USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Enabling AI Network Cross-Layer Design and Operations with Arcadia: A Simulation Platform at Scale

Zhaodong Wang, Satyajeet Singh Ahuja, Xu Zhang, Yuhui Zhang, Max Noormohammadpour, Gregory R. Steinbrecher, Thomas Fuller, Xin Liu, Kevin Quirk, Mikel Jimenez Fernandez, Abhinav Triguna, Yan Cai, and Steve Politis, *Meta Platforms;* Petr Lapukhov and Naader Hasani, *Nvidia;* Ying Zhang, *Meta Platforms*

https://www.usenix.org/conference/nsdi26/presentation/wang-zhaodong

This paper is included in the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation.

May 4–6, 2026 • Renton, WA, USA

ISBN 978-1-939133-54-0

Open access to the Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

Enabling AI Network Cross-Layer Design and Operations with Arcadia: A Simulation Platform at Scale

Zhaodong Wang, Satyajeet Singh Ahuja, Xu Zhang, Yuhui Zhang, Max Noormohammadpour, Gregory R. Steinbrecher, Thomas Fuller, Xin Liu, Kevin Quirk, Mikel Jimenez Fernandez, Abhinav Triguna, Yan Cai, Steve Politis, Petr Lapukhov†, Naader Hasani†, Ying Zhang
Meta Platforms, Nvidia†[*]

Abstract

The rapid evolution of Artificial Intelligence (AI) technology is fueling significant investments by hyperscalers, making AI networks crucial for large-scale training. Understanding the design impacts on AI training requires systematic, cross-layer evaluation. Production experience highlights the need for a robust simulation platform to guide network design and operations. This paper defines the platform requirements, addresses complex design challenges, and shares our experience building Arcadia, a scalable, high-fidelity simulation platform for AI Networks. It operates at the cluster level, focusing on overall cluster performance rather than individual job performance. By using our fast-forwarding, lock-free, and synchronization-cost reduction mechanisms, Arcadia achieves scalability and speed, allowing us to faithfully simulate real-world-scale training clusters and plays a key role in guiding Meta's AI network evolution.

1 Introduction

The demand for AI computational power has surged, growing about 4x annually since 2023. Model scaling is the main driver: LLaMA-2 contained 70 billion parameters [2], LLaMA-3 expanded to 405B parameters trained on 10K+GPUs with 3.4T tokens, while LLaMA-4 reaches 2 trillion parameters doubling the GPU requirement and 40T tokens. This rapid escalation in compute requirements is reminiscent only of the explosive growth in Internet bandwidth during the 1990s, marking a similarly transformative era in technology infrastructure. Modern AI workloads are highly parallel, making networking and interconnects central to system design. GPUs now dominate, offering 100x the bandwidth of CPUs, which shifts network design priorities toward performance, efficiency, scalability, and reliability over feature richness.

These unique demands exacerbate challenges of navigating the design space that are already difficult in general-purpose networks, making them even more complex in AI-specific environments. The networking community is urgently seeking new solutions for these requirements. Amid the excitement, it's vital to balance innovation with practical, data-driven engineering. This paper aims to contribute to this effort by sharing a scalable, production-grade simulation platform that can guide the design, optimization, and operation of AI network infrastructure at Meta.

While simulation is often associated with academic research, we find it indispensable in production environments due to the unprecedented speed of change and the complexity of the design space, challenges that have no prior precedent. AI network design involves a complex combinatorial optimization problem spanning multiple layers, including collective operations, transport protocols, smart NICs, scale-up networks, and scale-out topologies. It creates a vast design space with distinct performance trade-offs influenced by cost, power, and cooling constraints. This complexity is compounded by interdependencies across layers and physical infrastructure, where evolving bottlenecks shift unpredictably as hardware deployments occur asynchronously.

Production AI network decisions involve multi-billion dollar investments that directly impact training outcomes, necessitating simulation systems with exceptional trust and fidelity. This paper introduces, for the first time, the unique requirements of a production AI network simulation platform. First, the simulation system must serve as a platform that allows multiple teams to independently explore design choices such as transport protocols, NIC performance tuning, and topology design. If each team were to build its own simulator, it would result in duplicated engineering effort and prevent joint evaluation. Instead, a unified platform should provide modularity and isolation for team-specific work, while also offering shared building blocks and utility functions to streamline development and improve efficiency. The second key requirement is scalability and performance. The platform must support both small-scale experiments and large, cluster-level simulations, such as those required for GenAI training across tens of thousands of GPUs. Note that simulation is not limited to offline, long-term decision-making; operational use cases require rapid results to predict congestion and prevent performance degradation, requiring simulations to return results within

[*]Work performed while at Meta Platforms.

seconds. Third, accuracy is critical for both operational and design decisions. For operational use, predictions must closely reflect real-world measurement. For design decisions, while some gap between simulated and hardware results is expected, the emphasis is on trend analysis and relative comparisons rather than absolute precision.

In this paper, we present Arcadia, a production-grade simulation architecture designed specifically for AI training clusters. It satisfies these requirements through the following contributions.

- To assist high-level network design questions, unlike traditional network simulators [15, 23, 27, 55], Arcadia operates at cluster-level focusing on training throughput rather than individual job performance—a critical distinction since improving individual job completion times can paradoxically reduce overall cluster utilization.
- A simulation platform supporting multi-granularity and multi-level network simulation needs. Arcadia provides three simulation modes: job-mode treats each training job as a single entity for high-level analysis; flow-mode simulates network at flow granularity assuming steady-state transmission rates; packet-mode models individual packet journeys including switch buffers, Priority Flow Control, and congestion feedback. This design allows users to balance fidelity against execution speed. As a simulation platform, Arcadia also provides common utility to manage input traces, topology library, and continuous validation.
- Novel algorithms for scalability and performance.Simulating millions of training iterations is prohibitively expensive. We propose a fast-forwarding algorithm that simulates only periods where all active jobs complete concurrent iterations, assuming unchanged training speeds during unsimulated portions. Second, parallel execution optimizations achieve remarkable speedup. Through Parallel Discrete-Event Simulation (PDES) with topology-aware partitioning, Arcadia achieves 58.8× speedup over ASTRA-SIM [56] baseline while maintaining accuracy. Third, we propose several optimizations, including propagation-delay-based logical-process partitioning, load-balancing across threads, and per-thread data structures to minimize synchronization overhead.
- Pushing the accuracy limits with continuous calibration in production. We not only conduct extensive evaluation against production measurement, we also make it a continuous calibration system. Continuous calibration is essential due to daily changes in firmware, libraries, hardware, and optimizations, especially when other dependent systems may change, e.g., training model, data ingestion process. Overall, Arcadia achieves 95%+ accuracy through systematic validation against Meta's production training clusters, with mean absolute percentage error under 5% for packet-mode and 10% for flow-mode simulation.

Arcadia has operated as a production service at Meta for three years, playing a critical role in managing real-world AI networks and enabling exploration of "what-if" scenarios. Over the past three years, it has supported more than ten teams across 50 use cases, including cross-model design, routing, bandwidth planning, job scheduling, and capacity management. This paper presents an overview of these analyses, revealing for the first time the complex interdependencies and design decisions encountered in production environments. We further examine several case studies, such as topology design, switch buffer sizing analysis, and congestion prediction, to illustrate how Arcadia provides quantitative validation for design choices and uncovers unexpected details. These examples demonstrate the platform's value in confirming decisions and deepening understanding of intricate system behaviors.

By describing how Arcadia is used at Meta, this paper offers unique insights into the operational challenges and design decisions that emerge across multiple generations of training cluster evolution. Sharing our experiences and lessons learned, we highlight the importance of simulation platforms in the new era of AI and aim to inspire future research in this area. We also plan to open source Arcadia in the future.

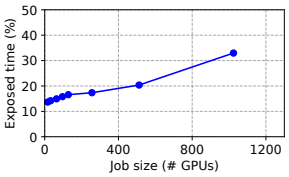


Figure 1: The fraction of exposed network time to end-to-end job training time with job size.

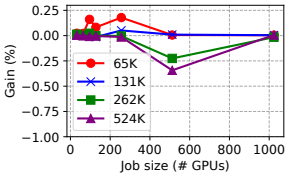
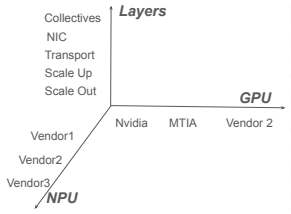


Figure 2: Performance gain over the baseline's training time across different training data sample sizes (Bytes).



Design space	Use cases
Scale Out	Network scale (§9.1), Fiber (§9.1), Topology (CLOS, BCube [30], Rail optimization [53])
Scale Up	Switch buffer sizing (§9.2), In-network compute [41]
Routing	ECMP, OSPF, DPR, QP scaling (§9.3), Congestion prediction (§10.1)
Transport/NIC	RoCE [56], DCQCN [72], RDMA NIC, UEC
Collectives	NCCL, Optimized AllReduced
Job	Job scheduling, Job placement, Host utilization (§9.1), Proactive job migration
Model	LLM [2], Recommendation [48], Image segmentation [40], MoE
Others	Cost-performance trade-off, Reliability

Figure 3: Complex Design Space in AI Networks

2 Motivation

We motivate the need for a production simulation platform.

2.1 Importance of AI Training Network

Transitioning from CPU- to GPU-based training, Meta's infrastructure supports large-scale, distributed systems to meet expanding AI requirements. Meta's training clusters use state-of-the-art designs [39, 42], with servers that have both CPUs and GPUs connected by NICs and RDMA technologies like RoCE [47].

As models continue to grow in size, network performance increasingly becomes the bottleneck. To show the increasing importance of network performance, we examine production data of some recommendation models (such as [40]) from one of Meta's clusters over a two-month period. These training jobs vary in size, and we group them by job size and overall

training sample sizes, measuring their *exposed network time*. We define exposed network time as the duration spent exchanging updated gradients that is *not* overlapped by gradient computation. Figure 1 shows the fraction of exposed network time relative to total training time for different job sizes. Given that the scaling law [31] continues to hold for neural networks, future larger models will drive even more traffic through the training cluster.

2.2 Complex Design Space in AI Training Network

Large design space: Our AI network design presents a combinatorial optimization challenge across multiple dimensions. As we illustrate in Figure 3, the design space spans five logical layers: collective operations (AllReduce, AllToAll), transport protocols (RoCE, MetaRoCE, TCP), smart NIC generation, scale-up networks (NVLink, MTIA solutions), and scale-out topologies (CLOS, rail-optimized). For example, to support high performance, we employ various technologies (NVLink, PCIe, GPUDirect RDMA [45], and Ethernet) for intra- and inter-host GPU communication.

Platform diversity amplifies complexity. Hardware combinations include Nvidia (multiple generations), AMD, and Meta's MTIA accelerators spanning multiple generations [5]. This creates a 4×3×5 design sub-space with distinct performance characteristics. While not all combinations are equally important, the prioritization in production requires closed examination across cost constraints, hardware availability, power limitations, and cooling requirements.

Complex cross-layer dependency: The complexity is further amplified due to interdependencies across layers. First, the same optimization may yield different benefits for different models. For example, accelerating the AllReduce operation has limited impact on DLRM [40] compared to a GenAI model [2], given that most collective operations in DLRM are AllToAll. Although AllReduce is dominant in GenAI, its performance hinges on the design choices on multiple layers, such as collective algorithms (*e.g.,* Ring vs. Tree), routing (*e.g.,* E-ECMP [22]), and switch hardware (*e.g.,* buffer design [7] or in-network computing [8]). Second, as infrastructure evolves, bottlenecks shift unpredictably—hardware upgrades don't occur uniformly, making any stage from PCIe to packet processing a potential constraint(*e.g.,* host congestion [12], packet processing [58, 59], PCIe [30], etc.). On the software side, improvements in one module (*e.g.,* updating NCCL) can have heterogeneous effects across collective operations and message sizes, necessitating co-tuning with backend network configurations [22]. Different environments thus require different parameter settings.

Need a Simulation Platform for Collaboration: Both the unprecedented speed of change and its complexity require cross-team coordination. Multiple teams optimize distinct system components while seeking to understand global performance impact of local optimizations. This necessitates simulation frameworks that accurately model component interactions across the entire stack, from customized accelerators and high-speed interconnects to software optimizations and network design decisions.

Production example of cross-layer dependency: The strategy to provide high performance for individual job does not necessarily translate into high performance for the entire cluster. Job scheduling strategies and GPU utilization are equally important to overall cluster performance. To explain this point, we conduct a trace-driven simulation using historical data from Meta's production environment (the same dataset used in Figure 1 based on our production scheduler [17]). We shorten each job's training time in the trace and replay the modified trace using a FIFO job scheduler (details are in §A). We then compute the performance gain based on the overall time to complete all training jobs. Figure 2 presents the resulting performance gains, grouped by job size and training data sample size. Surprisingly, we find that even though every individual job completes more quickly, the total completion time of those jobs can sometimes increase. The reason is lower overall cluster utilization: the FIFO scheduler attempts to fill newly available cluster slots with the first job that fits, sometimes leaving GPUs unused. Thus, network planners must consider job and cluster performance jointly. Understanding cluster-level performance requires simulating the cluster over an extended time frame, rather than focusing on a single job.

2.3 AI Network Operations Guidance from Simulation

Network operations, such as draining a rack switch for maintenance, can have significant impacts on ongoing AI training jobs. Unplanned or poorly timed interventions risk interrupting long-running jobs, stranding expensive GPU resources, and causing cascading delays across the cluster. At Meta, we have risk prediction systems to decide if a network operation is safe to execute and when to execute [57]. For AI backend networks, Arcadia performs such risk assessment and evaluation of alternative operational strategies. For example, Figure 2 highlights how a simple FIFO policy can lag behind a production scheduler [17], which delays low-priority jobs, staggers job starts to reduce GPU stranding, and allows preemption. Given the high cost of AI infrastructure and the extended duration of training jobs, careful planning of network usage and maintenance activities yields substantial operational and financial returns. Simulation empowers operators to anticipate the consequences of network changes, optimize scheduling and resource allocation, and ensure that maintenance actions are performed with minimal disruption to production workloads.

3 Related Work

Training Cluster Design. Maximizing training performance under budget constraints requires identifying end-to-end bottlenecks across GPUs, CPUs, storage, and the network. Simulation is a practical tool for attributing runtime to each component and guiding hardware upgrades, software optimizations, and hardware–software co-design [14, 24, 29, 38, 62]. Meta follows a co-design approach spanning custom accelerators [5, 37],

	Simulation Platform	Cluster Scalability	AI-Model Context	Continuous Calibration
FSIM [49]	No	Yes	No	No
NS3 [27]	Yes	No	No	No
ASTRA-SIM [56]	No	No	Yes	No
UNISON [15]	No	Yes	No	No
DONS [23]	No	Yes	No	No
SIM-AI [55]	Yes	Yes	Yes	No
Arcadia	**Yes**	**Yes**	**Yes**	**Yes**

Table 1: Comparison of representative simulators.

GPU hosts [36], and protocols [22].

Simulators. Prior simulators do not fully meet our production needs (Table 1). Packet-level tools (*e.g.,* NS-3, OMNeT++) are accurate but do not scale to hyperscale cluster sizes [27,35,52]. Flow-level simulators scale better but are too coarse for diagnosing AI training performance and adapting to diverse configurations [28,49,54]. Hybrid and data-driven approaches narrow this gap, yet typically lack job-level abstractions and tight integration with AI workloads [32,34,51,61]. AI-specific simulators (SIM-AI, ASTRA-SIM) provide limited abstraction flexibility, calibration, and production alignment [55,56]. These gaps motivate a simulator that is scalable, extensible, and continuously calibrated against production to provide reliable operational insights.

4 Arcadia Design

In this section, we provide an overview of Arcadia's design and a sketch of its workflow.

4.1 Design principles

In this paper, we share the following insights gained from developing and operating production-grade simulators. First, the principle governing accuracy: trend analysis and relative comparisons matter more than absolute precision for long-term planning, but proximity to production deployment requires higher accuracy. Second, continuous calibration represents a critical requirement. Production environments evolve daily with firmware updates, NCCL library changes, hardware replacements, and performance optimizations. Static one-time calibration leads to compounding errors and misleading conclusions. Third, fidelity-performance tradeoffs demand flexibility. Different use cases require varying accuracy levels—fault-tolerance strategy exploration may not need precise network timing, while production parameter tuning demands high fidelity.

4.2 Design Overview

Arcadia design is centered around three key areas: simulation platform, scalability and accuracy. During each simulation run, Arcadia predicts the training performance of a set of jobs (defined in §5.2) in a training cluster. Users can specify Arcadia's fidelity level (§6) by selecting a simulation mode (job, flow, or packet), adding or overriding modules, or skipping part of the timeline within Arcadia.

Simulation as a Service Arcadia is achieves the platform support through multiple design choices. First, it has a generic abstraction based on the Discrete Event Simulation (DES) framework [11]. Users can add or override modules via an input configuration to adjust simulation fidelity or explore alternative algorithms and policies. Second, it provides a set of common functions needed for all types of simulation experiments, such as workload modeling, topology management. Third, to answer high-level design questions for network interconnect as a whole, we design cluster-level simulation, by faithfully simulating job level scheduling, failure, and recovery lifecycle events.

Multi-mode Simulation and flexible fidelity Arcadia pushes the scalability and performance limits through multiple ways. First, it supports simulation at three modes with different granularity and performance. Second, it employs a set of novel optimization methods for parallelization and speedup. Third, it can be configured to tune the tradeoff between scalability and speed. Figure 4(a) illustrates the workflows for different fidelity levels (*i.e.,* job, flow, packet levels). In each workflow, once a module finishes, it emits an event that, by default, triggers the next module. Alternatively, users can emit custom events to bypass certain steps or trigger their own modules. For more complex behaviors, users may also modify the classes shown in Figure 4(b). Rebuilding Arcadia with these new modules seamlessly integrates them into the simulation.

Continuous accuracy calibration in Production High fidelity is essential for a production-grade simulation platform, particularly when it is used to detect operational risks and assess feasibility in live environments. Unlike previous simulators that only have one-time accuracy evaluations, Arcadia implements a continuous validation pipeline that regularly ingests production data and performs ongoing calibration. Achieving and maintaining high accuracy in the face of production noise and constant change is challenging, requiring an iterative refinement process. Insights gained from each calibration cycle has been used to directly improve Arcadia, ensuring it remains aligned with real-world conditions.

4.3 Simulation Workflow

Arcadia is a discrete-event simulator: it does not execute real training jobs. Instead, it simulates job arrivals, scheduling, GPU/network execution, and failures to produce timing estimates. In the simulation, when a job-arrival event occurs, the (simulated) scheduler selects a start time and allocates GPUs to that job. Each job is described by model parameters (model type, number of iterations, training data location, priority, etc.) and a training workload DAG (defined in §5.2). Given these parameters and the current simulated cluster state, Arcadia predicts the job's per-iteration time by accounting for both GPU-host execution, network communication and GPU host allocation. The predicted iteration time can vary over time due to contention with other concurrently running simulated jobs. In addition to normal completion, a job may be interrupted by simulated hardware failures or preempted by higher-priority jobs. In such cases, the job state rolls back

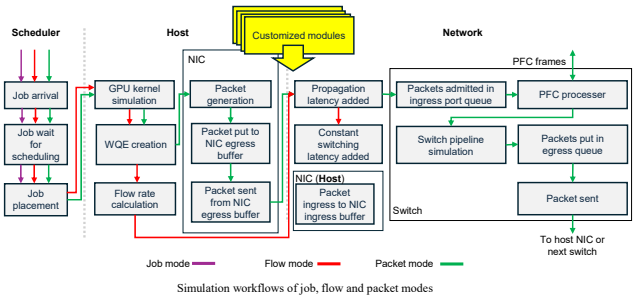
to its most recent checkpoint and returns to the (simulated) job pool for rescheduling. To implement checkpoints, we store the number of remaining training iterations in each job's state (Figure 4(b)); this iteration-level granularity is suitable because an iteration typically lasts only a few seconds on real GPUs.

It is worth noting that Arcadia 's simulation workflow is not Meta-specific: it captures the general life cycle of distributed AI training jobs. What is Meta-specific in our evaluation is the *instantiation and parameterization* used to run the simulations. On the hardware side, our GPU clusters are largely dedicated to AI workloads, enabling co-design of components and operating envelopes (e.g., MTIA [37] and GrandTeton [36]) and site-specific constraints such as power capping and thermal throttling. On the software/network side, we use workload-optimized stacks (*e.g.,* MetaRoCE [10]) and Meta-specific scheduler policies and trace formats. Arcadia is portable because these choices appear as pluggable modules (scheduler, network/transport) and calibrated parameters (compute/communication/storage performance, availability, and failure/preemption); users outside Meta can substitute their own topology and measurements (or vendor specs) to fit these inputs.

5 Simulation As A Service

5.1 Training workload modeling

As is common practice, Arcadia models the workloads of training jobs as Directed Acyclic Graphs (DAGs), describing the sequence of computation and communication stages along with their dependencies. The network and host simulators handle communication and computation stages, respectively. Computation stages typically involve algorithm modules, submodules, or GPU kernels, while communication stages include collective algorithms or peer-to-peer communications. The dependency between these stages is their execution order; model-, data-, tensor-, or pipeline-parallelism can then be applied to the dependency as optimizations (See Figure 25 for an example). The DAG represents the end-to-end execution timeline of training workloads across all system components. In the context of computation–communication overlap, a large computational kernel (*e.g.,* a matrix multiplication) can be partitioned into smaller sub-kernels. Each sub-kernel produces intermediate results that trigger dependent communication operations, such as gradient exchanges. By pipelining these fine-grained computation and communication tasks, the system enables overlap between backpropagation and parameter synchronization, thereby reducing idle time and improving training throughput. Additionally, other than communication and computation dependencies, Arcadia also requires users to specify the location of training data storage and job priority for scheduling purposes. A natural question is *why do we ask Arcadia users to explicitly specify the workload DAG?* Unlike prior work [55], which takes a more direct approach by requiring users to provide the framework type (*e.g.,* Megatron [3],



Simulation workflows of job, flow and packet modes

Figure 4: Arcadia provides extensible, cross-layer simulation workflows, allowing users to choose between job, flow, and packet modes or customize their own modules and workflows.

DeepSpeed [1]) and a set of parameters (*e.g.,* world_size, num_experts) for computation stages, then call NCCL [6] for communication to infer the workload DAG, we adopt this approach for the following reasons:

(1) Many models in Meta's code base use *generic* frameworks (*e.g.,* `torch.distributed.pipeline` [9]) that require manual and detailed setup and cannot be fully described by a small set of parameters.

(2) Capturing a model's *training-workload* DAG is practical in a trace-driven manner: production logs already record timestamped compute phases and communication events (e.g., collective type and tensor/message size). From these traces, we reconstruct dependencies (program order and collective-induced synchronization) using standard techniques such as those in [62]. We validate DAG fidelity by cross-checking that the reconstructed event order and collective sizes match independent profiling/traces from real runs under the same training configuration.

(3) An explicit DAG makes it convenient for engineers to experiment with new schedules, kernels, and communication-computation overlap strategies before deploying them on training clusters.

(4) Non-Meta users can get workload DAGs by using distributed training frameworks (*e.g.,* Megatron [3]) to log compute and communication events along with their timing. For what-if and sensitivity studies, users can instead synthesize parameterized DAG templates: compute-node durations are set using microbenchmarks or vendor specifications, and communication nodes are derived from the chosen parallelization strategy (*e.g.,* DP/TP/PP) and tensor sizes.

5.2 Cluster-level simulation

The jobs that share the compute and network resources in training clusters can affect each other's performance, as well as overall cluster utilization. It is essential to model training job workloads and simulate job scheduling.

Arcadia supports various scheduling algorithms, including First-Come, First-Served (FCFS), Smallest Job First (SJF), and MAST [17] (Meta's production scheduler). When the

cluster can accommodate a new job or a high-priority job needs to preempt a lower-priority job, Arcadia uses the user-specified algorithm to select GPUs for that job. Once the job is placed, Arcadia runs host and network simulations to update the cluster state. If the training data is not co-located with the GPU hosts allocated to the job, Arcadia sets a communication stage for training data transmission before training starts.

6 Multi-mode Simulation for Scalability

Not all use cases need high-fidelity network simulations. For example, when studying GPU utilization in a training cluster, fine-grained network prediction is not necessary, as AI jobs generally run for long durations and we do not require performance tuning. To address diverse needs, Arcadia offers multiple simulation modes that provide different levels of fidelity and execution speed.

6.1 Host simulation

Although Arcadia emphasizes network performance, estimating training performance on the host side is essential. For computation time, Arcadia collects execution time measurements for each GPU kernel from production data (*e.g.,* through CUDAEVENTRECORD) or reruns the kernel on the host for simulation. The predicted kernel runtime is taken as the average of these measurements or runtimes. Consistent with observations in prior work [55], we do not observe large runtime variability for the same kernel on the same GPU. Note that Arcadia only simulates hosts equipped with GPUs and not those used solely for disk-based training data storage. The latter do not lie on the critical path of end-to-end model training and only contribute network flows for data transfer to GPU hosts. For GPUs that do not exist, we expect users to define their behaviors or processing speeds.

Other than executing GPU kernels, hosts initiate network traffic via the following steps: 1) execute the collective algorithm, 2) create work queue entries (WQEs) for RDMA NIC, and 3) send or receive network traffic. For step (1), we use self-defined algorithms [22] through Arcadia 's module interface to select the logical topology on the backend network for collective operations (*e.g.,* Ring vs. Tree for AllReduce) instead of hijacking NCCL in prior work [55], since NCCL's implementation [22] can be suboptimal in our production environment. For step (2), hosts may create RDMA WQEs for collective operations and read the data from memory. We simply assume that the latency is constant [1]. The total traffic generated by each GPU host is defined by the workload DAG. GPU-to-GPU and GPU-to-NIC traffic volumes are then determined by the collective algorithms. Finally, the network simulator models the transmission of these flows.

6.2 Network simulation

Arcadia provides three simulation modes—*job*, *flow*, and *packet*—each offering a different balance between fidelity and execution speed. Figure 4 shows the granularity differences among these modes.

1. Job-mode simulation: This mode treats each training job as a single entity without modeling host or network details. Instead, users specify a job's completion time (or its distribution), the number of required GPUs, and the arrival time. This approach is primarily used to assess high-level aspects of cluster design, such as job queuing algorithms or fault-tolerance strategies (*e.g.,* hot vs. cold spares).

2. Flow-mode simulation (FSM): In this mode, FSM simulates the network at a flow granularity. A *flow* is defined as the amount of data exchanged between two GPU hosts. We assume each flow transmits continuously at a steady rate, effectively simplifying NICs to packet generators and consumers, and switches to rate limiters, without packet-level details. Once routes are chosen by the routing algorithm, a flow's rate is set by the slowest link on its path. Whenever a new flow enters the network, FSM recalculates rates for all active flows to reach a new equilibrium of maximum bandwidth usage without congestion (see §B for details). For a flow, end-to-end transmission time includes data transfer time, host latency, switching delay, and propagation delay.

3. Packet-mode simulation (PSM): In packet mode, Arcadia simulates the journey of individual data packets across the network. We treat each RDMA queue pair (QP) as a packet source. For GPU-to-GPU communication, the sending rate is the NVLink's bandwidth. In Arcadia, we focus on the backend, simulating GPU-to-GPU communication as flow mode. For backend network communication, each QP starts by generating packets at a fixed interval. Packets enter a NIC queue, which transmits at its capacity. When packets reach a switch, PSM creates an packet_admit event to place the packet in the port's ingress queue. At the same time, it invokes a Priority Flow Control (PFC) [46] checker to see if the queue occupancy exceeds a specified watermark. If it does, the switch issues a pause signal to halt upstream transmission until congestion resolves or a timeout elapses. A pipeline_pkt_processing event then processes packets in the ingress queue after a fixed delay (simulating switch-processing latency) before enqueuing them for the egress port. In addition, each switch port can host multiple queues, enabling scheduling algorithms like WRED [18], strict priority (SP), or weighted round robin (WRR) [19]. Users can override these packet-processing events to test custom switch behaviors. At the destination NIC, a packet_arrival event triggers any corresponding protocol actions (*e.g.,* dynamic congestion control in DCQCN [63]), allowing the sender NIC to adjust its send rate if necessary (*e.g.,* when inter-packet arrival exceeds 50 μs for packets tagged with a specific CNP bit). In this manner, PSM captures packet-level effects like buffer occupancy and congestion feedback, enabling high-fidelity analysis of network behavior.

6.3 Fast forwarding mode

Other than the three simulation modes, we offer a "fast-forwarding" knob that reduces runtime by trading off some

[1] Note that this latency depends on the host architecture. In our production environment, it typically remains at the sub-millisecond level, which is significantly shorter than the runtime of GPU kernels.

fidelity. Simulating every iteration of a training job can be prohibitively expensive, especially under packet mode, because jobs may run for millions of iterations. Instead, we simulate only a period in which all active jobs complete a few iterations concurrently (an example in Figure 26). During the period that is not simulated, we assume each job's training speed remains unchanged so long as no new jobs arrive and no existing jobs complete. More concretely, let $J_1, \ldots, J_N$ be the jobs sharing the network at time J_{start}, and assume J_M is the slowest job, requiring T^M_{oneiter} to finish one iteration. We simulate the system for $n \times T^M_{\text{oneiter}}$, where n is a user-chosen parameter. Once these n iterations are complete, we fast-forward to the time when the first job among $J_1, \ldots, J_N$ finishes, and then update the network state as if it had been continuously running (but without the load from the completed job). If new jobs arrive, we add them to the cluster and repeat the same process. This approach can dramatically cut simulation time while preserving essential insights.

6.4 Performance Optimization

Simulating a training cluster involves multiple concurrent jobs, which demands fast execution, especially for PSM. Arcadia accelerates execution by adopting a Parallel Discrete-Event Simulation (PDES) framework [21]. We implemented PDES [27] with additional optimizations that leverage the characteristics of AI training workloads.

Propagation-delay-based LP partitioning: Within the PDES framework, the network's nodes (*i.e.,* hosts and switches) must be divided into logical processes (LPs). Events within each LP are processed in time order, and LPs run in parallel. At the start of each round, an LP can only execute events up to the time $\min\{E_i\} + lookahead$ [21], where E_i is the next event's timestamp for the i-th LP, and *lookahead* is a fixed interval—typically the minimal packet travel time on any inter-LP link. After each round, all LPs synchronize for packets moving across LPs. The round's runtime is $\max\{P_i\} + C_{\text{sync}}$, where P_i is the processing time of the i-th LP's events for that round, and C_{sync} is the synchronization overhead. To minimize C_{sync}, we aim to reduce the packets traveling between LPs. To minimize $\max\{P_i\}$, we want as many LPs as possible, balanced in workload. However, increasing the number of LPs can also raise synchronization costs, and we might not have enough cores. In Arcadia, we partition nodes based on link propagation delay: if two directly connected nodes have a propagation delay below a threshold ζ, they are assigned to the same LP. This approach clusters physically close nodes, such as GPU hosts in the same rack or row in a rail-optimized topology, into a single LP. Since our production scheduler typically allocates GPUs for a single training job to nearby hosts (a strategy known as gang scheduling [17]), one LP usually has only one job's traffic, reducing the traffic across LPs. Thus, C_{sync} is reduced. For nodes at higher tiers in the network hierarchy (*e.g.,* aggregation or spine switches in a Clos topology), which are in different rooms from GPU hosts, we assign them to different LPs. It raises *lookahead* value between LPs and reduces the number of synchronization rounds. Note that, choosing an appropriate ζ depends on the cluster's job-size distribution (not shown). In practice, we build a table mapping the average job size to a proper ζ.

Parallelization and memory management:

- *Load-balancing LPs across threads*: By default, we often assign LPs to threads using round robin, which can lead to load imbalance. In our experiments, the slowest thread was up to 10× slower than the average, forcing all other threads to wait. We apply the classic algorithm from [26], which checks thread utilization every 2 seconds of wall-clock time, and if imbalance exceeds 5%, redistributes LPs across threads. This yields a 1.45× speedup.
- *Per-thread data*: Existing PDES implementations typically use global variables (*e.g.,* counters for scheduled but unexecuted events). We replace these with per-thread variables and aggregate results only when needed. Although each query requires collecting data from multiple threads, queries are not frequent. This yields a 2.3× speedup.
- *Process flows in parallel for FSM*: Other than PSM, we also accelerate execution by processing network flows in parallel. We partition the flows and links into segments such that no two flows in distinct segments share any links. This segmentation is achieved by building a bipartite graph whose vertices represent flows and links, with an edge between a flow vertex and a link vertex if the flow traverses that link. Although various algorithms (*e.g.,* disjoint sets, depth-first search DFS) can identify connected components (segments) in this graph, we opt for DFS due to its simplicity and low overhead, as they do have significant differences in performance. Since these segments typically remain stable after new job arrivals or completions, we update the partitions only once every five job updates.

7 Continuous Fidelity

The performance of Meta's training cluster is continually improving, so predictions from Arcadia that are accurate today may not hold true in the future. In this section, we explain how we regularly align Arcadia with the latest production environment to maintain up-to-date accuracy.

7.1 Aligning simulation results to production scale

Because we use Arcadia's predictions for network management (see §10.1), it is essential to align the predicted training time of AI jobs with real production measurements. To achieve this, we apply linear regression to Arcadia's initial predictions, incorporating the total number of GPUs as an explanatory variable. The refined prediction is:

$$T_e = \alpha_1 N_g + \alpha_2 N_g \cdot T_i + \alpha_3 \log(N_g) + \alpha_4 (1/N_g) + \gamma \cdot T_i + c,$$

where T_e and T_i denote the enhanced and initial predictions, respectively, and N_g is the total number of GPUs in the cluster.[2]

[2]It was validated only within Meta's training centers [22] and has not been tested outside.

The coefficients $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \gamma$, and c are obtained through training.

7.2 Continuous calibration

Since Arcadia cannot fully simulate every detail of the infrastructure, it relies on constant parameters (*e.g.,* the delay for creating WQEs, packet processing delay on switches or hosts) that may become outdated as hardware and software evolve (*e.g.,* due to library updates such as newer NCCL versions). These changes can impact performance, as highlighted by developers [25]. Manually setting or updating these parameters presents two key challenges. First, conducting a comprehensive daily parameter sweep across all devices is infeasible. Second, low-latency telemetry is difficult to measure accurately and isolate from other system factors. Third, not all production updates map cleanly to a corresponding module in the simulator, which often uses simplified abstractions. To address these issues, we leverage end-to-end network flow completion times observed in production (which are easy to measure) and compare them with Arcadia's predictions. Parameter values are then adjusted to minimize the squared error between simulated and measured completion times:

$$\arg\min_{\Phi} \quad \sum_i \left(T^i(\Phi) - T^i_{prod}\right)^2, \tag{1}$$

$$\text{s.t.} \quad T^i(\Phi) \geq T^j(\Phi) \Rightarrow T^i_{prod} \geq T^j_{prod}, \tag{2}$$

where $T^i(\Phi)$ and T^i_{prod} represent the simulated and actual completion times for flow i under the parameter set Φ, respectively. The constraints in Eq. (2) represent the ordering of completion times between flows (*i.e.,* the simulator should maintain the same order of flow completion times as in production), ensuring the accuracy of what-if analysis. This ordering-optimization problem is non-convex and NP-hard. To solve this problem, we add a penalty function to transform the optimization goal to

$$\begin{aligned}\arg\min_{\Phi} \quad & \sum_i \left(T^i(\Phi) - T^i_{prod}\right)^2 \\ & + \lambda \sum_{i<j,\, T^i_{prod} < T^j_{prod}} \left[\max(0, T^i(\Phi) - T^j(\Phi))\right],\end{aligned} \tag{3}$$

where λ is a regularization coefficient that penalizes violations of the production completion-time ordering. Using a hill-climbing algorithm to solve Eq. (3), we iteratively refine these parameters and verify that each updated parameter set aligns with hardware specifications.

7.3 Learnings from calibration experience

- *Aligning the data type with production data:* Large-scale training commonly uses mixed precision (*e.g.,* BF16/FP16) rather than FP32 for many tensors that participate in communication (*e.g.,* gradients and/or activations, depending on the framework and configuration). Because communication volume scales with the element size (4 bytes for FP32 vs. 2 bytes for BF16/FP16), assuming the wrong dtype can introduce up to a 2× error in per-collective message sizes, which directly propagates to communication time and overall training-time estimates. Therefore, when constructing the workload DAG, we compute each flow size as # elements exchanged × bytes-per-element of the *actual* tensor dtype (plus any packing/padding required by the communication library), rather than using only the number of parameters.
- *Aligning packet overhead:* In production, payload bytes are not sent directly; each message is packetized and encapsulated by a protocol stack, adding per-packet headers/trailers, and large messages are segmented into many MTU-sized packets. Thus, the wire bytes equal the payload plus repeated per-packet overhead, reducing effective goodput—especially for small messages and latency-sensitive collectives. Concretely, with a 1500 B MTU and a typical RoCEv2 encapsulation, per-packet overhead is on the order of ~50–60 B, leaving ≈ 1460 B of RDMA payload per packet; overhead is only ~4% for MTU-sized transfers but can exceed ~18% for a 256 B message (and is even higher for smaller payloads). Thus, we need to model packetization using the production MTU and protocol stack, converting each DAG communication node into a packet stream and accounting for header overhead when estimating transmission time.
- *Aligning unsimulated overhead:* No simulator, including Arcadia, can capture every detail of a training system. As discussed in §7.2, we propose a calibration method that updates constants representing the execution time contributed by unsimulated system components. In Eq. (3), our optimization objective explicitly mitigates the risk of overfitting to a limited set of production traces. As more traces from the production environment become available, the calibration process refines these constants, further reducing overfitting and improving fidelity.

8 Evaluation

We evaluate Arcadia by comparing its predictions to actual training times from Meta's production environment and assessing its simulation performance.

Key findings:

- Arcadia achieves continuous, accurate predictions of AI model training times in production. The gap between Arcadia's predictions and real-world results at Meta is minimal: roughly 5.0% for PSM and 10.1% for FSM. Arcadia provides a wide range of fidelity options.
- Arcadia efficiently simulates large training clusters. Relative to the ASTRA-SIM baseline [27], Arcadia achieves up to 58.7x speedup on a 40-core server.

8.1 Methodology

We benchmark Arcadia by replaying a three-month period of AI model training jobs from a Meta's production cluster. We focus on FSM (flow-mode) and PSM (packet-mode)[3] to estimate the completion times of models during this pe-

[3]We omit the job-mode results here because it is not for training-time predictions.

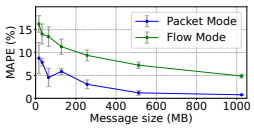


Figure 5: MAPE across various models with different message sizes

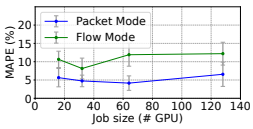


Figure 6: MAPE across various models with different job sizes

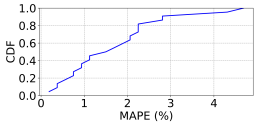


Figure 7: Host simulation accuracy

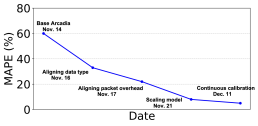


Figure 8: Accuracy breakdown

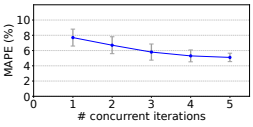


Figure 9: MAPE with different numbers of iterations to run for fast forwarding

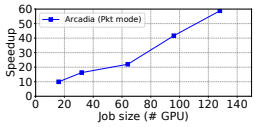


Figure 10: Speedup over ASTRA-SIM across job sizes

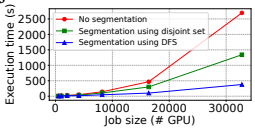


Figure 11: Comparison of various accelerations across different job sizes for FSM

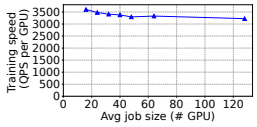


Figure 12: Training speed vs. Average job size

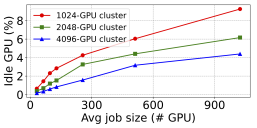


Figure 13: Avg job size vs Idle GPU for different sizes of training clusters

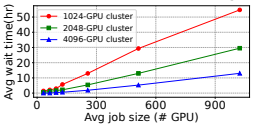


Figure 14: Avg job size vs Avg waiting time in the queue for different sizes of training clusters

riod. We compare Arcadia against simulators with Meta's production settings. Therefore, we use ASTRA-SIM [56] with barrier synchronization algorithm [27], another AI training simulator developed by Meta. ASTRA-SIM emphasizes software-hardware co-design on the host side and integrates a backend network model using NS3. For a fair comparison, we only evaluate their respective backend network components. Note that Arcadia 's GPU kernel execution-time predictions rely on production logs, whereas ASTRA-SIM, by design, does not use actual production logs.

Setup: Arcadia is configured to mirror one of Meta's training clusters with 4,096 GrandTeton GPU hosts [36], each with 8 H100 GPUs [43]. These hosts are in a three-level Clos network. Within each host, GPUs are interconnected by GPU Direct [45] at 400 Gbps non-blocking bandwidth; each host has an RDMA-enabled NIC providing a 400 Gbps link to other hosts. The overall cluster has 4,096 leaf switches, 1,024 aggregation switches, and 128 spine switches. Arcadia runs on a machine with two 20-core CPUs (2.4 GHz each) and 223 GB of memory.

Metrics: We measure Arcadia 's accuracy using the mean absolute percentage error (MAPE). For a model with predicted training time T_p and actual training time T_a in production, MAPE = $|T_p - T_a|/T_a$. We also measure Arcadia 's execution time to demonstrate scalability for large-scale cluster simulations, and report speedups as the ratio of the baseline's runtime to Arcadia 's runtime.

8.2 Accuracy

Overall accuracy: Figure 5 shows how MAPE decreases as message size increases for collective operations in both FSM and PSM. Larger messages are more bandwidth-bound than latency-bound, making it easier to predict their completion time accurately. Figure 6 indicates that job size is not strongly correlated with MAPE; even large jobs sharing links and switches with others do not significantly affect accuracy.

Host accuracy: To assess Arcadia 's host simulation, we randomly select 25 AI model-training jobs that ran on Meta's GrandTeton [36] GPU host instrumented with performance profilers (*e.g.,* PyTorch.Profiler). Figure 7 shows the CDF of prediction errors. We can see that for 60% of predictions, the error is under 2.0%.

Accuracy breakdown: Figure 8 shows how Arcadia 's accuracy (shown for PSM; FSM follows a similar trend but is not shown) has improved over time through the methods described in §7. Dtype and packet overhead alignments contribute the most in accuracy, since they directly determine the amount of data exchanging in the network. Once continuous calibration was introduced, Arcadia 's accuracy stabilized (not shown). If we stop calibrating Arcadia, after four months, validation on DHEN (a click-rate prediction model [60]) shows a 3.27% drop in accuracy. In production, running PSM for every iteration of every job is infeasible due to excessive simulation time. Thus, we employ fast forwarding (§6.4) to simulate only K iterations per job, inevitably affecting accuracy. Figure 9 shows that while larger K yields better accuracy, it also increases runtime. By contrast, FSM is efficient enough to simulate large-scale clusters without fast forwarding. We evaluate the sensitivity to the regularization coefficient in §C.1.

8.3 Performance

Packet mode (PSM): Figure 10 shows that PSM achieves a 9.9×–58.8× speedup over ASTRA-SIM, making large-scale cluster simulations feasible. In addition to the optimizations discussed in §6.4, a key difference lies in the parallel computation frameworks: ASTRA-SIM's NS-3-based implementation rely on Message Passing Interface (MPI) [4], which is geared toward distributed computing across multiple machines rather than multi-threaded execution on a single server. Consequently, MPI introduces extra communication overhead in a multi-threaded context. By contrast, Arcadia is specifically designed for multi-threading, reducing communication overhead and leveraging modern multi-core servers, such as the 166-core machines readily available at Meta.

Flow mode (FSM): Figure 11 shows FSM 's scalability, comparing its bipartite graph optimization against both single-thread (712% faster) and disjoint-set–based multi-thread approaches (354% faster). As the number of GPUs scales, Arca-

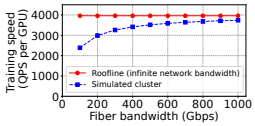


Figure 15: Fiber speed vs Avg job training speed

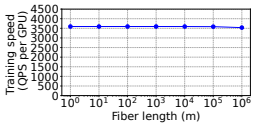


Figure 16: Fiber length vs Avg job training speed

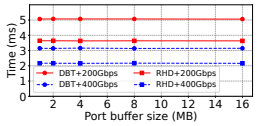


Figure 17: Per-port buffer size at spine switches vs Job completion time

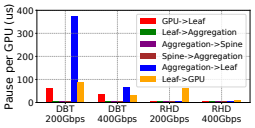


Figure 18: Port pause time per GPU at different switches for all-reduce

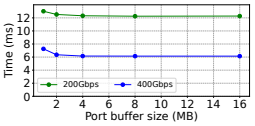


Figure 19: Per-port buffer size at aggregate switches vs Job completion time

dia effectively parallelizes across multiple threads to handle different connected components, avoiding a dramatic increase in runtime.

The size and type of models also have impact on simulation performance. We evaluate the model impact and scalability in §C.2 and §C.3.

9 AI Network Insights with Arcadia

Arcadia has played an important role in AI infrastructure development across multiple teams at Meta, including model design, hardware development, and routing optimization. Figure 3 showcases over 20 successful use cases, and we highlight three notable examples.

9.1 How to Scale AI Cluster

As AI models grow, training jobs increase in size and place higher demands on the AI network. A key challenge is scaling the AI network. Specifically, how large should the cluster be to mitigate GPU stranding and fragmentation? What is the most effective way to physically interconnect GPUs? How does network bandwidth affect QPS?

Setup: Using FSM, Arcadia simulates a training cluster for DLRM models [40]. We sample job sizes from Meta's production traces and maintain a 500-job queue. Scheduling is handled by MAST [17], our production scheduler. We measure the steady-state training speed (QPS, *i.e.,* training samples per second) of a three-level CLOS network with a spine:aggregation:leaf ratio of 1:8:32. Each leaf switch connects to a GrandTeton host containing 8 H100 GPUs.

GPU Stranding: Figure 12 shows that the per-GPU training speed (QPS) decreases by about 10.1% as average job size grows, due to heavier inter-GPU traffic. However, the slow decline implies that different jobs' network flows intersect minimally despite rising traffic. Figures 13 and 14 show that idle GPUs increase while average job wait times also increase with job size. Surprisingly, doubling the cluster size does *not* halve GPU stranding, but *does* halve wait times; furthermore, fluctuating job completion times has minimal effect on GPU stranding.

Bandwidth configuration: We conducted simulations on a CLOS training cluster with 4,096 GPU hosts. Figure 15 shows that training speed increases with network bandwidth. At 800Gbps fiber bandwidth, training speed closely approaches the idealized scenario of infinite bandwidth.

Fiber length: Figure 16 shows that fiber length, which is typically less than 100 meters in training clusters, has a minimal effect on training speed. This indicates that the propagation delay of data packets, determined by the fiber length, does not significantly impact training performance.

Enabled scaling strategy: Our simulations demonstrate that fiber distance has minimal impact, enabling flexible job placement. Increasing link speed from 100 Gbps to 1,000 Gbps can yield up to a 56% speedup. Larger clusters increase both utilization and training throughput, highlighting the importance for Meta to develop large clusters for training large models. Furthermore, the negligible impact of fiber length indicates that training models across geographically distributed clusters is feasible.

9.2 Does Shallow-buffer Switch Work?

Switch buffer sizing is important for microsecond-level performance. Meta 's data centers increasingly adopt disaggregated, commodity switches [16], raising questions: Are shallow-buffer switches sufficient for AI networks, or are high-radix, deep-buffer switches necessary? We explore the impact of switch buffer size at various network layers on training performance to guide our hardware selection.

Setup: Using PSM (packet-mode), we simulate a 4,096-host CLOS network (spine:aggregation:leaf ratio = 1:8:32). We consider all-to-all and all-reduce collective operations spanning 4,000 and 16,000 GPUs in GrandTeton hosts, each exchanging 256 MB per operation over 400 Gbps links. Priority Flow Control (PFC) [46] is used to manage congestion. Different buffer sizes are modified in PSM switch struct (Figure 4 (b)).

Spine switch's buffer size and its impact on all-reduce operation: Figure 17 shows minimal impact of spine-buffer size on all-reduce completion times, regardless of collective algorithm (*e.g.,* DBT [41] or RHD [48]) or NIC bandwidth (200 Gbps vs. 400 Gbps). Figure 18 confirms that PFC-induced pauses largely occur at aggregation switches, not spines.

Aggregation switch's buffer size and its impact on all-to-all operation: All-to-all typically involves fewer GPUs than all-reduce and often bypasses the spine layer. Figure 19 indicates that beyond 2 MB, additional buffering yields diminishing returns compared to increasing from 1 MB to 2 MB. Figure 20 shows pause signals at the upstream path (NIC→Leaf→Aggregation) for 400 Gbps NICs, and downstream (Aggregation→Leaf→NIC) for 200 Gbps, reflecting that NIC-leaf links are the principal bottleneck. Additionally, in Figure 21, we explore how buffer size impacts throughput during infinite-size all-to-all operations with 200Gbps NICs. We observe that during congestion, throughput drops to 175Gbps for a 1MB buffer and to 180Gbps for a 16MB buffer. Larger buffers enable the network to maintain its full speed of

200Gbps for a 6.8× longer period.

Implications to Buffer Spec:

- Different switches require varying buffer sizes. Larger buffers in leaf and aggregation switches can improve performance by up to 14.3%, while buffer size in spine switches has less significant impact.
- Bottlenecks in all-to-all and all-reduce collective operations primarily occur at the leaf switch-NIC links. This is because the link pause can be back-propagated to the source, causing delays in these operations.

9.3 How to Configure QP Scaling?

ECMP often struggles in AI networks due to limited flow entropy [13]. *Queue-pair (QP) scaling* mitigates this by splitting a single flow into multiple subflows at the host, distributing traffic more evenly across available paths. However, choosing an optimal QP-scaling factor is hard: although more subflows can improve load balancing, they also raise the risk of collisions and can prolong completion times, especially if subflows exceed the number of paths.

Setup: Using PSM, we simulate a rail-optimized topology [44] in which each host has 8 GPUs and its own leaf switch. The cluster uses ECN-based congestion control. NIC bandwidth is 200 Gbps; aggregation links run at 400 Gbps. We focus on the completion time for all-reduce jobs of various sizes, with each GPU processing the same amount of data.

Results: Figure 22 shows that for single all-reduce jobs not sharing links, a QP-scaling factor of 32 nearly achieves roofline performance, where distribution perfectly matches link capacity. In contrast, Figure 23 shows two concurrent 64-GPU jobs sharing some hosts. Even sharing just one host inflates completion time by 40% compared to the ideal, partly due to ECN's difficulty in precisely measuring congestion when flows overlap.

Impact on Routing Design:

- For collective operations lacking link-sharing, a QP-scaling factor of 32 closely matches ideal link utilization.
- Sharing links across multiple jobs complicates QP scaling. When two jobs share leaf switches, completion times can rise by 40%, underscoring the challenge of accurately estimating congestion with overlapping flows.

10 Experiences and Future Directions

In this section, we discuss our years-long journey in simulator evolution, and share insights on leveraging simulations in production and future AI network design.

10.1 Detecting Congestion in Production

Building on Arcadia's ability to efficiently and accurately simulate large-scale training clusters, we demonstrate how it can be used to improve network management. Specifically, we replay an 8-day trace of AI training jobs from a production training cluster with 160 GPU hosts in a three-level Clos network. This network has a switch ratio of 1:5:10 at the spine, aggregation, and leaf layers, respectively, with each link operating at 400 Gbps.

We employ FSM to detect congestion by identifying jobs that fail to fully utilize the bandwidth between their GPU hosts and the leaf switches. In the production cluster, congestion is signaled by pause-frame generation under Priority Flow Control (PFC). Since Arcadia simulates the entire job life cycle—including scheduling decisions—it can accurately identify when congestion arises, often triggered by the arrival of a new job that overloads the training cluster.

Figure 24 illustrates Arcadia's congestion-detection ability. In Figure 24(a), Arcadia correctly pinpoints a 3.8-day congestion period caused by a single job, while Figure 24(b) shows that it detects congestion arising from multiple concurrent training jobs. Thus, even in flow mode, Arcadia enables network managers to allocate resources more effectively, and experiment with new policies for improved utilization.

10.2 Evolution of simulators

Inaccuracy of analytical models: Analytical models were first used in AI software/hardware co-design to approximate the performance of different design. For example, assuming a non-blocking network, we can estimate the finish time of each collective using the total bytes to be exchanged, divided by the link capacity. However, such a linear model cannot capture the complex dynamics of multiple flows and jobs running simultaneously. For example, we observe that, as network load increases, even a single all-reduce can show a 16% to 5000% (50x) difference between a simplified analytical model and a network simulator.

Leveraging existing TCP simulators: Our first attempt was to leverage open source simulator NS3 for a prototype. Our goal is to understand how network congestion could affect training performance. The simulation assumes TCP as its underlying transport, which matched with our first generation of AI training environment. The results already show much higher accuracy than an analytical model, demonstrating the effectiveness of network simulation.

Augmenting with AI Networking features: To match more closely with real-world systems, we added new functions in NS3 to be able to simulate RoCE. We also added switch and NIC models to capture the dynamics of shared buffering and PAUSE/PFC based flow control. Equipped with these functions, we are able to answer questions like how different routing algorithms affect end-to-end performance.

Striking for Performance: The NS3-based simulation approach quickly hit scalability challenge when we study multiple jobs and larger AI zones. When scaling to 2,048 GPUs, the simulation lasts hours and stresses the memory. With large language models gaining more interest, scaling to 32K GPU has become a requirement. We tried to optimize the NS3 code significantly, resulting in PSM, but the performance was still unsatisfactory. This has resulted in the development of Arcadia from scratch.

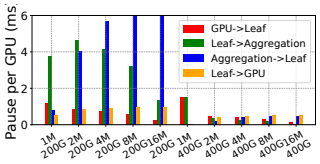


Figure 20: Port pause time per GPU at different switches for all-to-all

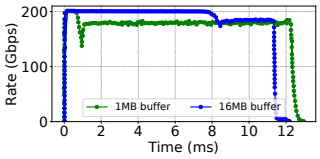


Figure 21: Transmission rates of different buffer sizes with time

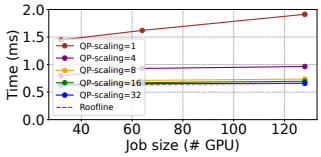


Figure 22: Job size vs Job completion time for different QP-scaling factors

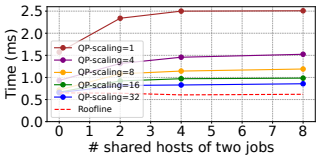


Figure 23: Number of shared hosts vs Job completion time for different QP-scaling factors

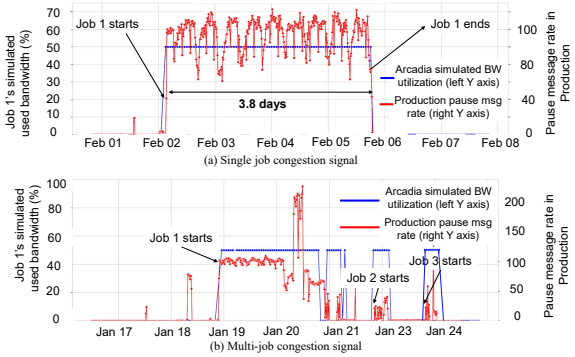


Figure 24: Congestion signals detected by Arcadia

Comparison with Emulation: Today, Arcadia is used by multiple teams for what-ifs at different layers of the stack. An alternative is to evaluate using lab testbed through emulation. For example, while Arcadia generates results for in-network compute, we can also test its gain with real hardware on a smaller topology. Cross-validation is needed between the two systems. In the future, we can also incorporate statistical models derived from testbed measurements.

10.3 Future work in exploring AI design space

We will continue expanding the scope of Arcadia by increasing its simulation granularity and integrating with other simulators as well as models.

Training ASIC + Network Co-Simulation: In one example, we can refine the simulation of the compute portion. Currently, it uses an analytical model, but this could be replaced with a more accurate ASIC model. For instance, a behavioral model (BModel), commonly used in the ASIC design, models chip blocks' functionality at a higher level rather than being cycle accurate. This model can enable SoC architecture design exploration, workload power/performance analysis, and designing performant kernels before hardware availability. We can build a co-simulation environment using Arcadia to study network effects while allowing BModel to capture computation and memory effects. Integrating these simulators helps understand the performance impacts of coordinating multiple new training ASICs in the same job, revealing potential network bottlenecks.

Evaluating New Transport for AI Accelerators: There have been initiatives exploring new transport mechanisms optimized for training, including selective retransmission [53], packet spray [20], and receiver-based control [50]. In such efforts, it is essential to integrate the modeling of endpoint performance with the characteristics of the transport. These transport can be added to PSM. By understanding how changes in architecture influence transport performance, we can make data-driven decisions early in the development process.

Cross-cluster AI Training: Cross-cluster (*i.e.,* inter-datacenter) training is an increasingly important operating point for large-scale AI systems. As model sizes and training throughput requirements grow, single-site clusters can become constrained by power and thermal envelopes, maintenance windows, and localized resource contention, making it difficult to provision sufficient capacity with predictable turnaround time. Spanning training across multiple clusters can provide elasticity (borrowing capacity from other sites), improved robustness to site-level disruptions, and new cost/performance trade-offs, but it also introduces WAN-dominated communication on critical paths and additional scheduling/placement complexity. Arcadia can help analyze these trade-offs by simulating cross-cluster placements and parallelism strategies (DP/TP/PP) under a configured inter-cluster connectivity model (WAN RTT/bandwidth and transport behavior), allowing users to quantify how different site assignments and scheduling policies affect iteration time, utilization, and sensitivity to contention or failures.

11 Conclusion

In this paper, we introduce Arcadia, a scalable and extensible simulator that provides cluster-level insights into AI training job performance[4]. Arcadia executes efficiently and continuously aligns its simulation results with production data, enabling both in-depth evaluations of individual jobs and holistic cluster management. It supports network planning, design, operation, and performance tuning for AI training clusters and is widely used in Meta production.

Acknowledgments

We thank Gilad Goldfarb and Ashay Narsale for their collaboration within Meta's Infrastructure team, and Vipul Deokar and Omar Baldonado for their leadership and support. We are grateful to our shepherd, Matt Welsh, for his guidance and feedback. We also thank the anonymous reviewers for their constructive comments.

References

[1] Deepspeed. https://github.com/microsoft/DeepSpeed.

[2] Introducing llama 2. https://ai.meta.com/llama/.

[4]This work does not raise any ethical issues.

[3] Megatron-lm. https://github.com/NVIDIA/Megatron-LM.

[4] Mpi for distributed simulation. https://www.nsnam.org/docs/models/html/distributed.html.

[5] Mtia v1: Meta's first-generation ai inference accelerator. https://ai.meta.com/blog/meta-training-inference-accelerator-AI-MTIA/.

[6] Nccl. https://developer.nvidia.com/nccl.

[7] Not all deep buffer switches are created equal. https://blogs.juniper.net/en-us/enterprise-cloud-and-transformation/not-all-deep-buffer-switches-are-created-equal.

[8] Nvidia scalable hierarchical aggregation and reduction protocol (sharp). https://docs.nvidia.com/networking/display/sharpv300.

[9] Pytorch. https://pytorch.org/.

[10] Roce networks for distributed ai training at scale. https://engineering.fb.com/2024/08/05/data-center-engineering/roce-network-distributed-ai-training-at-scale/.

[11] Simpy. https://simpy.readthedocs.io/en/latest/.

[12] Saksham Agarwal, Rachit Agarwal, Behnam Montazeri, Masoud Moshref, Khaled Elmeleegy, Luigi Rizzo, Marc Asher de Kruijf, Gautam Kumar, Sylvia Ratnasamy, David Culler, et al. Understanding host interconnect congestion. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 198–204, 2022.

[13] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[14] AMD. Radeon gpu profiler. https://gpuopen.com/rgp/, 2023.

[15] 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. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 115–131, 2024.

[16] Sean Choi, Boris Burkov, Alex Eckert, Tian Fang, Saman Kazemkhani, Rob Sherwood, Ying Zhang, and Hongyi Zeng. Fboss: Building switch software at scale. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 342–356, New York, NY, USA, 2018. Association for Computing Machinery.

[17] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, et al. Mast: Global scheduling of ml training across geo-distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 563–580, 2024.

[18] Cisco. Cisco ip telephony flash cards: Weighted random early detection (wred). https://www.ciscopress.com/articles/article.asp?p=352991&seqNum=8, 2004.

[19] Cisco. Defining qos queues. https://www.cisco.com/assets/sol/sb/Switches_Emulators_v2_2_015/help/nk_configuring_quality_service16.html, 2023.

[20] Advait Dixit, Pawan Prakash, Y. Hu, and Ramana Kompella. On the impact of packet spraying in data center networks. pages 2130–2138, 04 2013.

[21] Richard M Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[22] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 57–70, 2024.

[23] 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.

[24] X Yu Geoffrey, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A {Runtime-Based} computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 503–521, 2021.

[25] GitHub. Why nccl-tests performs differently with different nccl versions? https://github.com/NVIDIA/nccl/issues/837, 2023.

[26] Ronald L Graham. Bounds on multiprocessing anomalies and related packing algorithms. In *Proceedings of the May 16-18, 1972, spring joint computer conference*, pages 205–217, 1971.

[27] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[28] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. Loggopsim: simulating large-scale applications in the loggops model. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 597–604, 2010.

[29] Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, et al. Xdl: an industrial deep learning framework for high-dimensional sparse data. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, pages 1–9, 2019.

[30] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed {DNN} training in heterogeneous {GPU/CPU} clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

[31] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[32] Cameron Kiddle, Rob Simmonds, Carey Williamson, and Brian Unger. Hybrid packet/fluid flow network simulation. In *Seventeenth Workshop on Parallel and Distributed Simulation, 2003.(PADS 2003). Proceedings.*, pages 143–152. IEEE, 2003.

[33] Hong Liu, Ryohei Urata, Kevin Yasumura, Xiang Zhou, Roy Bannon, Jill Berger, Pedram Dashti, Norm Jouppi, Cedric Lam, Sheng Li, et al. Lightwave fabrics: at-scale optical circuit switching for datacenter and machine learning systems. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 499–515, 2023.

[34] Jason Liu. Packet-level integration of fluid tcp models in real-time network simulation. In *Proceedings of the 2006 Winter Simulation Conference*, pages 2162–2169. IEEE, 2006.

[35] Levente Mészáros, Andras Varga, and Michael Kirsche. Inet framework. *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, pages 55–106, 2019.

[36] Meta. Ocp summit 2022: Open hardware for ai infrastructure. https://engineering.fb.com/2022/10/18/open-source/ocp-summit-2022-grand-teton/, 2022.

[37] Meta. Our next-generation meta training and inference accelerator. https://ai.meta.com/blog/next-generation-meta-training-inference-accelerator-AI-MTIA/, 2024.

[38] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie (Amy) Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, KR Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 993–1011, New York, NY, USA, 2022. Association for Computing Machinery.

[39] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, et al. High-performance, distributed training of large-scale deep learning recommendation models. *arXiv preprint arXiv:2104.05158*, 2021.

[40] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[41] NVIDIA. Massively scale your deep learning training with nccl 2.4. https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/, 2019.

[42] NVIDIA. Nvidia. dgx a100 system user guide. https://docs.nvidia.com/dgx/pdf/dgxa100-user-guide.pdf, 2021.

[43] NVIDIA. Nvidia. dgx h100 system user guide. https://www.nvidia.com/en-us/data-center/dgx-h100/, 2023.

[44] NVIDIA. Nvidia dgx superpod: Next generation scalable infrastructure for ai leadership, reference architecture. https://docs.nvidia.com/https:/docs.nvidia.com/dgx-superpod-reference-architecture-dgx-h100.pdf, 2023.

[45] NVIDIA. Nvidia gpudirect. https://developer.nvidia.com/gpudirect, 2023.

[46] NVIDIA. Priority flow control (pfc). https://docs.nvidia.com/networking/display/onyxv3104206/priority+flow+control+(pfc), 2023.

[47] NVIDIA. Rdma over converged ethernet (roce). https://docs.nvidia.com/networking/display/ofedv502180/rdma+over+converged+ethernet+(roce), 2023.

[48] Rolf Rabenseifner. Optimization of collective reduction operations. In *Computational Science-ICCS 2004: 4th International Conference, Kraków, Poland, June 6-9, 2004, Proceedings, Part I 4*, pages 1–9. Springer, 2004.

[49] Yusuke Sakumoto, Ryouta Asai, Hiroyuki Ohsaki, and Makoto Imase. Design and implementation of flow-level simulator for performance evaluation of large scale networks. In *2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 166–172. IEEE, 2007.

[50] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. Tcp congestion control with a misbehaving receiver. *SIGCOMM Comput. Commun. Rev.*, 29(5):71–78, oct 1999.

[51] Benjamin Sliwa and Christian Wietfeld. Data-driven network simulation for performance analysis of anticipatory vehicular communication systems. *IEEE Access*, 7:172638–172653, 2019.

[52] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[53] Vijay Vasudevan, Amar Phanishayee, Hiral Shah, Elie Krevat, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, and Brian Mueller. Safe and effective fine-grained tcp retransmissions for datacenter communication. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 303–314, New York, NY, USA, 2009. Association for Computing Machinery.

[54] Pedro Velho, Lucas Mello Schnorr, Henri Casanova, and Arnaud Legrand. On the validity of flow-level tcp network models for grid and cloud simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 23(4):1–26, 2013.

[55] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. Simai: Unifying architecture design and performance tunning for large-scale large language model training with scalability and precision. https://github.com/aliyun/SimAI, 2025.

[56] 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.

[57] Yiting Xia, Ying Zhang, Zhizhen Zhong, Guanqing Yan, Chiun Lin Lim, Satyajeet Singh Ahuja, Soshant Bali, Alexander Nikolaidis, Kimia Ghobadi, and Manya Ghobadi. A social network under social distancing: Risk-Driven backbone management during COVID-19 and beyond. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 217–231, 2021.

[58] Qiongwen Xu, Sebastiano Miano, Xiangyu Gao, Tao Wang, Adithya Murugadass, Songyuan Zhang, Anirudh Sivaraman, Gianni Antichi, and Srinivas Narayana. State-compute replication: Parallelizing high-speed stateful packet processing. *arXiv preprint arXiv:2309.14647*, 2023.

[59] Qiongwen Xu, Michael D Wong, Tanvi Wagle, Srinivas Narayana, and Anirudh Sivaraman. Synthesizing safe and efficient kernel extensions for packet processing. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 50–64, 2021.

[60] Buyun Zhang, Liang Luo, Xi Liu, Jay Li, Zeliang Chen, Weilin Zhang, Xiaohan Wei, Yuchen Hao, Michael Tsang, Wenjun Wang, et al. Dhen: A deep and hierarchical ensemble network for large-scale click-through rate prediction. *arXiv preprint arXiv:2203.11014*, 2022.

[61] Qizhen Zhang, Kelvin KW 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*, pages 287–304, 2021.

[62] Hongyu Zhu, Amar Phanishayee, and Gennady Pekhimenko. Daydream: Accurately estimating the efficacy of optimizations for {DNN} training. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 337–352, 2020.

[63] 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. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.

A Setup in the trace-driven simulation

For each job i, we note the time of completing its first 100,000 iterations as its training time t^i_{old}. We group jobs by both their training sample sizes and job sizes, and for each group, we define its *new* training time t^i_{new} as the minimum t^j_{old} in that group. We then replay the same arrival patterns using a FIFO scheduler to allocate jobs on a three-level CLOS network with 4,096 GPUs (see Figure 27 for the topology). We calculate the performance gain as the ratio of the timespan for completing all training jobs using the new training time to that using the old training time. This approach simulates applying an optimization (such as optical switches [33] that dynamically adjusts connectivity to reduce latency) uniformly to all jobs within the same group.

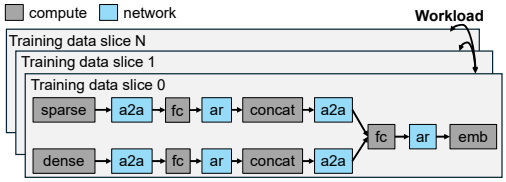


Figure 25: The forward propagation workload of a variant of recommendation model [40] in one iteration with data parallelism (DP) optimization. (Here, sparse, dense, fc, concat, emb represent the calculations of sparse, dense features, fully connected layer, dot product concatenation, and embedding respectively; a2a,ar are for all-to-all and all-reduce operations.)

B Flow rate update

Algorithm 1 Flow rate update

```
while True do
    for each flow f do
        Continue if a link of this flow reaches its capacity
        for each link l in the flow do
            Calculate the rate weight w^l_f by network setting
        end for
        /* Δ is the minimal rate unit to add */
        Add min{w^l_f | l ∈ f}Δ rate to the flow
        Update the status of the links of the flow
    end for
    Exit if no flow is updated in this loop
end while
```

Algorithm 1 describes how flow rates are updated. Each link gradually increases the traffic for flows sharing that link, following the user's specified bandwidth-allocation algorithm, until reaching full utilization. A flow will not increase its rate on any given link if another link in the flow's path is already at capacity, as the flow's overall rate is determined by its slowest link.

C Evaluation

C.1 Regularization coefficient sensitivity

We study the sensitivity to the regularization coefficient λ in §7.2. We first normalize per-iteration completion times to $[0,1]$ to make λ comparable across workloads. We then sweep $\lambda \in \{0, 0.1\sigma, 0.3\sigma, 0.5\sigma, 1\sigma, 2\sigma, 5\sigma\}$, where σ

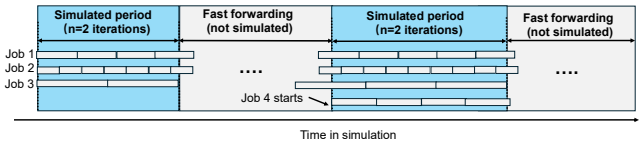


Figure 26: An example of fast forwarding for acceleration: All jobs need to concurrently run at least $n = 2$ iterations.

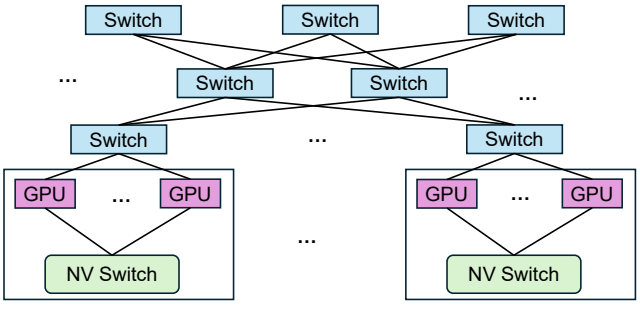


Figure 27: Simulated training cluster topology

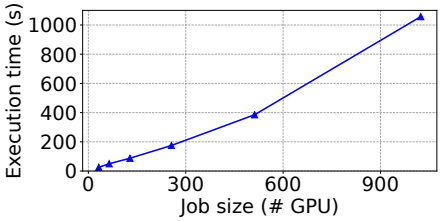


Figure 28: Job size vs Execution time for DLRM

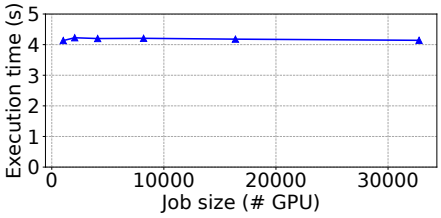


Figure 29: Job size vs Execution time for GenAI

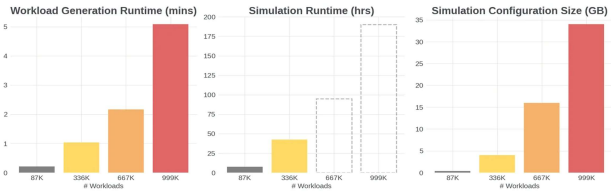


Figure 30: Simulation runtime vs Workload Size

is the standard deviation of the residuals under the unregularized least-squares solution: $r_i = T^i(\hat{\Phi}_{\text{LS}}) - T^i_{\text{prod}}$, and $\hat{\Phi}_{\text{LS}} = \arg\min_\Phi \sum_i (T^i(\Phi) - T^i_{\text{prod}})^2$ (*i.e.,* $\lambda = 0$). We find that MAPE remains essentially unchanged as λ increases up to approximately 1.2σ, and starts to increase (not shown). Because larger λ more strongly penalizes rank-order violations, we select the largest λ that does not noticeably increase MAPE, balancing numerical accuracy with order consistency.

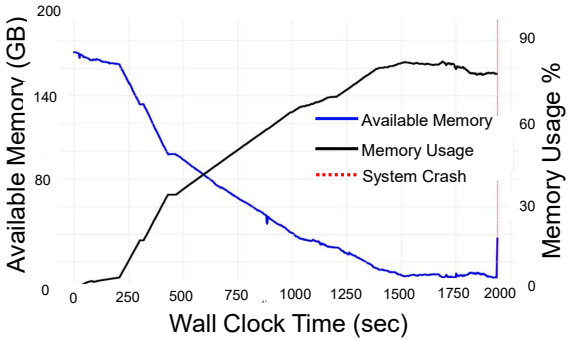


Figure 31: Memory usage with time until OOM

C.2 Impact of models

Figures 28 and 29 show Arcadia's execution time on FSM for simulating DLRM [40] and GenAI [2] model training across various GPU numbers. While Arcadia's execution time for DLRM models linearly increases with GPU count, it remains stable for GenAI models. This difference is because network flows for DLRM models increase quadratically with GPU numbers, whereas for GenAI models, the increase is linear[5]. Fortunately, Arcadia's effective multithreading handles these growing flows, ensuring scalability for larger models.

C.3 Scalability

We find that the primary scalability bottleneck when simulating larger training jobs is memory consumption, which grows with the number of workload nodes in the per-iteration DAG. To quantify the memory limit, we progressively increase the number of workload nodes and run the simulator until it encounters an out-of-memory (OOM) failure. We observe that the simulator begins to crash at ~ 1.3 million workload nodes. Figure 31 reports the wall-clock time and memory footprint during this stress test: we incrementally feed workload nodes into the simulator until reaching 1.3 million, at which point the process terminates due to OOM. We also measure simulation runtime as a function of DAG size and find it scales approximately linearly with the number of workload nodes (*i.e.,* the size of the simulation input), as shown in Figure 30.

[5] DLRM involves numerous all-to-all collective operations, unlike GenAI.