Towards Easy and Realistic Network Infrastructure Testing for Large-scale Machine Learning

Jinsun Yoo[1], ChonLam Lao[2], Lianjie Cao[3], Bob Lantz[3], Minlan Yu[2], Tushar Krishna[1], Puneet Sharma[3]

[1]Georgia Institute of Technology, [2]Harvard University, [3]Hewlett Packard Labs

Abstract

This paper lays the foundation for GENIE, a testing framework that captures the impact of real hardware network behavior on ML workload performance, without requiring expensive GPUs. GENIE uses CPU-initiated traffic over a hardware testbed to emulate GPU to GPU communication, and adapts the ASTRA-sim simulator to model interaction between the network and the ML workload.

1 Introduction

The growth in both model size and training data has pushed ML training clusters to scale beyond tens of thousands of GPUs [9, 13, 19, 28]. Distributed training involves communication where GPUs periodically share the results of partial computation (such as activations and gradients). Communication often becomes a bottleneck for ML training, diminishing the returns from scaling compute. For example, Mixture of Experts (MoE), an increasingly popular technique of Large Language Models, involves all-to-all communication in the critical path [6, 15, 17].

A large GPU cluster relies on an equally complex network infrastructure that spans across multiple NICs, switches, and links. Each node is connected to the network via multiple high-bandwidth NICs that enable low-latency, high-bandwidth communication between GPUs across nodes. Several layers of network switches connect tens of thousands of such nodes, while links provide the physical connection between switches or switches and NICs. Placing and configuring this set of hardware is a non-trivial task. Correct configuration across NICs and switches are necessary to enable load-balancing schemes like multipathing or spraying, or traffic engineering optimizations that are tailored to the unique characteristics of ML training traffic. Furthermore, the large number of hardware components exposes the network infrastructure to failures or degraded performance, adding to the management and debugging challenges [2, 10, 16, 22, 25, 31].

This complexity motivates a number of use cases for a framework to test the *real network infrastructure*. First, we want to understand how different network configurations (from HW buffer size to congestion control algorithms) or events (such as failures) influence the ML workload [9, 13, 26, 27, 33]. Specifically, we want to understand how these effects propagate through a workload and impact overall performance. Second, there is vendor demand to validate the network infrastructure before running ML training workloads. Industry operators report that network hardware failures or misconfigurations account for a significant portion of costly failures and restarts [9, 13].

While prior work has modeled network behavior with simulators, simulators alone are not enough to fully satisfy the above use cases. First, publicly available simulators may not accurately model novel proprietary networks such as HPE Slingshot [4]. Additionally, simulators cannot validate if the hardware network is configured properly to deliver the desired performance for the ML workload. Finally, simulators cannot easily model unexpected network anomalies that occur in real deployments. Figure 1 depicts how the performance of an AllReduce collective suffers as an unpredicted NIC degradation occurs. We repeatedly issue AllReduce collectives across 16 nodes equipped with A100 GPUs and Connectx-6 NICs and plot the performance. The simulator does not reflect the network anomaly that it did not predict.

Unfortunately, to execute an ML workload on a real network infrastructure one must secure an equally large number of GPUs. This is increasingly difficult due to GPU scarcity and cost. We draw attention to the fact that our focus is on the network traffic behavior on the network hardware, not on how computation is done on the GPU. To the best of our knowledge, there is no framework that can execute ML workloads and generate real network traffic without GPUs.

In this paper, we share our vision for GENIE (GPUs Eliminated for Network Infrastructure Examination). GENIE provides realistic ML workload performance testing of network infrastructure without requiring expensive GPUs. The key idea is to replicate GPU behavior and generate network traffic from the CPU, while capturing the workload behavior through the ASTRA-sim simulator [36].

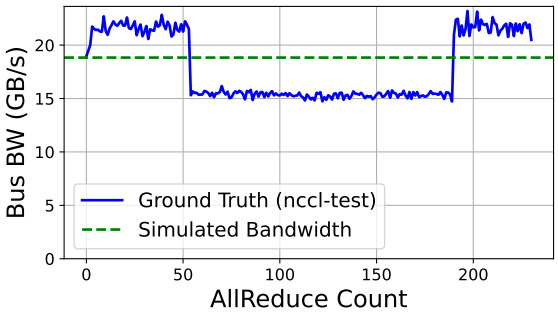

Figure 1: Bandwidth for several AllReduce runs against simulation. We inject an anomaly which the simulation does not anticipate.

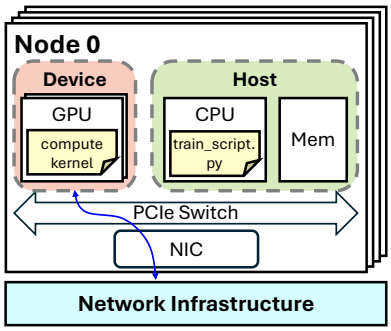

Figure 2: A typical deployment of training processes across multiple nodes. The blue arrow depicts the communication between nodes.

2 Background

2.1 ML Training Process Behavior

Developers write complex ML training processes with frameworks such as PyTorch [1] or JAX [8]. These processes interact with various hardware components and run across multiple nodes. Fig. 2 depicts a typical deployment, where a large number of nodes are connected through a multi-layered network fabric. Within each node, the training process runs on the host CPU, handling operations such as launching compute kernels on accelerator devices (commonly GPUs) and streaming data from host memory to device memory as input to compute kernels. Training nodes must periodically share data stored in memory such as weights, activations, and gradients through collective communication. While the training process initiates the communication, the data bypasses the host OS and is passed into the network directly through transports such as RDMA.

The dependency of these operations is determined by factors such as the model definition and the parallelization strategies. The training process executes each operation whose upstream operation has completed and the dependency has been resolved. If an operation lies on the critical path, delays can degrade the performance of the overall workload.

2.2 Communication in ML Training

As ML models grow explosively, distributed training increasingly relies on various parallelism schemes to leverage distributed computational resources (e.g., expert parallelism and pipeline parallelism), leading to a high and complex communication overhead. While this overhead amplifies the need for specialized network designs, the complexity of the network makes management, debugging, and identifying best practices more challenging.

Building and managing a network infrastructure for ML traffic has become increasingly complex. For example, ML traffic requires high-bandwidth, low-latency communication across GPUs, making GPUDirect RDMA and low-level transports (e.g., congestion control) essential. Additionally, ML traffic exhibits unique workload characteristics, such as low-entropy patterns, necessitating load-balancing schemes like multipathing or spraying at the hardware NIC and switches. Furthermore, topology and traffic engineering optimizations tailored to ML workloads add to network complexity. These increasing network demands make management, debugging, and identifying best practices more challenging.

2.3 Representing ML Workload

To study workload behavior without rerunning it on all of the hardware, a suitable representation format is essential. Chakra [32], supported by the MLCommons initiative [18], is a widely used graph based representation of AI/ML workloads. It captures the execution of distributed workloads into a graph, where vertices denote operators and edges denote their dependencies.

Several tools take a Chakra graph as input, allowing us to easily study a wide array of *arbitrary* workloads. One of such tools is the ASTRA-sim distributed ML simulator [36]. ASTRA-sim is an event-based simulator. It uses a modular design where users choose between different compute, memory, or communication models. For example, ASTRA-sim supports a network backend based on the ns-3 simulator, which allows it to simulate RDMA traffic at packet level using ns-3 features [14].

3 Motivation

In this section, we describe the motivation behind `GENIE` in detail, and discuss how existing work falls short of our goal.

3.1 Studying the Impact of Network on AI/ML Workload

When building the network for AI/ML workloads, it is crucial to understand how the network interacts with the workload. Researchers seek to understand important questions such as *What is the best set of network configuration or hardware topology I can use for a given workload?* or, *I developed a new congestion control mechanism. How can I showcase its*

efficiency for AI/ML workloads?. Answering these questions helps improve system design or allows us to more effectively showcase and validate new ideas.

A simple way to evaluate the above questions is to prepare a training cluster and run the training workload on it. However, this does not scale very well due to the expensive cost of GPUs. As a result, academic evaluations have been limited to a small set of clusters. Even in large industry participants, non-production research projects are limited to small evaluations.

An alternative direction would be to use a simulator. For example, some works use the ns-3 simulator to simulate RDMA messages [14, 34]. However, simulators can only model publicly available protocols such as RDMA over Converged Ethernet (RoCE) [12]. They are limited in modeling commonly used proprietary technology, such as Infiniband, Slingshot, or Spectrum-X [4, 11, 21]. Another inherent limitation of simulators is their inaccuracy compared to real deployed hardware and software.

One deficiency GENIE aspires to solve is the ability to capture the relation between network events and the whole workload, not just a single collective. Simply testing how collectives perform on different configurations is not enough. The unique characteristics of ML traffic such as bursty behavior means that single collective benchmarks may not accurately reflect the statistic pattern of whole workload performance.

For this reason, benchmarking tools like nccl-tests [20], which only measure a single collective operation across different network configurations, are not suitable for our goals (not to mention that they require GPUs). Gloo-benchmark [7] can run on CPUs without GPUs, but it is also limited to a single collective and is not a suitable option for studying the network-workload relationship.

When running a workload with a network testing framework, we want to capture the interaction between the workload and network components. Some benchmarking tools in other domains trace workloads and replay operations at fixed times based on the original issued timestamps [5]. However, operations within ML workloads have dependencies between each other, and simply replaying them at fixed timestamps fails to capture the true workload behavior.

3.2 Easily Verifying Network Integrity

Some hardware failure or software misconfigurations could be diagnosed before starting a training job, saving precious GPU hours. For example, Meta reports their experience of diagnosing bad performance in production training jobs, which was tracked back to a mismatch in the packet scheduling algorithms loaded on different switches [9]. In the end, they had to involve the network vendor to fix the issue. Similarly, while developing GENIE, we found that congestion control was disabled in our network switch, which led to backpressure and packet drops. Both of these examples could have been identified before training began.

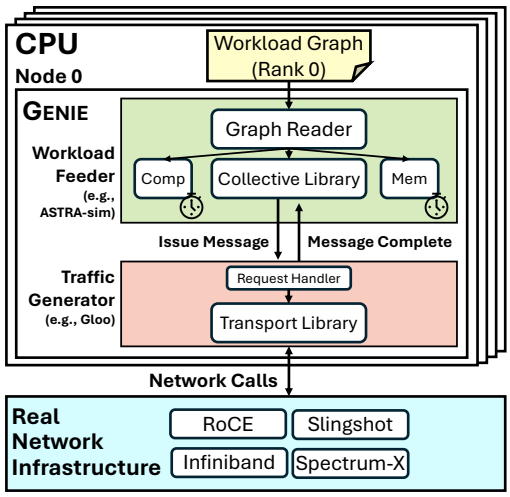

Figure 3: Overall Architecture of GENIE. GENIE can work on a cluster with only CPU nodes connected via the network infrastructure.

However, diagnosing these issues are difficult. Inspecting every hardware/software component is tedious in a largescale deployment. On the other hand, it is expensive for network vendors to purchase large numbers of GPUs for the sole purpose of testing their network infrastructure. Simulators can help (to some extent) predict how failures affect workload performance. However, they cannot detect failures themselves. Altogether, this motivates the case for a network testing framework that does not rely on costly GPUs.

4 GENIE Design

Figure 3 is an overview of our design of GENIE. GENIE has two key components: the **traffic generator** generates network traffic from CPU[1], while the **workload feeder** captures the interaction between the workload and the real network.

Modeling Workload with ASTRA-sim: GENIE uses the ASTRA-sim simulator to capture the workload behavior of real training processes. By design, each instance of GENIE replicates the behavior of the actual training process. The default ASTRA-sim is a sequential process that simulates a single event based timeline. We modify the simulator to run in a distributed setting. GENIE instances are duplicated across CPU nodes, where each instance represents a training rank. GENIE uses ASTRA-sim's graph reader to traverse through the workload graph provided as input, which encodes operators and their dependencies. The graph reader issues each operator as discussed below. Once an operator has finished, the graph reader then issues the operators whose dependencies are resolved.

ASTRA-sim models non-communication operators locally and `sleep`s for the simulated duration as opposed to launching GPU kernels. For collectives, its internal collective library breaks down a collective into send and receive messages. The

[1] Intra-node communication is beyond the scope of this work.

collective library triggers the traffic generator to issue the messages, and the traffic generator reports once the message finishes. If the collective completes, the graph reader moves on to the next operation.

ASTRA-sim's collective library allows users to test common collective algorithms such as Ring or Tree. Alternatively, it also allows users to use arbitrary collective algorithms generated by synthesizers or Domain Specific Languages [3, 30, 35, 37].

Note how we are not actually training a model with real inputs and weights. Instead, we simply recreate the local delay a real training process would see for compute or memory operations. While we do generate real RDMA traffic with the same message size as in the training process, the payload contains meaningless data.

The distributed instances of GENIE do not use a special synchronization mechanism. Instead, they communicate through collective communication, just as training processes do on a GPU cluster. Each instance emulates compute and memory operations locally. They communicate over the network fabric only when a collective operation occurs in the original workload.

GENIE captures the relationship between network behavior and the workload performance as follows: An optimal network leads to reduced time in the collective communication (and vice versa for a suboptimal network configuration). If there are dependent operations, the graph reader stops traversing the workload graph and waits for the collective dependency completes and the dependency is resolved. In the end, any delay along the critical path slows down the workload process. As a result, GENIE can capture how network issues impact workload performance.

Creating GPU Communication with Traffic Generator: We implement a traffic generator as the network backend for ASTRA-sim. A key requirement of GENIE is to generate GPU traffic with CPUs. The traffic generator acts as an interface between ASTRA-sim and the real network. It exposes endpoints for point-to-point send and receive messages. Once ASTRA-sim's collective library breaks down a collective into send and receive messages, it calls the traffic generator through these endpoints. The traffic generator then creates and injects traffic into the network.

Once ASTRA-sim triggers the traffic generator, the request handler calls the transport library to perform tasks such as assigning memory buffers or issuing network calls. To generate RDMA traffic, for example, the traffic generator can directly call libraries such as libibverbs [29] or libfabric [23]. Alternatively, it can also use high level libraries that encapsulate the low level libraries. Examples include the point-to-point messages functions of Gloo [7] or perftest [24]. Both tools can generate arbitrary point-to-point messages only with CPUs in an application agonistic manner. Choosing the right implementation to balance fidelity and flexibility is left to future work.

While we discuss generating RDMA traffic as an example, GENIE is designed to be easily extensible. GENIE is envisioned to work on arbitrary network infrastructure supporting various transports. The traffic generator should switch between different software stack or network drivers depending on the network fabric it is deployed on.

Network Infrastructure The direct interaction with real network is what sets GENIE apart from simulators. Once the traffic generator injects network traffic into the network through the NIC, the traffic traverses through the underlying network fabric to the destination. The switches in the network fabric do not differentiate traffic generated with GENIE from traffic generated from training processes on GPU clusters.

Injecting traffic directly into the real network fabric allows GENIE to realistically and accurately test network hardware and validate different configurations. GENIE is designed to be modular and portable in nature. This will allow us to deploy and test GENIE on a wide array of production networks, such as RoCE, Slingshot, Infiniband, and Spectrum-X.

5 Conclusion and Future Work

In this paper we motivated GENIE, a framework to test real network infrastructure for large-scale ML workloads without requiring costly GPUs. Several efforts remain to fully realize the vision of GENIE. First we will build GENIE and carefully study and validate the fidelity of the GPU emulation. We will then run evaluations across a wide range of network configurations, workloads, and failure scenarios showcasing GENIE's potential.

References

[1] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 929–947,

New York, NY, USA, 2024. Association for Computing Machinery.

[2] Ultra Ethernet Consortium. UEC White Paper. https://ultraethernet.org/wp-content/uploads/sites/20/2023/10/23.07.12-UEC-1.0-Overview-FINAL-WITH-LOGO.pdf, 2023.

[3] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. MSCCLang: Microsoft collective communication language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 502–514, 2023.

[4] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2020.

[5] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Trans. Storage*, 17(4), October 2021.

[6] DeepSeek-AI et al. DeepSeek-V3 Technical Report, 2025.

[7] Facebook. Gloo. https://github.com/facebookincubator/gloo.

[8] Roy Frostig, Matthew Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of Systems for Machine Learning (SysML 18)*, 2018.

[9] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024.

[10] Google. Google Falcon. https://cloud.google.com/blog/topics/systems/introducing-falcon-a-reliable-low-latency-hardware-transport, 2023.

[11] Infiniband Trade Association. Infiniband Trade Association. https://www.infinibandta.org/.

[12] Infiniband Trade Association. RDMA over Converged Ethernet. https://www.roceinitiative.org/.

[13] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024.

[14] Tarannum Khan, Saeed Rashidi, Srinivas Sridharan, Pallavi Shurpali, Aditya Akella, and Tushar Krishna. Impact of RoCE Congestion Control Policies on Distributed Training of DNNs. In *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 39–48, 2022.

[15] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, Boston, MA, jul 2023. USENIX Association.

[16] Qiang Li, Yixiao Gao, Xiaoliang Wang, Haonan Qiu, Yanfang Le, Derui Liu, Qiao Xiang, Fei Feng, Peng Zhang, Bo Li, Jianbo Dong, Lingbo Tang, Hongqiang Harry Liu, Shaozong Liu, Weijie Li, Rui Miao, Yaohui Wu, Zhiwu Wu, Chao Han, Lei Yan, Zheng Cao, Zhongjie Wu, Chen Tian, Guihai Chen, Dennis Cai, Jinbo Wu, Jiaji Zhu, Jiesheng Wu, and Jiwu Shu. Flor: An Open High Performance RDMA Framework Over Heterogeneous RNICs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 931–948, Boston, MA, jul 2023. USENIX Association.

[17] Juncai Liu, Jessie Hui Wang, and Yimin Jiang. Janus: A Unified Distributed Training Framework for Sparse Mixture-of-Experts Models. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 486–498, New York, NY, USA, 2023. Association for Computing Machinery.

[18] MLCommons. Chakra Working Group at MLCommons. https://mlcommons.org/working-groups/research/chakra.

[19] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing,*

Networking, Storage and Analysis, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[20] NVIDIA. NCCL Tests. https://github.com/NVIDIA/nccl-tests/tree/master.

[21] NVIDIA. NVIDIA Spectrum-X Network Platform Architecture. https://resources.nvidia.com/en-us-accelerated-networking-resource-library/nvidia-spectrum-x.

[22] NVIDIA. NVIDIA Infiniband Adaptive Routing Technology—Accelerating HPC and AI Applications. White Paper. https://resources.nvidia.com/en-us-cloud-native-supercomputing-dpus-campaign/infiniband-white-paper-adaptive-routing, 2023.

[23] OpenFabrics Allance. libfabric. https://ofiwg.github.io/libfabric/.

[24] OpenFabrics Allance. Perftest. github.com/linux-rdma/perftest.

[25] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking . In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 66–85, New York, NY, USA, 2022. Association for Computing Machinery.

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

[27] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. CASSINI: Network-Aware job scheduling in machine learning clusters. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1403–1420, Santa Clara, CA, April 2024. USENIX Association.

[28] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.

[29] rdma-core Maintainers. libibverbs. https://github.com/linux-rdma/rdma-core/tree/master/libibverbs.

[30] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. TACCL: Guiding collective algorithm synthesis using communication sketches. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 593–612, 2023.

[31] Leah Shalev, Hani Ayoub, Nafea Bshara, and Erez Sabbag. A Cloud-Optimized Transport Protocol for Elastic and Scalable HPC. *IEEE Micro*, 40(6):67–73, 2020.

[32] Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, and Tushar Krishna. Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces. In *arXiv:2305.14516 [cs.LG]*, 2023.

[33] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 739–767, Boston, MA, April 2023. USENIX Association.

[34] 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. In *NSDI2025*, 2025.

[35] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Swati Gupta, and Tushar Krishna. TACOS: Topology-aware collective algorithm synthesizer for distributed machine learning. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 856–870, 2024.

[36] 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 *ISPASS 2023*, 2023.

[37] Jinsun Yoo, William Won, Meghan Cowan, Nan Jiang, Benjamin Klenk, Srinivas Sridharan, and Tushar Krishna. Towards a Standardized Representation for Deep Learning Collective Algorithms. *IEEE Micro*, 45(2):1–9, 2025.