## Scope

**Briefly describe how your contribution fits within the scope of HPC and how it addresses the specific focus of the selected track(s)**

The paper presents Moye, a scalable and high-fidelity simulator for large-scale LLM training on GPU clusters. It fits squarely within HPC because it addresses performance modeling and system simulation for distributed workloads that combine 3D parallelism, collective communication, and cluster-scale execution. Section I motivates simulation for both enumeration and extrapolation of training strategies on expensive large clusters, Table I positions Moye against prior simulators, Section VIII evaluates accuracy and simulation cost up to 8,192 GPUs, and Section IX demonstrates what-if strategy search for GPT-485B on 8,192 GPUs. These contributions align most directly with HPC tracks focused on performance modeling, large-scale system simulation, and AI training infrastructure.

## Motivation and Problem Definition

| | |
|---|---|
| Have you clearly stated the research objective and its scientific or technical importance? | **yes** |
| Have you defined the problem scope and assumptions with sufficient precision? | **yes** |
| Have you provided quantitative evidence or benchmarks to motivate your problem? | **yes** |

**Explain**

The objective is clearly stated in Section I: simulating LLM training for enumeration and extrapolation on massive clusters. Section II.B precisely scopes the three main challenges: acquiring real workloads without full-scale deployment, modeling collective communication without high overhead, and capturing overlap-induced slowdowns. Quantitative motivation is provided in Section II.B, Table II (showing 23%-58% of compute ops overlap with communication) and Figure 3, as well as the 2-hour simulation time for just 128 GPUs using existing packet-level simulators.

## Relation to State of the Art

| | |
|---|---|
| Have you identified and summarized the most relevant and recent prior work? | **yes** |
| Have you discussed how your work differs from and improves upon these prior efforts? | **yes** |
| Have you articulated how your approach addresses gaps in the state of the art? | **yes** |
| Have you verified the accuracy of all references and included DOIs where available? | **yes** |

**Explain**

Section I, Table I, and Section XI summarize recent simulators like Proteus, SimAI, vTrain, and FlexFlow. Table I and Section II.B explicitly articulate gaps, such as the lack of MoE support, analytical memory simulation failures, and the prohibitive overhead of packet-level network simulation. Section III details how Moye addresses these via ex-situ tracing, white-box modeling, and an ML slowdown predictor.

## Contributions

| | |
|---|---|
| Have you enumerated the main contributions, including theoretical, methodological, and empirical advances? | **yes** |
| Have you demonstrated how these contributions advance or generalize existing knowledge in HPC? | **yes** |

**Explain**

Section I enumerates three main methodological contributions: (1) ex-situ tracing for per-device workloads, (2) hybrid white-box collective communication modeling, and (3) a black-box ML model for predicting overlap slowdown. Sections VIII and IX empirically demonstrate that these advances generalize to massive HPC clusters (extrapolating up to 8,192 GPUs) and highly complex workloads (MoE, 3D parallelism), achieving a ~91.8x speedup over state-of-the-art packet-level simulators (Section VIII.B, Figure 8).

## Experimental Methodology

| | |
|---|---|
| Have you described the experimental or simulation environment, including hardware, software, and configuration details? | **yes** |
| Have you validated correctness and discussed sources of measurement variability? | **yes** |
| Have you provided quantitative results that include appropriate baselines and ablation studies where relevant? | **yes** |
| Have you performed and reported statistical significance analysis for performance or accuracy improvements? | **n/a** |
| Have you examined potential barriers to reproducibility (e.g., hardware dependencies, software versions, or configuration settings) and discussed how they are addressed? | **yes** |

**Explain**

Section VII and Section VIII-A describe the implementation and evaluation environment, including PyTorch 2.1, DeepSpeed 0.13.1, Megatron-LM, NCCL 2.18.6, CUDA 12.1, the 256-GPU H800 cluster, the 8-GPU A800 cluster, the RTX 3090 platform, evaluated model families, and baseline simulators; Table VI summarizes the evaluated MoE workloads. Correctness is validated through end-to-end and component-level comparisons in Section VIII-B through Section VIII-E, including Figures 7–13 and Tables VII–XI, but the paper does not explicitly analyze run-to-run measurement variability or repeated-trial uncertainty, so this item is marked no. Appropriate baselines and ablations are reported in Figure 7, Figure 8, Figure 9, and Table VII, with additional module-level evaluations in Figure 10, Figure 12, Figure 13, and Tables VIII–XI. The paper does not report formal statistical significance tests, confidence intervals, or hypothesis testing; instead, it reports prediction errors and error-threshold coverage (e.g., Table XI). Reproducibility barriers are addressed by specifying the hardware, software stack, model settings, and communication environment in Section VII and Section VIII-A, and the paper states that Moye will be open-sourced (Section I).

## Limitations

| | |
|---|---|
| Have you identified the main limitations or assumptions that constrain the proposed approach? | **yes** |
| Have you discussed how conclusions might change under different problem scales, architectures, or workloads? | **yes** |

**Explain**

Section X explicitly states that Moye currently does not support sequence/context parallelism and that its communication modeling intentionally sits between simple alpha-beta models and packet-level simulation; future work is expected to capture additional network-centric factors such as transport protocols more efficiently. The paper also discusses scale sensitivity in Section VIII-B: relative to SimAI, Moye has comparable end-to-end error at small scales (2–3% at 8–32 GPUs), but the error rises to 10.5% at 256 GPUs. Differences across workloads and platforms are also examined in Figure 7 and Figure 13, which cover dense versus MoE models and H800/A800/RTX 3090 settings. Together, these results make clear which conclusions are robust and which remain dependent on scale, hardware, and workload characteristics.

# Scalable and Efficient Simulation of LLM Training

***Abstract*—Simulation offers unique values for both enumeration and extrapolation purposes, and is becoming increasingly important for managing the massive machine learning (ML) clusters and large-scale distributed training jobs. In this paper, we build Moye to tackle three key challenges in large-scale training simulation: (1) tracing the runtime training workloads at each device in an ex-situ fashion so we can use a single device to obtain the actual execution graphs of 1K-GPU training, (2) accurately estimating the collective communication without high overheads of discrete-event based network simulation, and (3) accounting for the interference-induced computation slowdown from overlapping communication and computation kernels on the same device. Moye supports both dense and MoE workloads with high fidelity and further enables GPU memory–aware ex-situ simulation. Moye delivers on average 8% error in training step—∼3x lower than state-of-the-art simulators—for GPT-175B on a 256-GPU H800 cluster with 3D parallelism on Megatron-LM under 3 minutes.**

## I. Introduction

The unprecedented success of large language models (LLMs) has been driven in large part by the large-scale infrastructures that allow the model and training dataset to scale. Organizations have constructed massive clusters with tens of thousands of GPUs to train models with hundreds of billions or even trillions of parameters [1]–[3], in a continuous stride to improve model capabilities.

Training LLMs on such massive scales consumes substantial time and financial resources. It is also inherently complex and rapidly evolving with a myriad of algorithm/model innovations [4], [5], parallelism strategies [6]–[8], kernel optimization ( [9], [10]), and hardware designs [11]. As a result, accurate simulation of LLM training has started to garner attention as an essential building block to effectively manage training infrastructures and systems [12]–[16].

Broadly, simulation is useful for two main purposes: enumeration and extrapolation. In LLM training context, simulation can be used to enumerate combinations of parallelism strategies, optimization techniques, and hardware configurations that are available in the current cluster, to derive the optimal training plan for a given job and to determine efficient resource allocation schedules across jobs [12], [13]. Simulation is also useful (perhaps more so) to extrapolate beyond the currently available resources, which is paramount for strategic decision making such as capacity planning [1], [17] that involve what-if questions with significant impact. For example, what speed-up can be expected by scaling the current cluster by a factor of 5, or by increasing the network bandwidth by 2x? This also greatly facilitates the development of new optimizations, which only need to be prototyped on a small scale for the simulator to extrapolate its potential benefits on a much larger scale.

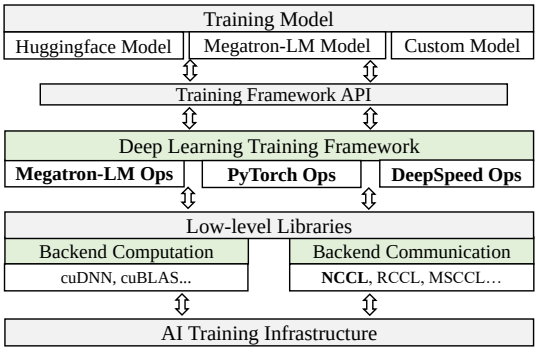



Fig. 1: Overview of the technical stack for LLM training.

Prior work has made salient progress on training simulation [12]–[16], [18]–[21]. Yet, they fall short in three key aspects that hinder their capabilities in supporting both enumeration and extrapolation practically.

First, accurately simulating LLM training with complex parallelisms—data parallelism (DP), tensor parallelism (TP), pipeline parallelism (PP), and expert parallelism (EP) for MoE—requires obtaining the real training workloads: per-device execution graphs encoding computation, communication, and memory events with their dependencies. Existing simulators either rely on *in-situ* tracing that demands full-scale deployment [12], [15], [16], [19], or manually construct workloads without faithfully supporting the full combination of parallelism strategies and framework-specific operations [13], [14], [20], [23], fundamentally limiting *ex-situ* simulation capability (Table I). GPU memory prediction is equally challenging: peak memory is governed by framework allocator-level dynamics that analytical models cannot accurately capture (§II-B), yet accurate memory estimation is a prerequisite for out-of-memory (OOM)-free deployment.

Second, simulating collective communication (CC) is critical because (1) it pieces all the devices and nodes together according to the parallelisms, and (2) its performance can often be the bottleneck [1], [24]–[26]. Most existing work [13]–[16], [19] rely on a coarse-grained $\alpha$-$\beta$ model [27] to simulate the CC time without considering the actual communication patterns and implementation optimizations. On the other hand, more recent work such as SimAI [12] explicitly simulates each peer-to-peer (P2P) send and receive primitive of a CC kernel using a packet-level event-based network simulator such as ns-3 [28]. This fine-grained approach provides high accuracy at the cost of prohibitive overheads even at a medium scale: simulating a 128-GPU job takes SimAI over 2 hours with an optimized ns-3 implementation (§II-B).

Third, overlapping communication and computation opera-

| Simulator | Training Workload | | | 3D Parallelism | MoE/EP Support | Comm. Modeling | Overlap Slowdown |
|---|---|---|---|---|---|---|---|
| | Framework Ops | Ex-Situ Sim. | GPU Mem Sim. | | | | |
| FlexFlow [18] | ✗ | – | ✗ | ✓ | ✗ | $\alpha-\beta$ model | ✗ |
| Astra-sim [14] | ✗ | – | ✗ | – | ✗ | $\alpha-\beta$ model | ✗ |
| TrioSim [22] | ✗ | ✗ | ✗ | – | ✗ | $\alpha-\beta$ model | ✗ |
| SimAI [12] | ✓ | – | ✗ | ✓ | – | Event-driven | ✗ |
| vTrain [23] | – | – | ✗ | ✓ | ✗ | $\alpha-\beta$ model | ✗ |
| Calculon [20] | ✗ | – | Analytical | ✓ | ✗ | $\alpha-\beta$ model | ✗ |
| Proteus [13] | ✗ | – | Analytical | ✓ | ✗ | $\alpha-\beta$ model | – |
| **Moye (ours)** | ✓ | ✓ | Ex-Situ Profiling | ✓ | ✓ | White-box | ✓ |

TABLE I: Comparison of Moye against representative simulators; ✓ indicates full support, ✗ no support, and – partial or conditional support.

tions, an optimization widely used to improve efficiency [1], [29]–[31], incurs non-negligible slowdowns to the computation due to contention for shared resources (e.g., cache, DRAM bandwidth) [13], [32]. This, however, has largely been overlook thus far. Simulating interference-induced slowdown is particularly challenging since it depends on many idiosyncrasies ranging from scheduling logic of the ML framework and the underlying library to hardware-specific implementation optimizations. Much of these details are vendor-proprietary and inaccessible.

In this work, we build Moye, a practical simulation platform for large-scale LLM training. Moye is built with three key design choices. (1) It adopts an *ex-situ* tracing approach to accurately capture training workloads using just a single device, avoiding the need for a full-scale cluster deployment. The key technical idea here is to transform the parallel initialization process at each rank into a sequential one, allowing a single device to act as each rank iteratively to trace the corresponding execution graph and GPU memory footprints. (2) It employs white-box modeling to simulate a CC kernel's performance with four parts: connection setup overhead, intra-server and inter-server transmission, and possible data reduction time. The model is supplemented with exhaustive profiling to obtain key parameters (e.g. chunk size which determines number of rounds of transmission for a message) optimized by CC libraries like NCCL for different hardware features and software configurations that affect the individual components above. This hybrid approach strikes a good balance between accuracy and efficiency. (3) It introduces a black-box ML-based slowdown predictor to model the slowdown caused by overlapping operations. The XGBoost-based predictor relies on categorical features such as transmission protocol and channel configuration in NCCL, to numerical kernel-level performance statistics such as streaming multiprocessors (SM) occupancy and DRAM utilization, to capture their complex interactions.

We implement Moye to support mainstream training frameworks: PyTorch, DeepSpeed [33], and Megatron-LM [17]. Our implementation with ∼10k LoC also includes a suite of automation tools to facilitate workload tracing and runtime profiling. We evaluate Moye on H800 and A800 clusters across a variety of scenarios and models, achieving an average prediction accuracy of 91.4% in training step time, and is up to 3x better than the state-of-the-art. For 256-GPU training of GPT-175B, Moye achieves 90% accuracy in less than 3 minutes. Beyond dense models, Moye provides high-fidelity support for MoE workloads (average error 3.4%) and further enables GPU memory–aware ex-situ simulation. We plan to open source Moye to the community.

## II. Background and Motivation

### A. Large-scale LLM Training

Figure 1 illustrates the layered technical stack underlying LLM training, from high-level models and frameworks to low-level GPU libraries and networked systems. LLM training jobs are executed on frameworks such as Megatron-LM [17] and PyTorch [12], [17], [33]. They implement parallelism strategies combining DP, TP, PP, and EP to efficiently scale model training across large GPU clusters. Chiefly, DP replicates the model and splits the data, TP divides model parameters among GPUs, PP stages model layers for pipelined execution, while EP distributes experts of a MoE model across devices. Collective Communication Libraries (CCLs) then facilitate data transfer across ranks. NCCL is the de-facto CCL in production [12], [17], [33], [34].

### B. Design Challenges

Training simulation has attracted attention recently [12]–[14], [21]. Inspired by prior work, we also separately consider computation and communication, and computation can be accurately simulated by offline profiling on the target device. Yet, to achieve the four design goals, there are three key challenges that prior work has not fully addressed. Table I summarizes existing simulators along these dimensions.

**Challenge 1:** *How to obtain real training workloads on each device without a full-scale deployment?*

Accurate training simulation requires the real *training workload*—the per-device execution graph encoding the dependencies among computation and communication operations together with tensor shapes and data types. In LLM training with complex parallelisms, each rank independently initializes its assigned submodel based on TP, PP, and EP settings, resulting in unique workloads per rank. However, most existing simulators [12]–[14], [22], [23] do not faithfully support the combination of parallelism strategies used in production systems such as DeepSeek-V3 [8], and therefore cannot directly generate realistic per-rank workloads. This fundamentally limits their *ex-situ* simulation capability (Table I), i.e., the ability to explore large-scale scenarios without deploying on the target cluster. Beyond workload fidelity, accurate GPU

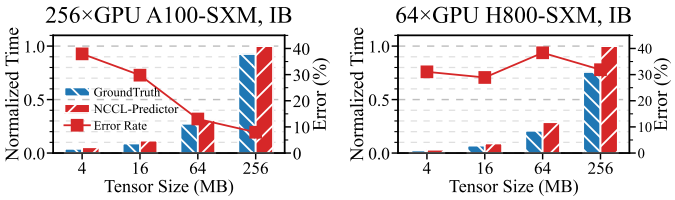



Fig. 2: Comparison of predicted communication time (by NCCL-Predictor) and ground truth for `all-reduce`.

memory prediction—essential for determining whether a given configuration is even deployable (i.e., free of OOM)—poses a similar challenge, because peak memory consumption is dominated by framework allocator-level dynamics (e.g., caching-allocator pooling, activation reuse, and early deallocation) that cannot be accurately derived from model and parallelism parameters alone.

Existing approaches to workload acquisition each suffer from fundamental limitations. Runtime tracing [15], [16], [19] can extract faithful workloads but requires a full-scale deployment, severely limiting the simulator's practical utility at large scales. Alternatively, Proteus [13] and Calculon [20] manually choreograph per-device workloads, yet this approach fails to capture *framework-specific operations*—custom or highly optimized operations whose behavior depends on environment-specific factors (see §VIII-C). GPU memory prediction faces the same challenge: analytical models [13], [20] estimate memory from parameter counts and activation shapes, yet real consumption is governed by allocator-level dynamics—caching-allocator pooling, activation reuse, and early deallocation—that are tightly coupled to the framework runtime and vary across configurations. This mismatch can cause significant errors, reaching up to 2.58× in extreme cases (see §VIII-C), rendering OOM feasibility screening untrustworthy for ex-situ exploration.

**Challenge 2:** *How to accurately simulate collective communication without high overheads?*

Existing approaches fall into two categories. Most simulators [13], [14], [18], [20], [35] adopt an $\alpha - \beta$ model [27] that calculates communication time as $tensor_size/bandwidth + latency$. Even the most refined variant, NCCL-Predictor [35], greatly overestimates the effective bus bandwidth of `all-reduce` operations (Figure 2), because the $\alpha - \beta$ abstraction cannot capture NCCL's internal optimizations across hardware offloading, protocols, and algorithms.

The second approach uses white-box, packet-level simulation [12], [14] with discrete-event network simulators such as ns-3 [28] to faithfully model each P2P transfer within a collective. While more accurate, this is prohibitively expensive: SimAI [12] with an optimized multi-threaded ns-3 takes over 2 hours (7655s) to simulate a single iteration on a 128-GPU cluster. Extrapolating, a design-space sweep of 100 configurations for an 8,192-GPU system would require ∼1.5 years of simulation time (§VIII-B), confining packet-level simulators to small-scale case studies rather than systematic configuration search. The challenge is therefore to retain the accuracy of white-box communication modeling while avoiding packet-level overhead.

[1]Experiments here are conducted on the NVIDIA A800 cluster described in §VIII-A, with software environment aligned to the configuration in §VII.

| Model | Overlapped op | Slowdown | Kernel Type | Slowdown |
|---|---|---|---|---|
| VGG19 | 57.53% | 37.67% | GEMM | 26.95% |
| GPT2 | 51.75% | 10.61% | Sum | 50.83% |
| Llama3-8B | 26.79% | 7.15% | GroupedGEMM | 11.78% |
| Mixtral-8x7B | 23.02% | 10.55% | Grad | 61.37% |

TABLE II: Statistics about (1) the proportion of overlapped kernels in different models and the slowdown factors in a training step, and (2) slowdown factors of some common computation operations. Evaluated with PyTorch (VGG19), DeepSpeed (GPT2), Megatron-LM (Llama3-8B), and Flux [36] (Mixtral-8×7B).

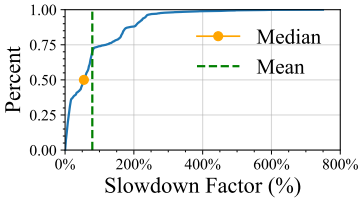



Fig. 3: CDF of slowdown factor of kernels in Mixtra-8x7B.

| GPU | Kernel | CUDA | Slowdown |
|---|---|---|---|
| 3090 | GEMM | 11.8 | 26.4% |
| | GEMM | 12.1 | 39.3% |
| | LayerNorm | 11.8 | 18.0% |
| | LayerNorm | 12.1 | 93.8% |
| A800 | GEMM | 11.8 | 13.4% |
| | GEMM | 12.1 | 17.8% |
| | LayerNorm | 11.8 | 9.1% |
| | LayerNorm | 12.1 | 20.7% |

TABLE III: Slowdown of some kernels across GPU architectures and CUDA versions.

**Challenge 3:** *How to account for the interference between overlapping computation and communication?*

Overlapping computation and communication is widely used to improve training efficiency [1], [30], [31], yet it introduces non-negligible slowdowns to computation kernels even when they run on separate streams.[1] As shown in Table II, 23–58% of computation operations overlap with communication across frameworks, with per-step slowdown up to 1.37×; at the individual kernel level, the average slowdown is 37.73% and can reach 800% (Figure 3). Similar slowdowns have been reported in production clusters [32]. Despite this impact, most existing simulators ignore overlap-induced slowdown; Proteus [13] applies a coarse heuristic that varies only with GPU architecture and is not generalizable to unseen models.

Accurately modeling overlap slowdown requires kernel-level replay—e.g., the simulator must schedule gradient `all-reduce` at the precise point during backpropagation [31]—and the slowdown factor itself varies significantly with hardware and software setup. As Table III shows, the same kernel exhibits very different slowdowns across GPU architectures and CUDA versions, because proprietary resource scheduling determines how SMs, cache, and DRAM bandwidth are shared among concurrent kernels.

## III. Design Overview

**Key design choices.** We highlight the key choices in building Moye to address the three challenges in §II-B.

- To obtain training workloads without a full-scale deployment, Moye employs a novel ex-situ tracing approach that uses a single device to sequentially emulate each rank, faithfully tracing per-rank execution graphs and profiling operation running times(§IV-A).
- To achieve accurate and efficient communication simulation, Moye employs white-box models derived from NCCL's

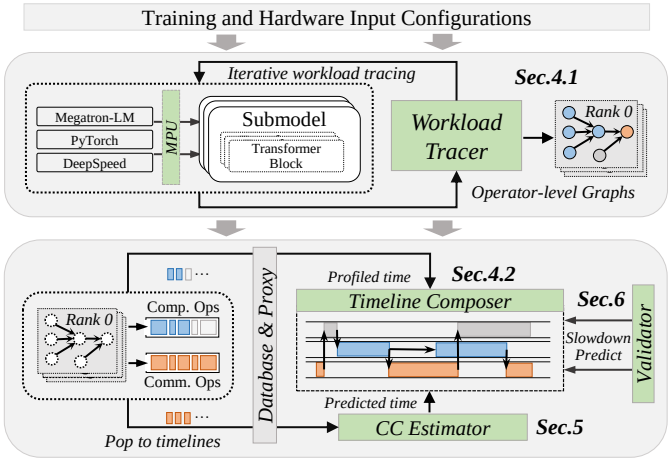



Fig. 4: Moye architecture overview. The green components represent the core modules of Moye.

chunk-based collective implementation, with key parameters profiled offline to capture hardware- and software-specific behavior (§V).

- To account for interference between overlapping computation and communication, Moye adopts a black-box model that predicts slowdown from kernel-level features such as SM occupancy and cache hit rate (§VI).

**System architecture.** Figure 4 shows Moye's architecture. Its input includes three categories: user-defined settings specifying the training framework, parallelism strategies (DP/TP/PP/EP groups); model details defining the model structure and hyperparameters; and hardware configurations detailing device specifics (GPU type), cluster size, network topology, NCCL parameters (e.g., NCCL_TOPO_FILE, NCCL_BUFFSIZE), etc. Given these inputs, the *workload tracer* extracts training workloads for each rank, including the per-rank execution graph and operation running times (§IV-A). The *CC estimator* module (§V) leverages the white-box models with parameters corresponding to the given settings to estimate each communication kernel's performance. Then the *timeline composer* module (§IV-B) re-constructs the global timelines of the end-to-end training process by assembling all per-rank execution graphs with estimated communication operation performance and inter-rank dependencies. The *validator* module (§VI) applies a machine learning model to adjust the running times of overlapping operations, accounting for the interference-induced slowdown, and outputs the final training step time result. Additionally, Moye maintains a *database* for profiling data, reducing redundant measurements across simulations.

## IV. Workload Tracing

### A. Per-Rank Workload Tracing

To launch a training job, the ML framework registers the global parallelism parameters (e.g. DP/TP/PP/EP groups) given by user into the so-called model parallelism unit (MPU), which maintains all states related to parallelisms for a given rank. Then according to the MPU, each rank concurrently initializes its own part of the model that it needs to train for

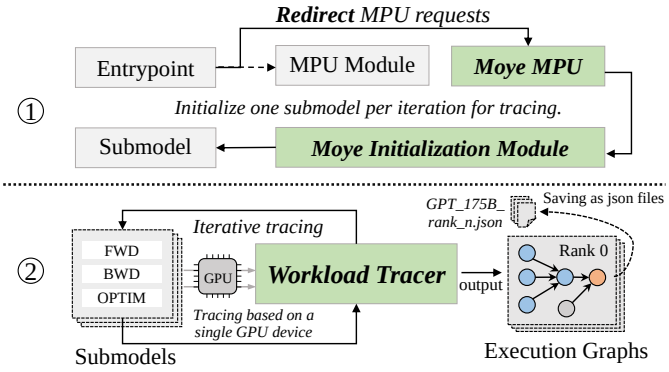



Fig. 5: Moye's MPU module and workload tracer.

(Figure 5 ① ), and starts training with an execution graph that is optimized for its submodel based on the hardware/software environment and user configurations.

Moye extracts the per-rank execution graph in an ex-situ fashion by transforming the above parallel process into a sequential one (Figure 5 ② ). Specifically, Moye hijacks all the calls to the original MPU and re-directs them to its own MPU, and registers the global parallelism parameters as usual. Moye's MPU only manipulates the input arguments to the MPU when necessary; it does not change the original implementation to ensure correctness. Then with a single device, in each iteration $i$, Moye uses this unique rank ID $i$ to initialize the corresponding submodel and execute one complete training step, so all information regarding the computation operations, including their dependencies (including those w.r.t other ranks) and running times, can be profiled. For MoE workloads, Moye exposes an interface that both replays recorded routing decisions and allows users to actively control expert-token distributions. Each simulated rank is launched as a separate single-process `torchrun` job for execution isolation; within this job, Moye samples GPU memory online, capturing allocator-level dynamics such as activation allocation, reuse, and early deallocation (see §VIII-C).

The caveat here is collective communication: one device obviously cannot execute a communication operation. To resolve this, notice that when the communication operation is launched it also needs to call into the MPU which determines the device's position in the global communication groups for proper execution. This is re-directed to Moye's MPU again allowing it to trace the communication operation's truthful information in the simulated setting (group association, message sizes, etc.); meanwhile, instead of returning results using the simulated (distributed) setting, Moye's MPU returns results corresponding to single device setting, so the communication operation can proceed and training is not blocked. This way Moye traces the actual execution graphs for each rank and running times of computation operations without requiring the full-scale cluster. Implementation details of tracing vary across frameworks which we will discuss in §VII.

Note that after per-rank workload tracing, all collective communication operations on the per-node execution graph are placeholders without any running time. The job of Moye's

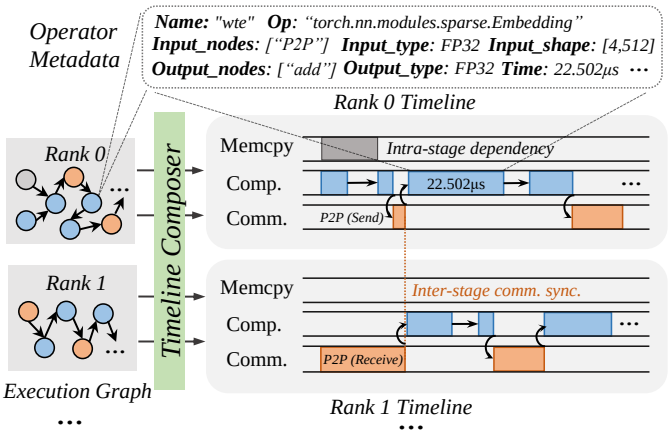



Fig. 6: Illustration of how Moye schedules operations in the execution graph to timelines across ranks.

CC estimator and validator is precisely to estimate and refine the communication times, respectively, in the next phase.

### B. Timeline Composing

Before presenting the communication estimation and validation, we discuss how Moye obtains the end-to-end training step time by composing the global timeline with the per-rank workloads obtained above.

Assuming all communication running times are known, Moye's timeline composer can recover the training process by replaying the per-rank execution graphs. timeline composer is an event-driven simulator that walks through the per-rank execution graphs and puts all operations (events) onto the global timelines accordingly, including the computation timeline, communication timeline, and memory copy (memcpy) timeline. As shown in Figure 6, a critical piece of information missing from the per-rank graphs is dependencies across ranks; for instance the downstream stages in rank $i$ requires activations from the upstream stages in rank $i-1$ before starting the forward computation in PP. We provide predefined dependencies and matching rules for both inter-stage and intra-stage events for common parallelism strategies, such as 1F1B. To enhance flexibility, this module is exposed as an API, allowing users to input new dependencies and matching rules for custom strategies. To accelerate replay, Moye relaxes synchronization for ranks whose progress is structurally balanced. This is effective for dense-model TP ranks with identical sequence lengths, while MoE workloads retain full dependencies because routing imbalance can change the critical path.

## V. Communication Modeling

NCCL provides CC functions as single CUDA kernels [35], pipelining each message into equal-sized chunks—the basis for our closed-form models below. NCCL supports ring-based [35] and tree-based [37] algorithms for `all-reduce`, and ring-based algorithms for other CC kernels. Moye models each kernel as two sequential stages: *synchronization*, where the kernel waits until all ranks in the communication group are ready (the last rank to arrive determines the start time, capturing straggler effects), and *execution*, which corresponds to the kernel's actual data-transfer and reduction time.

| #GPUs | conn_setup | data_reduction | total_trans |
|---|---|---|---|
| 32 | 7.3% | 31.7% | 61.0% |
| 64 | 6.4% | 38.3% | 55.3% |
| 128 | 5.8% | 46.2% | 48.1% |
| 256 | 5.4% | 39.3% | 55.4% |

TABLE IV: Running time breakdown of NCCL `all-reduce` kernel across GPU counts, profiled on A100 clusters. total_trans includes both intra- and inter-server transmission.

**Basic model.** We perform a running time breakdown analysis of CC kernel in Table IV. By inspecting NCCL's GPU-side implementation [38], [39] and tracing the control flow of CC kernels, we find that the execution time can be decomposed into (at most) four *non-overlapping* phases, using `all-reduce` as an example:

$$T_{comm_kernel} = T_{conn_setup} + T_{intra_trans} + T_{data_reduction} + T_{inter_trans} \quad (1)$$

Here $T_{\text{conn_setup}}$ is connection setup time; $T_{\text{intra_trans}}$, $T_{\text{inter_trans}}$ are the times to transmit tensor chunks according to the specific algorithm, and $T_{\text{data_reduction}}$ is the computation time of the reduce operation over the received tensor chunks and the local tensor. Note that not all four components are present in other CC kernels. Thus, this decomposition is implementation-faithful rather than an empirical regression artifact.

Building on Eq. (1), and following the per-round send/recv schedule encoded in NCCL's ring and tree kernels [38]–[40], we derive closed-form execution-time models for each CC kernel. The key insight is that each CC kernel's algorithmic structure—ring topology, binary tree, or concurrent P2P decomposition—determines the number of chunk-transfer rounds and the mix of intra-/inter-node links traversed, yielding the following closed-form models:

$$T^{Ring}_{all\text{-}gather} = \alpha + \gamma \cdot \eta(N-1), \quad (2)$$

$$T^{Ring}_{reduce\text{-}scatter} = \alpha + \eta\big[(N-1) + \gamma \times (N-1)\big] + \delta \times tensor_size, \quad (3)$$

$$T^{Ring}_{\texttt{all-reduce}} = \alpha + \eta\big[(N-1) + \gamma \times 2(N-1)\big] + \delta \times tensor_size, \quad (4)$$

$$T^{Tree}_{\texttt{all-reduce}} = \alpha + \gamma(\eta-1) + 2\beta(K-1) + 2\gamma \log_2(M) + \delta \times tensor_size, \quad (5)$$

$$T^{P2P}_{all\text{-}to\text{-}all} = \alpha + \max\big(\beta \cdot \eta(K-1),\ \gamma \cdot \eta(N-K)\big). \quad (6)$$

Here $N$ is the total number of devices, $M$ the number of nodes (servers), and $K = N/M$ the number of devices per node. We denote the connection setup time as $\alpha$, and the intra-server and inter-server transmission times as $\beta$ and $\gamma$. The reduce time depends on *tensor_size* and the reduce throughput $\delta$. Finally, $\eta$ is the number of rounds (or chunks) for this tensor, which corresponds to NCCL's chunk-based implementation. For Eqs. (2)–(4), the $(N-1)$ terms arise from the ring topology where each rank forwards a chunk to the next in sequence. Eq. (5) replaces this linear dependence with $\log_2(M)$ inter-node hops and $K-1$ intra-node hops reflecting the tree structure. In Eq. (6), the max operator

captures that intra-node transfers (to $K-1$ GPUs via NVLink) and inter-node transfers (to $N-K$ GPUs via NIC) proceed concurrently over independent physical channels, so the slower path dominates.

This modeling decouples scale-dependent terms from fabric-dependent parameters: job scale and parallelism enter only through $N, M, K,$ *tensor_size*, and $\eta$, while $(\alpha, \beta, \gamma, \delta)$ summarize the steady-state behavior of intra- and inter-node communication for a fixed hardware/software stack. Once these parameters are calibrated on a moderate-size cluster, the same closed-form expressions can be evaluated for much larger $N$ and $M$ without additional network-level simulation.

In practice, large jobs incur additional latency from multi-tier topologies and aggregate contention. To capture these second-order effects without packet-level simulation, Moye applies a scale correction factor $c_{\text{scale}}(N) \geq 1$ to the inter-server transmission time:

$$\gamma_{\text{eff}}(N) = \gamma \cdot c_{\text{scale}}(N), \tag{7}$$

so the closed-form models retain their algorithmic scaling structure while $c_{\text{scale}}(N)$ adjusts for residual network effects at larger scales.

**Offline exhaustive profiling.** The fabric-dependent parameters $(\alpha, \beta, \gamma, \delta, \eta)$ cannot be modeled analytically because they depend on *chunk_size* (dynamically tuned by NCCL [35]), hardware type, and NCCL version. Moye obtains them via offline profiling: we instrument NCCL to extract the chunk size, enumerate representative combinations of $N$, $M$, *tensor_size*, and link type, and profile $(\alpha, \beta, \gamma, \delta)$ for each resulting *chunk_size*. In common Clos-based topologies, $(\alpha, \beta, \gamma, \delta)$ converge once the job spans a small number of nodes; residual scale-dependent effects are captured by $c_{\text{scale}}(N)$ (see §VII). Taken together, combining white-box analytical models with offline-profiled parameters and a lightweight scale correction factor provides a good first-order estimation of CC kernel performance with very low overheads.

## VI. Overlapping Slowdown Prediction

Accurately capturing and modeling the resource contention between overlapping kernels presents substantial challenges in predicting the slowdown factor as discussed in §II-B. To address this, Moye introduces an ML-based slowdown prediction model with a number of hardware- and software-specific features, which we present here.

GPU devices execute concurrent computation and communication kernels with spatial multitasking, where each kernel is assigned to separate streams with a subset of the SMs. At the same time, they also contend for shared resources such as cache and DRAM bandwidth. These factors result in performance interference, and Moye needs to capture them when predicting the slowdown.

**Feature extraction.** Table V summarizes the key features used by our model. We categorize features into two groups: communication-related details and computation kernel metrics. For communication, we consider features such as protocol and algorithm employed by NCCL, CC kernel type, bucket size, and channel configuration. These parameters capture the complexity and overheads of communication: bucket size, for instance, determines when a communication kernel will be launched after accumulating one bucket of data. On the computation side, we profile kernels at a fine-grained level on a single device to measure their baseline performance without overlapping. We collect running times, compute throughput, memory utilization, and cache statistics using NVIDIA Nsight Systems and Nsight Compute. For instance, achieved occupancy and SM activity levels indicate compute intensity, while DRAM utilization and memory throughput—along with L1/L2 hit rates—shed light on memory access latency and potential bottlenecks. Additionally, we classify kernels by their functional types (e.g., matrix multiplication, layer normalization) to further refine our predictions.

| Type | Feature | Specification |
|---|---|---|
| Comm. details | Protocol | Communication protocol used within NCCL |
| | Algorithm | Communication topology algorithm applied in NCCL |
| | Collective | Name of the collective function in the kernel |
| | Bucket size | Size of the tensor bucket used in collective communication |
| | Channel number | Number of communication channels in NCCL |
| Comp. Kernel Metric | Running time | Running time profiled on single-device training |
| | Compute throughput | Average SM throughput |
| | Memory throughput | Percentage of cycles where DRAM was active |
| | DRAM throughput | Peak DRAM throughput |
| | Achieved occupancy | Average percentage of active warps on each SM |
| | L1 hit rate | Hit rate for L1 cache |
| | L2 hit rate | Hit rate for L2 cache |

TABLE V: Features in the slowdown prediction model.

**XGBoost-based model.** We leverage XGBoost [41], a tree-based gradient boosting framework, to predict the slowdown factors caused by overlapping kernels with the above features. XGBoost is well-suited to this task for two key reasons. First, our dataset contains heterogeneous features that are both numerical (e.g., SM throughput, DRAM utilization) and categorical (e.g., GPU type, kernel type), a scenario where tree-based models excel. Second, our target slowdown factors, influenced by the interplay of diverse system conditions, remain bounded within a range well-captured by the training set. As we expand our dataset—incorporating additional profiles from various models, GPU architectures, and cluster scales—the model's generalization ability improves, enabling accurate slowdown predictions across increasingly complex and heterogeneous environments.

## VII. Implementation

We implement Moye as illustrated in Figure 4. Moye is developed based on PyTorch 2.1, DeepSpeed 0.13.1, and Megatron-LM, and NCCL 2.18.6, all running on CUDA 12.1, comprising ∼10K lines of code (LoC). ∼2K LoC are dedicated to Megatron-LM, 3K LoC to DeepSpeed and PyTorch, and the remaining 5K LoC for common core modules that underpin the simulation engine, training workload tracing, overlapping induced slowdown dataset collection, and CC parameter exhaustive profiling.

## VIII. Evaluation

In this section, we first evaluate Moye's end-to-end accuracy and runtime cost (§VIII-B), then break down the accuracy and

| Model | Layers | Hidden | Heads | FFN | Experts | Expert FFN | Top-$k$ | Attn. |
|---|---|---|---|---|---|---|---|---|
| Mixtral-8×1.75B | 8 | 4096 | 32 | – | 8 | 14336 | 2 | GQA |
| Qwen3-A3B | 48 | 2048 | 32 | 6144 | 128 | 768 | 8 | GQA |
| DeepSeek-V3 variant | 32 | 2048 | 64 | 4096 | 32 | 512 | 2 | MLA |

TABLE VI: Key configuration parameters for the three evaluated MoE workloads. FFN denotes the dense (non-expert) FFN hidden size; Mixtral routes all FFN computation through experts.

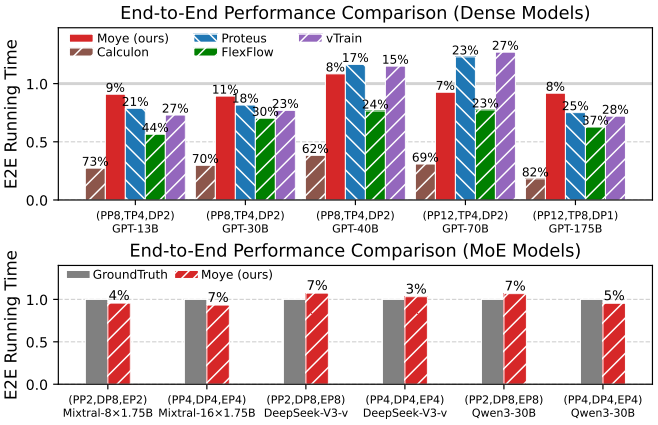



Fig. 7: End-to-end training step time comparison on dense and MoE models (including Mixtral, DeepSeek-V3 variant, and Qwen3-A3B-30B). All results are normalized to ground truth ($y$=1.0/gray bars). Numbers above bars denote prediction error. Baseline systems do not support MoE simulation.

effectiveness of its key modules (§VIII-C, §VIII-D, §VIII-E). We provide the experimental setup in §VIII-A.

### A. Experiment Setup

**Models.** We evaluate Moye on both dense and MoE models. For dense workloads, we use VGG19 [42] and GPT models [17] ranging from 13B to 485B parameters. For MoE workloads, we use the Mixtral series [43], Qwen3-A3B [44], and a DeepSeek-V3 variant derived from DeepSeek-V3 [45]; their key configurations are summarized in Table VI.

**Clusters.** Our evaluation utilizes a 256-GPU H800 cluster (32 servers with 8x 80GB GPUs each) and an 8-GPU A800 cluster. Both use 400 GB/s NVLink for intra-server communication, while the H800 cluster is equipped with 400 Gb/s NDR InfiniBand for inter-server links. Additionally, an RTX 3090 cluster is used for kernel slowdown analysis (§VIII-E).

**Baselines.** We compare Moye to five open-source simulators:

- **Proteus** [13]: A recent simulator for integrated simulation of parallel strategies and runtime behaviors.
- **FlexFlow** [18]: An internal simulator for throughput estimation across different parallelization strategies.
- **Calculon** [20]: NVIDIA's analytical performance simulator for hardware-software codesign exploration.
- **vTrain** [23]: A GPT-specific simulation framework for 3D parallel LLM training.
- **SimAI** [12]: It uses packet-level network simulation for high-fidelity communication modeling.

### B. End-to-End Results

**Accuracy.** We evaluate the end-to-end accuracy of Moye on H800 clusters of varying scales using Megatron-LM. As shown in Figure 7, Moye consistently achieves high fidelity across

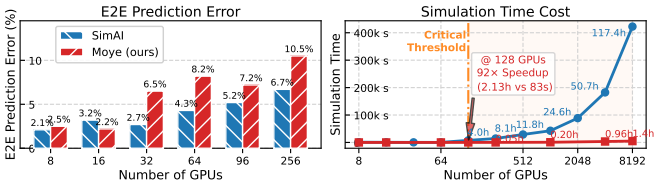



Fig. 8: Comparison of Moye and SimAI on GPT series (H800 cluster). Left: end-to-end prediction error across cluster scales. Right: simulation time cost (log-log scale). Beyond 128 GPUs, Moye achieves a 91.8× speedup over SimAI while maintaining comparable accuracy.

diverse parallelism configurations. For 96-GPU scenarios, the prediction error is 7% for GPT-70B ($12 \times 4 \times 2$) and 8% for GPT-175B ($12 \times 8 \times 1$). In contrast, baseline simulators deviate significantly: Proteus and FlexFlow show errors exceeding 25%, while Calculon is overly optimistic (error >69%) due to its reliance on ideal analytical models. Even as a GPT-specific baseline, vTrain still shows 15%–28% end-to-end error on the dense cases in Figure 7. Moye also attains high accuracy on MoE models. We evaluate three MoE families that stress distinct execution paths: Mixtral (standard top-2 gating), the DeepSeek-V3 variant (MLA, group-limited routing, shared experts, and all-to-all communication), and Qwen3-A3B (128 experts with top-8 selection). As shown in the lower subplot of Figure 7, no existing baseline supports MoE simulation, so we compare Moye against ground truth only. Across all six MoE configurations, Moye achieves 3%–7% end-to-end prediction error. In addition to step time, Moye accurately predicts peak GPU memory usage, with average errors of 1.5% on dense models and 3.2% on MoE models, enabling reliable OOM feasibility screening for ex-situ exploration. Across models, Moye achieves average errors of 4.5% and 17.8% for computation and communication operations.

**Comparison with packet-level simulator.** We compare Moye against SimAI [12], a packet-level network simulator built on ns-3, on GPT-175B training across 8–8,192 H800 GPUs (Figure 8). In terms of accuracy, both simulators achieve comparable end-to-end prediction errors at small scales (2–3% at 8–32 GPUs). As the cluster grows, Moye's error increases moderately to 10.5% at 256 GPUs, while SimAI remains at 6.7%—a gap attributable to SimAI's finer-grained packet-level communication modeling. However, the simulation time gap is dramatic. At 128 GPUs, Moye completes in 83 seconds—a 91.8× speedup over SimAI's 7,655 seconds. This advantage widens further at scale: at 8,192 GPUs, Moye finishes in 1.4 hours while SimAI requires over 117 hours. For a design-space exploration sweeping 100 configurations on an 8,192-GPU cluster, Moye would complete in under 6 days, whereas SimAI would require over 1.5 years. These results demonstrate that Moye offers a compelling accuracy–efficiency trade-off, making it practical for iterative what-if analysis at datacenter scale where packet-level simulation is prohibitively expensive.

**Ablation study.** To isolate the contribution of each key module, we perform an ablation study on GPT-175B (FP16) trained on a 256-GPU H800 cluster with 3D parallelism

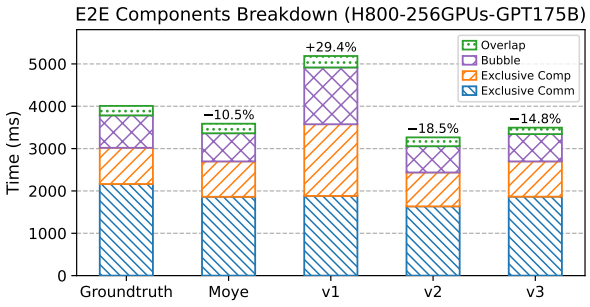



Fig. 9: Component breakdown of GPT-175B training step time on 256 H800 GPUs with 3D parallelism. Each bar decomposes the step time into exclusive computation, exclusive communication, bubble, and overlap. Numbers above bars denote end-to-end prediction error.

| Variant | Excl. Comp. | Excl. Comm. | Bubble | Overlap | End-to-End |
|---|---|---|---|---|---|
| Moye (Full) | −2.2% | −14.0% | −13.2% | +1.1% | −10.5% |
| w/o Ex-situ Tracer (v1) | +98.2% | −13.0% | +75.0% | +21.1% | +29.4% |
| w/o White-box Comm. (v2) | −6.3% | −24.4% | −19.0% | −6.8% | −18.5% |
| w/o Slowdown Pred. (v3) | −2.5% | −13.9% | −15.7% | −31.9% | −14.8% |

TABLE VII: Ablation study on GPT-175B (256 H800 GPUs, 3D parallelism). Each column shows the signed prediction error of the corresponding time component relative to ground truth.

$(\text{PP} = 16, \text{TP} = 8, \text{DP} = 2)$. We disable one module at a time and report the signed prediction error for each time component against ground truth (Figure 9 and Table VII).

Removing the ex-situ tracer inflates computation error to $+98.2\%$, which propagates to $+29.4\%$ end-to-end deviation—the largest among all variants. Replacing the white-box communication model with $\alpha$-$\beta$ worsens end-to-end accuracy from 10.5% to 18.5%, while disabling slowdown prediction degrades overlap estimation from $+1.1\%$ to $-31.9\%$. The three modules are complementary: the ex-situ tracer establishes a faithful computation substrate, the white-box model determines accurate communication intervals, and the slowdown predictor corrects their nominal overlap into realistic performance.

**Simulation cost.** We report two runtime costs in Table VIII: (1) the per-configuration simulation cost, which includes initialization and execution, and (2) the one-time, reusable workload tracing cost. Across DDP settings, the per-configuration simulation cost remains around 10 seconds, while tracing takes about 18 seconds. For GPT workloads with 3D parallelism, the per-configuration simulation cost grows from 167.6 seconds at 256 GPUs to 4976.9 seconds at 8,192 GPUs, while the one-time trace cost ranges from 1350.2 to 3498.2 seconds. These results show that Moye keeps each simulation run practical even at large scale, while amortizing the tracing cost across repeated design-space exploration runs.

### C. Computation and Memory Usage Simulation

We evaluate Moye's accuracy in simulating computation workloads on both H800 and A800 clusters, using VGG19 (with DeepSpeed), GPT-13B and Mixtral-8×1.75B (both with Megatron-LM) under various PP/EP configurations.

**Accuracy.** Moye reconstructs the computation workload by tracing all computational operations and reproducing their execution sequence based on the execution graph. Per-GPU operation running times are derived through Moye's workload tracer without requiring a full-scale deployment. As shown in Figure 10, Moye achieves a maximum overall error of 8.31%. In contrast, while Proteus and FlexFlow maintain relatively low error rates for VGG19 under DeepSpeed (8.6% and 17.81%, respectively), their errors escalate dramatically for GPT-13B under Megatron-LM, reaching 91.53% and 109.14%, respectively. The root cause of this discrepancy is that Proteus and FlexFlow rely heavily on standard PyTorch operations, failing to incorporate the custom fused operations that frameworks like Megatron-LM employ for performance optimization. Such specialized operations cannot be accurately modeled without extracting their runtime characteristics directly from the target training framework (as we disccused in §II). Calculon overestimates computation time in an overly idealized manner (error rate $> 400\%$), because its modeling relies on hardware's analytical FLOPS values, which severely deviate from real hardware execution. vTrain only supports GPT-13B here, and it still incurs 63.7%–204.2% error because its profiled kernel stack (sampled from a custom `ShardedGptModel` with a plain `F.linear` path) is not aligned with the real Megatron-Core/Transformer-Engine execution stack.

| #GPUs | VGG19 | | | | GPT 13B∼175B | | | |
|---|---|---|---|---|---|---|---|---|
| | Init. | Execution | Sim. Total | Trace | Init. | Execution | Sim. Total | Trace |
| 256 | 0.4 | 10.3 | 10.7 | 18.2 | 8.4 | 159.2 | 167.6 | 1350.2 |
| 1024 | 0.3 | 9.7 | 10.0 | 18.5 | 35.9 | 682.8 | 718.7 | 1926.7 |
| 4096 | 0.4 | 10.2 | 10.6 | 17.8 | 149.9 | 3307.8 | 3457.7 | 2688.3 |
| 8192 | 0.3 | 10.3 | 10.5 | 18.3 | 374.7 | 4602.2 | 4976.9 | 3498.2 |

TABLE VIII: Runtime costs of Moye in seconds. Sim. Total reports the per-configuration simulation cost, while Trace reports the one-time, reusable workload tracing cost. VGG19 is trained using the DDP mode, while GPT models (ranging from 13B to 175B) are trained using 3D parallelism.

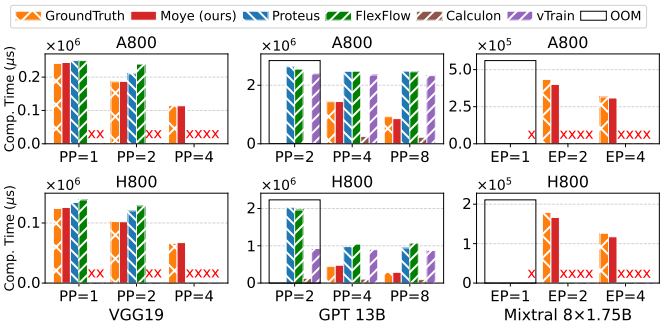



Fig. 10: Computation times of various workloads with various parallelization settings. Crosses denote cases not supported by the baselines, while squares mark configurations that encountered OOM errors during real execution.

**Generality.** Because Moye derives workloads through framework-level ex-situ tracing rather than pre-hardcoded construction, it generalizes naturally across CNN, dense Transformer, and MoE architectures (Figure 7)—including Qwen3-A3B and the DeepSeek-V3 variant, which none of the baselines can simulate. Coupled with high-fidelity GPU memory simulation (Figure 11), this enables reliable large-scale design-space exploration, as demonstrated by the 8,192-GPU ex-situ study in §IX.

**MoE routing-skew simulation.** We inject three deterministic routing profiles—balanced, moderate, and strong skew—into

| Routing | Load imbalance | Moye straggler | Comp. error |
|---|---|---|---|
| Balanced | 1.00× | 1.24× | 7.88% |
| Moderate | 1.54× | 1.08× | 4.40% |
| Strong | 3.00× | 1.22× | 6.64% |

TABLE IX: Routing-skew stress test on the DeepSeek-V3 variant (PP=2, TP=1, EP=4, 8 A800 GPUs). *Load imbalance*: ratio of token count on the hottest expert to the median. *Moye straggler*: ratio of the slowest EP rank's computation time to the fastest in Moye's ex-situ traces. *Comp. error*: relative error of Moye's critical-path computation time vs. ground truth.

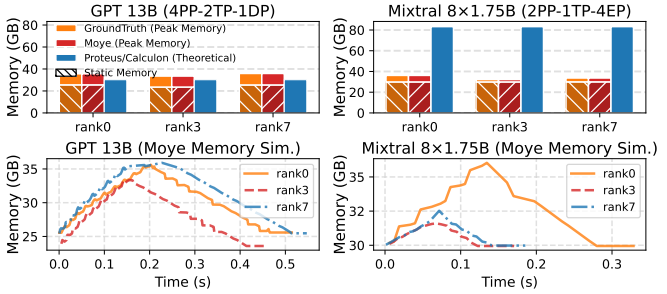



Fig. 11: Validation of Moye's GPU memory simulation for GPT-13B and Mixtral-8×1.75B on our 8-GPU A800 cluster (8 ranks), showing three representative ranks. It demonstrates Moye's ability to capture dynamic per-rank memory usage within a training step (excluding communication time). We set num_micro_batch=micro_batch_size=1 here.

the DeepSeek-V3 variant with fixed configuration (Table IX). Under strong skew, the hottest expert receives 3× as many tokens as the median. Moye predicts this stressed regime with 6.64% critical-path computation error, confirming its MoE fidelity extends to scenarios with significant expert-load imbalance.

**GPU Memory usage.** Moye supports high-fidelity GPU memory usage simulation through ex-situ profiling. Figure 11 (upper) shows the peak memory usage (including both static and dynamic components), where Moye achieves nearly 100% accuracy on GPT-13B and Mixtral-8×1.75B for different ranks. In contrast, analytical methods can result in 18.0% error for GPT-13B, and for MoE models (Mixtral-8×1.75B), the error even increases to 2.58× (a gap of 51 GB). Moye can also reflect dynamic GPU allocation behavior during training, as shown in Figure 11 (lower)—a capability absent in existing simulators.

### D. Communication Simulation

We now evaluate the effectiveness of Moye's communication simulation, comparing it against NCCL-Predictor [35], the underlying communication model used by both Proteus and FlexFlow. NCCL-Predictor leverages parameters tuned from production experience and employs an $\alpha - \beta$ model (§II-B).

**Accuracy.** Figure 12 shows the prediction accuracy of CC kernels across message sizes (4MB–256MB) and cluster scales. Moye achieves average errors of 12.6% and 13.8% on 8- and 12-server H800 clusters, improving upon NCCL-Predictor by 29.6% and 19.4%, respectively. Here, for 8- and 12-server clusters we set the scaling factor $c_{\text{scale}}(N)$ to 1.023 and 1.063, respectively. The gap is smallest on ring-based `all-reduce` (4.2%), where NCCL's optimizations align well with the $\alpha$-$\beta$ model, and largest on tree-based `all-reduce` (up to 33.5%). These improvements stem from Moye's profiling-based white-box model, which captures bandwidth utilization and synchronization overheads that the $\alpha$-$\beta$ abstraction overlooks. As a result, Moye maintains consistent accuracy across message sizes, whereas NCCL-Predictor averages 27.6% error on small messages (4MB and 16MB). vTrain, whose communication model also relies on analytical NCCL-style formulas, averages 41.5% and 33.9% error on the 8- and 12-server cases, respectively. For the all-to-all collective used by MoE expert dispatch/combine, Moye achieves 13.6% average error across 1- and 2-server H800 configurations, compared to 25.8% for the $\alpha$-$\beta$ model.

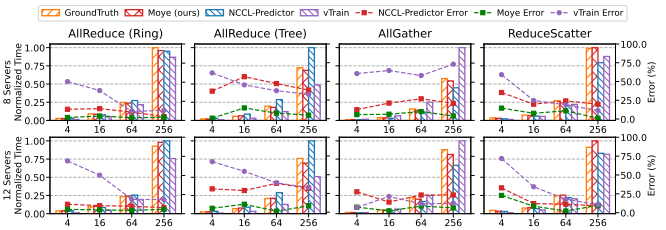



Fig. 12: Running time and prediction error for inter-server CC kernels on H800 GPUs across varying tensor sizes and cluster scales. Each subplot is normalized relative to the operation with the longest running time.

| Tensor size (MB) | 4 | 8 | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|---|
| Chunk size ($10^3$ bytes) | 18 | 36 | 72 | 72 | 72 | 144 | 288 |
| Intra-server Round trip ($\mu$s) | 57.43 | 64.36 | 73.62 | 74.61 | 77.46 | 75.55 | 99.23 |
| Inter-server Round trip ($\mu$s) | 54.3 | 72.32 | 106.73 | 112.9 | 111.25 | 174.2 | 278.98 |
| Reduce operation ($\mu$s) | 50.14 | 65.86 | 102.46 | 219.89 | 463.43 | 841.71 | 1657.72 |

TABLE X: The breakdown of Moye's `all-reduce` running time, profiled in the H800 cluster using 32 GPUs.

**Breakdown analysis.** Table X breaks down `all-reduce` runtime on a 32-GPU H800 cluster. Intra- and inter-server round-trip times grow moderately with tensor size, while reduction computation dominates at large tensors, scaling nearly linearly from 50.1 µs (4MB) to 1658 µs (256MB). These profiled components directly feed into Moye's closed-form models (§V).

### E. Slowdown Prediction

Most simulators overlook the performance slowdown caused by kernel overlap. We compare Moye with Proteus, the only baseline known to model this effect, which does so using a simple heuristic based on GPU architecture and model type.

**Prediction accuracy.** We evaluate slowdown prediction on ∼10,000 kernels (Table XI). Moye consistently outperforms the baseline heuristic, especially on data-center GPUs. On the H800, 57.7% of predictions fall within 15% error, compared to 40.2% for the baseline, and the RMSE is nearly 3× lower (0.56 vs. 1.55). Similar gains hold on the A800, while on consumer-grade RTX 3090 GPUs, Moye again shows substantially lower error across all metrics.

**Model-wise evaluation.** We evaluate slowdown-aware modeling on VGG19, GPT2, GPT-6.7B, and Mixtral-8×1.75B

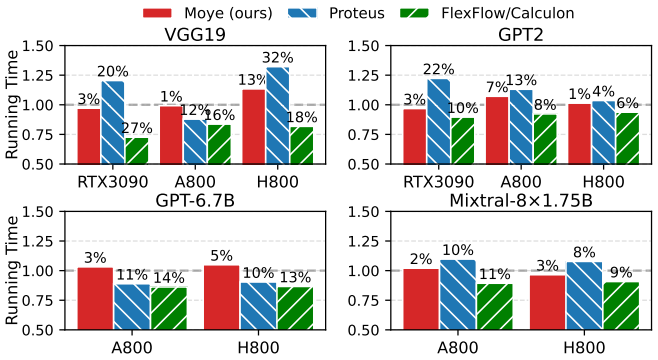



Fig. 13: Normalized model-wise computation time on H800, A800, and RTX3090 GPUs. Moye (prediction-based) is compared against Proteus (fixed-parameter slowdown modeling) and FlexFlow/Calculon (no slowdown modeling). Numbers above bars denote prediction error relative to ground truth.

| Solution | GPU | 5% Error | 10% Error | 15% Error | RMSE |
|---|---|---|---|---|---|
| Moye | 3090 | 61.48% | 73.62% | 76.52% | 1.0935 |
| Proteus | | 8.52% | 13.62% | 19.28% | 1.4953 |
| Moye | A800 | 61.03% | 69.82% | 75.01% | 0.7147 |
| Proteus | | 59.04% | 65.45% | 70.51% | 1.5170 |
| Moye | H800 | 43.18% | 51.59% | 57.73% | 0.5626 |
| Proteus | | 34.78% | 37.36% | 40.18% | 1.5493 |

TABLE XI: Fraction of cases where the slowdown prediction error is within the given error range.

under overlap-heavy settings (Figure 13). Moye achieves 2–5% model-level error, compared to 8–23% for Proteus and 10–37% for FlexFlow/Calculon, confirming that accurate slowdown prediction is essential for faithfully modeling modern training workloads with heavy overlap.

## IX. Use Case: Training Strategy Search

A practical question in large-scale training is: given a GPU cluster and a target LLM, how should the job and its parallelism strategy be configured to achieve high training efficiency? Answering this question by trial-and-error on the real cluster is prohibitively expensive, as each trial may waste thousands of GPU-hours. Moye provides a low-cost alternative by efficiently searching across a wide range of parallelism and system configurations.

The workflow proceeds in two stages. First, Moye conducts a GPU memory feasibility scan via grid search over parallelism choices (Figure 14). This step leverages its high-fidelity memory simulation to filter out infeasible configurations and avoid false OOM predictions that plague existing analytical models (e.g., Proteus and Calculon). Second, for all surviving candidates, Moye performs end-to-end simulation to estimate throughput and cost. By faithfully reconstructing the interaction among computation, communication, and memory behaviors, Moye enables a more reliable comparison of training efficiency across configurations. From these results, Moye recommends the configuration that minimizes time-to-train and monetary cost (Table XII).

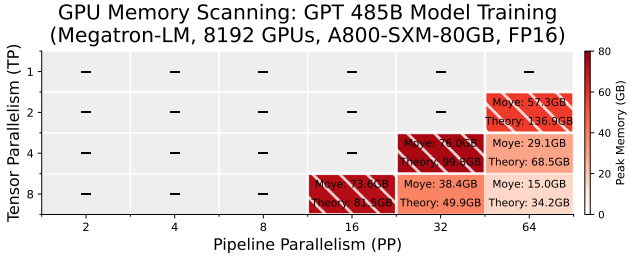



Fig. 14: GPU memory feasibility scan over PP/TP strategies for GPT-485B on 8,192 A800 GPUs. Gray cells are OOM, while red hatched cells are empirically feasible strategies misclassified as OOM by analytical baselines.

| Setting | Peak memory | Throughput | Time (days) | Cost |
|---|---|---|---|---|
| (PP16, TP8, DP64) | 73.6 GB | 455.8k tokens/s | 12.7 | $2.45M |
| (PP32, TP8, DP32) | 38.4 GB | 361.1k tokens/s | 16.0 | $3.09M |
| (PP64, TP8, DP16) | 15.0 GB | 331.9k tokens/s | 17.4 | $3.36M |
| (PP32, TP4, DP64) | 76.0 GB | 350.0k tokens/s | 16.5 | $3.19M |
| (PP64, TP4, DP32) | 29.1 GB | 222.1k tokens/s | 26.1 | $5.02M |
| (PP64, TP2, DP64) | 57.3 GB | 308.3k tokens/s | 18.8 | $3.62M |

TABLE XII: Top strategy-search results on 8,192 A800 GPUs for GPT-485B and 500B training tokens. Time and cost are derived from predicted throughput; red and gray rows denote Moye's and Calculon's choices.

## X. Limitations and Discussions

We plan to improve Moye in the following aspects.

**Communication prediction.** Moye positions itself between hand-tuned $\alpha$–$\beta$ models and packet-level network simulators. We plan to investigate learning based solutions like m3 [46] that efficiently predicts flow-level performance while capturing network-centric factors such as transport protocols.

**Parallelism.** Moye does not support sequence/context parallelism; we plan to integrate them in subsequent releases.

## XI. Related Work

We discuss related work other than those covered in §II.

**GPU simulation.** GPU computation simulators like SCALE-sim [47] and Accel-Sim [48] focus on instruction-level modeling, while other approaches [15], [16], [19], [49] rely on profiling or tracing to capture accurate operation execution times. Moye similarly adopts a trace-based methodology, but crucially does so without requiring a full-scale deployment.

**Network simulation.** Discrete event simulators like ns-3 [28] and OMNeT++ [50] are well-known to have high overheads due to their packet-level whole-stack simulation. Other than parallelization optimizations [51], [52], recently learning based approach to predict packet-level or flow-level performance without simulating the whole stack [46], [53], [54] has shown promising results. As mentioned before, integrating them with Moye for training simulation is a promising future direction.

## XII. Conclusion

We presented Moye, a high-fidelity simulator for large-scale distributed ML training that bridges critical gaps in existing approaches. Moye simulates without full-scale deployment using ex-situ workload tracing, estimates CC performance via white-box modeling, and accounts for the slowdown caused by kernel overlapping. Our evaluation shows that Moye not only achieves up to 3x lower simulation error than state-of-the-art baselines but also is highly efficient and practical. We plan to open source Moye to the community.

## References


[1] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "{MegaScale}: Scaling large language model training to more than 10,000 {GPUs}," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 745–760. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng

[2] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.02677

[3] https://ai.meta.com/blog/meta-llama-3-1, 2024.

[4] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*. PMLR, 2020, pp. 5156–5165. [Online]. Available: https://proceedings.mlr.press/v119/katharopoulos20a.html

[5] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *ICLR*, 2017. [Online]. Available: https://openreview.net/forum?id=B1ckMDqlg

[6] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer *et al.*, "Pytorch fsdp: experiences on scaling fully sharded data parallel," *arXiv preprint arXiv:2304.11277*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.11277

[7] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "{ZeRO-Offload}: Democratizing {Billion-Scale} model training," in *USENIX ATC*, 2021. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/ren-jie

[8] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.19437

[9] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, "FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision," https://arxiv.org/abs/2407.08608, 2024.

[10] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022. [Online]. Available: https://doi.org/10.52202/068431-1189

[11] "MemryX MX3 Chip," https://memryx.com/products/, 2024.

[12] R. Hao, S. Chen, C. Li, H. Wu, G. Sreekumar, Y. Liu, X. Zhou, J. Yang, Y. Sun, W. Chen, R. Zhao *et al.*, "Simai: Unifying architecture design and performance tuning for large-scale large language model training with scalability and precision," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, Apr. 2025, pp. 541–558. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/hao

[13] J. Duan, X. Li, P. Xu, X. Zhang, S. Yan, Y. Liang, and D. Lin, "Proteus: Simulating the performance of distributed dnn training," *IEEE Transactions on Parallel and Distributed Systems*, 2024. [Online]. Available: https://doi.org/10.1109/TPDS.2024.3443255

[14] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. 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)*. IEEE, 2023, pp. 283–294. [Online]. Available: https://doi.org/10.1109/ISPASS57527.2023.00035

[15] G. Lu, R. Chen, Y. Wang, Y. Zhou, R. Zhang, Z. Hu, Y. Miao, Z. Cai, L. Li, J. Leng *et al.*, "Distsim: A performance model of large-scale hybrid distributed dnn training," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, 2023, pp. 112–122. [Online]. Available: https://doi.org/10.1145/3587135.3592200

[16] H. Hu, C. Jiang, Y. Zhong, Y. Peng, C. Wu, Y. Zhu, H. Lin, and C. Guo, "dpro: A generic performance diagnosis and optimization toolkit for expediting distributed dnn training," *Proc. MLSys*, 2022. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2022/hash/b422680f3db0986ddd7f8f126baaf0fa-Abstract.html

[17] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1909.08053

[18] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks." *Proceedings of Machine Learning and Systems*, vol. 1, pp. 1–13, 2019. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2019/hash/b422680f3db0986ddd7f8f126baaf0fa-Abstract.html

[19] H. Zhu, A. Phanishayee, and G. Pekhimenko, "Daydream: Accurately estimating the efficacy of optimizations for {DNN} training," in *USENIX ATC*, 2020. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/zhu-hongyu

[20] M. Isaev, N. Mcdonald, L. Dennison, and R. Vuduc, "Calculon: a methodology and tool for high-level co-design of systems and large language models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023, pp. 1–14. [Online]. Available: https://doi.org/10.1145/3581784.3607102

[21] M. Liang, H. T. Kassa, W. Fu, B. Coutinho, L. Feng, and C. Delimitrou, "Lumos: Efficient performance modeling and estimation for large-scale llm training," *arXiv preprint arXiv:2504.09307*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2504.09307

[22] Y. Li, Y. Bao, G. Wang, X. Mei, P. Vaid, A. Ghosh, A. Jog, D. Bunandar, A. Joshi, and Y. Sun, "Triosim: A lightweight simulator for large-scale dnn workloads on multi-gpu systems," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1524–1538. [Online]. Available: https://doi.org/10.1145/3695053.3731082

[23] J. Bang, Y. Choi, M. Kim, Y. Kim, and M. Rhu, "vtrain: A simulation framework for evaluating cost-effective and compute-optimal large language model training," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 153–167. [Online]. Available: https://doi.org/10.1109/MICRO61859.2024.00021

[24] J. Duan, S. Zhang, Z. Wang, L. Jiang, W. Qu, Q. Hu, G. Wang, Q. Weng, H. Yan, X. Zhang *et al.*, "Efficient training of large language models on distributed infrastructures: A survey," *arXiv preprint arXiv:2407.20018*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.20018

[25] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble pipeline parallelism," *arXiv preprint arXiv:2401.10241*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2401.10241

[26] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Proc. NeurIPS*, 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/093f65e080a295f8076b1c5722a46aa2-Abstract.html

[27] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990. [Online]. Available: https://doi.org/10.1145/79173.79181

[28] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_2

[29] L.-W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, C. Yao, Z. Jiang *et al.*, "Flux: Fast software-based communication overlap on gpus through kernel fusion," *arXiv preprint arXiv:2406.06858*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2406.06858

[30] C. Chen, X. Li, Q. Zhu, J. Duan, P. Sun, X. Zhang, and C. Yang, "Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 178–191. [Online]. Available: https://doi.org/10.1145/3620666.3651379

[31] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on {GPU} clusters," in *USENIX ATC*, 2017. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/zhang

[32] C. Hwang, K. Park, R. Shu, X. Qu, P. Cheng, and Y. Xiong, "{ARK}:{GPU-driven} code execution for distributed deep learning," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 87–101. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/hwang

[33] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. 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*, 2020, pp. 3505–3506. [Online]. Available: https://doi.org/10.1145/3394486.3406703

[34] A. Weingram, Y. Li, H. Qi, D. Ng, L. Dai, and X. Lu, "xccl: A survey of industry-led collective communication libraries for deep learning," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 166–195, 2023. [Online]. Available: https://doi.org/10.1007/s11390-023-2894-6

[35] "Nvidia Collective Communications Library (NCCL)," https://developer.nvidia.com/nccl, 2024.

[36] "A fast communication-overlapping library for tensor/expert parallelism on GPUs." https://github.com/bytedance/flux, 2025.

[37] "Massively Scale Your Deep Learning Training with NCCL 2.4," https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/, 2019.

[38] "NCCL GPU-side implementation," https://github.com/NVIDIA/nccl/tree/master/src/device, 2025.

[39] Z. Hu, S. Shen, T. Bonato, S. Jeaugey, C. Alexander, E. Spada, J. Dinan, J. Hammond, and T. Hoefler, "Demystifying nccl: An in-depth analysis of gpu communication protocols and algorithms," *arXiv preprint arXiv:2507.04786*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2507.04786

[40] "NCCL ring and tree algo src codes," https://github.com/NVIDIA/nccl/tree/master/src/graph, 2025.

[41] "XGBoost," https://xgboost.readthedocs.io/en/stable/, 2014.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Online]. Available: https://doi.org/10.48550/arXiv.1409.1556

[43] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.04088

[44] "qwen3-a3b," https://huggingface.co/Qwen/Qwen3-30B-A3B.

[45] "deepseek-v3," https://huggingface.co/deepseek-ai/DeepSeek-V3.

[46] C. Li, A. Nasr-Esfahany, K. Zhao, K. Noorbakhsh, P. Goyal, M. Alizadeh, and T. E. Anderson, "m3: Accurate flow-level performance estimation using machine learning," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 813–827. [Online]. Available: https://doi.org/10.1145/3651890.3672243

[47] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1811.02883

[48] M. Khairy, Z. Shen, T. M. Aamodt, and T. G. Rogers, "Accel-sim: An extensible simulation framework for validated gpu modeling," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 473–486. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00047

[49] L. Luo, P. West, P. Patel, A. Krishnamurthy, and L. Ceze, "Srifty: Swift and thrifty distributed neural network training on the cloud," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 833–847, 2022. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2022/hash/0cafb7890f6a7d4de65507d5bb7e0187-Abstract.html

[50] A. Varga, "A practical introduction to the omnet++ simulation framework," in *Recent advances in network simulation: the OMNeT++ environment and its ecosystem*. Springer, 2019, pp. 3–51. [Online]. Available: https://doi.org/10.1007/978-3-030-12842-5_1

[51] K. Gao, L. Chen, D. Li, V. Liu, X. Wang, R. Zhang, and L. Lu, "DONS: Fast and Affordable Discrete Event Network Simulation with Automatic Parallelization," in *Proc. ACM SIGCOMM*, 2023. [Online]. Available: https://doi.org/10.1145/3603269.3604844

[52] S. Bai, H. Zheng, C. Tian, X. Wang, C. Liu, X. Jin, F. Xiao, Q. Xiang, W. Dou, and G. Chen, "Unison: A Parallel-Efficient and User-Transparent Network Simulation Kernel," in *Proc. ACM EuroSys*, 2024. [Online]. Available: https://doi.org/10.1145/3627703.3629574

[53] Q. Zhang, K. K. W. Ng, C. Kazer, S. Yan, J. a. Sedoc, and V. Liu, "MimicNet: Fast Performance Estimates for Data Center Networks with Machine Learning," in *Proc. ACM SIGCOMM*, 2021. [Online]. Available: https://doi.org/10.1145/3452296.3472926

[54] Q. Yang, X. Peng, L. Chen, L. Liu, J. Zhang, H. Xu, B. Li, and G. Zhang, "DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility," in *Proc. ACM SIGCOMM*, 2022. [Online]. Available: https://doi.org/10.1145/3544216.3544248