computing Machinery.

[63] Qizhen Zhang, Kelvin K. W. Ng, Charles Kazer, Shen Yan, João Sedoc, and Vincent Liu. Mimicnet: fast performance estimates for data center networks with machine learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 287–304, New York, NY, USA, 2021. Association for Computing Machinery.

[64] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion control for large-scale rdma deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 523–536, New York, NY, USA, 2015. Association for Computing Machinery.

9 Appendix

9.1 NCCL environment variables

Table 6 lists all the NCCL environment variables that documented in the official website [37] and whether they are supported in the SimAI simulator. We can see besides those we think are unnecessary for simulation, only four environment variables are not yet supported, which are adaptive routing and Infiniband SHArP related features. We leave them as the future work of the communication simulator of SimAI.

Table 6: NCCL environment variables and how they affects the communication patterns

NCCL environment variable	Description	Support?
NCCL_IB_HCA	Specifies which RDMA interfaces to use for communication	Yes
NCCL_IB_TC	Specifies the InfiniBand Traffic Class to use	Yes
NCCL_IB_QPS_PER_CONNECTION	Number of IB queue pairs to use for each connection between two ranks	Yes
NCCL_NVLS_ENABLE	Enable the use of NVLink SHARP (NVLS)	Yes
NCCL_MAX_ NCHANNELS	Sets the maximum number of NCCL channels per GPU	Yes
NCCL_MIN_ NCHANNELS	Sets the minimum number of NCCL channels per GPU	Yes
NCCL_MIN_NRINGS	Sets the minimum number of NCCL rings per GPU	Yes
NCCL_MAX_NRINGS	Sets the maximum number of NCCL rings per GPU	Yes
NCCL_CROSS_NIC	Controls whether NCCL should allow rings/trees to use different NICs, causing inter-node communication to use different NICs on different nodes	Yes
NCCL_NET_GDR_LEVEL	Allows the user to finely control when to use GPU Direct RDMA between a NIC and a GPU	Yes
NCCL_ALGO	Defines which algorithms NCCL will use	Yes
NCCL_PXN_DISABLE	Disable inter-node communication using a non-local NIC, using NVLink and an intermediate GPU	Yes
NCCL_TOPO_FILE	Path to an XML file to load before detecting the topology	Yes
NCCL_P2P_DISABLE	The NCCL_P2P_DISABLE variable disables the peer to peer (P2P) transport, which uses CUDA direct access between GPUs, using NVLink or PCI.	Yes

Continued on next page

Table 6: NCCL environment variables and how they affects the communication patterns (Continued)

NCCL environment variable	Description	Support?
NCCL_P2P_LEVEL	Define the levelwhen to use the peer to peer (P2P) transport between GPUs.	Yes
NCCL_P2P_DIRECT_ DISABLE	Define and set to 1 to disable direct user buffer access across GPUs.	Yes
NCCL_SHM_DISABLE	NCCL will use the network (i.e. InfiniBand or IP sockets) to communicate between the CPU sockets when SHM is disabled.	Unnecessary
NCCL_SOCKET_IFNAME	Specifies which IP interfaces to use for communication.	Unnecessary
NCCL_SOCKET_FAMILY	The NCCL_SOCKET_FAMILY variable allows users to force NCCL to use only IPv4 or IPv6 interface.	Unnecessary
NCCL_SOCKET_ NTHREADS	Specifies the number of CPU helper threads used per network connection for socket transport	Unnecessary
NCCL_NSOCKS_ PERTHREAD	Specifies the number of sockets opened by each helper thread of the socket transport	Unnecessary
NCCL_DEBUG	Controls the debug information that is displayed from NCCL.	Unnecessary
NCCL_BUFFSIZE	Controls the size of the buffer used by NCCL when communicating data between pairs of GPUs.	Yes
NCCL_NTHREADS	Sets the number of CUDA threads per CUDA block	Unnecessary
NCCL_CHECKS_ DISABLE	Disable argument checks on each collective call.	Unnecessary
NCCL_CHECK_ POINTERS	Enables checking of the CUDA memory pointers on each collective call.	Unnecessary
NCCL_LAUNCH_MODE	Controls how NCCL launches CUDA kernels.	Unnecessary
NCCL_IB_DISABLE	Prevents the IB/RoCE transport from being used by NCCL	Yes
NCCL_IB_TIMEOUT	Controls the InfiniBand Verbs Timeout.	Yes
NCCL_IB_RETRY_CNT	Controls the InfiniBand retry count.	Yes
NCCL_IB_GID_INDEX	Defines the Global ID index used in RoCE mode	Yes
NCCL_IB_ADDR_ FAMILY	Defines the IP address family associated to the infiniband GID dynamically selected by NCCL when NCCL_IB_GID_INDEX is left unset.	Unnecessary

Continued on next page

Table 6: NCCL environment variables and how they affects the communication patterns (Continued)

NCCL environment variable	Description	Support?
NCCL_IB_ADDR_RANGE	The NCCL_IB_ADDR_RANGE variable defines the range of valid GIDs dynamically selected by NCCL when NCCL_IB_GID_INDEX is left unset.	Unnecessary
NCCL_IB_ROCE_ VERSION_NUM	Defines the RoCE version associated to the infiniband GID dynamically selected by NCCL when NCCL_IB_GID_INDEX is left unset.	Unnecessary
NCCL_IB_SL	Defines the InfiniBand Service Level.	Unnecessary
NCCL_IB_AR_ THRESHOLD	Threshold above which we send InfiniBand data in a separate message which can leverage adaptive routing.	No
NCCL_IB_CUDA_ SUPPORT	The NCCL_IB_CUDA_SUPPORT variable is used to force or disable the usage of GPU Direct RDMA	Unnecessary
NCCL_IB_SPLIT_ DATA_ON_QPS	This parameter controls how we use the queue pairs when we create more than one,	Yes
NCCL_IB_PCI_ RELAXED_ORDERING	Enable the use of Relaxed Ordering for the IB Verbs transport.	Unnecessary
NCCL_IB_ ADAPTIVE_ROUTING	Enable the use of Adaptive Routing capable data transfers for the IB Verbs transport.	No
NCCL_CUMEM_ENABLE	Use CUDA cuMem* functions to allocate memory in NCCL.	Unnecessary
NCCL_NET	Forces NCCL to use a specific network, for example to make sure NCCL uses an external plugin and doesn't automatically fall back on the internal IB or Socket implementation.	Yes
NCCL_NET_PLUGIN	Set it to a suffix string to choose among multiple NCCL net plugins	Unnecessary
NCCL_NET_ SHARED_BUFFERS	Allows the usage of shared buffers for inter-node point-to-point communication	Yes
NCCL_NET_ SHARED_COMMS	Reuse the same connections in the context of PXN.	Unnecessary
NCCL_PROTO	Defines which protocol NCCL will use.	Yes
NCCL_COLLNET_ ENABLE	Enable the use of the CollNet plugin.	No
NCCL_COLLNET_ NODE_THRESHOLD	A threshold for the number of nodes below which CollNet will not be enabled.	No
NCCL_TOPO_ DUMP_FILE	Path to a file to dump the XML topology to after detection.	Unnecessary

Continued on next page

Table 6: NCCL environment variables and how they affects the communication patterns (Continued)

NCCL environment variable	**Description**	**Support?**
NCCL_NVB_DISABLE	Disable intra-node communication through NVLink via an intermediate GPU.	Yes
NCCL_GRAPH_ REGISTER	Enable user buffer registration when NCCL calls are captured by CUDA Graphs.	Unnecessary
NCCL_SET_STACK_SIZE	Set CUDA kernel stack size to the maximum stack size amongst all NCCL kernels.	Unnecessary
NCCL_DMABUF_ ENABLE	Enable GPU Direct RDMA buffer registration using the Linux dma-buf subsystem.	Unnecessary
NCCL_P2P_NET_ CHUNKSIZE	The NCCL_P2P_NET_CHUNKSIZE controls the size of messages sent through the network for ncclSend/ncclRecv operations.	Yes