# Spine-Free Networks for Large Language Model Training

Weiyang Wang and Manya Ghobadi, *Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, 02139, USA*

***We study the optimal parallelization strategy of large language models (LLMs) and demonstrate that LLM training workloads generate sparse communication patterns in the network. Consequently, we argue that LLM training clusters do not require any-to-any full-bisection networks. We then propose Rail-only, a novel datacenter network architecture tailored to LLMs' unique communication patterns. Rail-only networks eliminate the spine layer of conventional fabrics, resulting in lower cost and energy consumption. We demonstrate that Rail-only networks achieve the same training performance while reducing network cost by 38% to 77% and network power consumption by 37% to 75%, compared to traditional GPU clusters with full-bisection bandwidth.***

Large language models (LLMs) are taking the world by storm. The recent Llama3 model was trained on 16,384 GPUs for 54 days.[1] Today, service providers are building LLM training clusters with thousands of GPUs to train and fine-tune LLMs. State-of-the-art GPU clusters utilize a "Rail-optimized" network architecture derived from the classical Clos networks[2] to provide any-to-any connectivity to *all GPUs*.

*TODAY, SERVICE PROVIDERS ARE BUILDING LLM TRAINING CLUSTERS WITH THOUSANDS OF GPUs TO TRAIN AND FINE-TUNE LLMs.*

However, scaling a Rail-optimized network to thousands of GPUs introduces insurmountable challenges, such as deadlocks and priority flow control storms.[3] Furthermore, we calculate state-of-the-art large-scale networks' cost and power scaling (see the "Network Cost and Power Analysis" section). Figure 1 shows that the network becomes prohibitively costly as the scale goes up. For instance, interconnecting 65,536 GPUs when the network capacity is 400 Gbps costs $300 million and requires ≈6 MW of power at peak consumption. Consequently, datacenter providers resort to oversubscription to tame costs and energy consumption, worsening potential network deadlocks and degrading performance.

In this article, we show that efficiently training LLMs *does not* require any-to-any connectivity across all GPUs in the network, even for LLMs with all-to-all communication (see the "All-to-All Traffic for MoE Models" section). We propose an immediately deployable solution to lower the cost and energy consumption of LLM datacenters with commodity electrical switches. To do so, we make two primary contributions. First, we analyze the traffic pattern of training LLMs (see the "LLM Traffic Pattern Analysis" section).

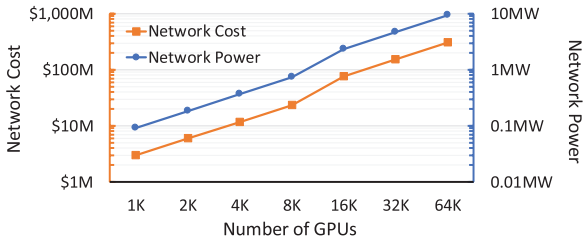


**FIGURE 1.** Network cost and power for a Rail-optimized network for large GPU clusters.

0272-1732 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies.
Digital Object Identifier 10.1109/MM.2025.3540663
Date of publication 13 February 2025; date of current version 29 April 2025.


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:35:13 UTC from IEEE Xplore. Restrictions apply.

We demonstrate that with an optimal parallelization strategy, an LLM training workload requires high-bandwidth (HB) any-to-any connectivity *only within a small subset of GPUs*, and each subset fits within a single GPU platform, such as a Nvidia DGX server. Across the platforms, most communication occurs between *a few GPU pairs with the same local rank* throughout the cluster. As a result, the conventional any-to-any approach to building Clos networks adds unnecessary complexity, cost, and power consumption during distributed LLM training.

Motivated by these observations, we propose a low-cost network architecture that accurately reflects LLM communication requirements, called *Rail-only* (see the "Rail-Only Network Design" section). Instead of forming a Clos to support any-to-any communication for all GPUs, as major GPU manufacturers advocate, our architecture ensures full-bisection bandwidth connectivity among subsets of GPUs with significant network traffic only. Our architecture removes the network equipment that does not carry significant traffic and achieves the same performance as a Rail-optimized network. We provide routing strategies that impose minimal performance overhead on all-to-all communication.

We derive an analytical formulation of LLM training performance and evaluate our Rail-only network architecture with this formulation. We provide insights into the performance impact of different network parameters and compare the cost and power consumption of a Rail-only network to a full-bisection Rail-optimized network. We show our LLM-centric network architecture reduces network cost and power by 38%–76% and 37%–75%, respectively (see the "Network Cost and Power Analysis" section ). Moreover, we show that a Rail-only network achieves the same performance as a Rail-optimized cluster for LLMs without mixture of experts (MoE) layers. Finally, we demonstrate a Rail-only interconnect only incurs 8.2%–11.2% throughput overhead for LLMs with MoEs requiring all-to-all traffic.

## BACKGROUND

### Intraplatform Connectivity: High-Bandwidth Domain

The rise of resource-intensive machine learning (ML) workloads has led to the dominance of GPU-centric platforms optimized for multi-GPU workloads. To accommodate the communication demand of LLM training, these platforms use HB local interconnects within a local domain of GPUs. For instance, Nvidia's DGX H100 server consists of eight H100 GPUs interconnected with NVSwitches, providing 3.6 Tbps of nonblocking bandwidth internally. The AMD MI300X platform employs AMD's Infinity Fabric to connect eight MI300X accelerators in a full-mesh topology with 3.6 Tbps of bandwidth per GPU. These platforms have a unifying property: provisioning several terabits per second of internal bandwidth across GPUs. This article refers to a platform with terabit per second internal bandwidth connectivity as an "HB domain," and the corresponding interconnect as an HB interconnect (HBI). We use Nvidia's DGX A100 and DGX GH200 cluster design throughout the rest of the article to represent two mainline designs with an HB domain size of 8 and 256, respectively.

*OUR ARCHITECTURE REMOVES THE NETWORK EQUIPMENT THAT DOES NOT CARRY SIGNIFICANT TRAFFIC AND ACHIEVES THE SAME PERFORMANCE AS A RAIL-OPTIMIZED NETWORK.*

### Interplatform Connectivity: Network Interface Card Domain

While GPU-centric platforms provide high internal bandwidth, they can scale to a limited number of GPUs. To expand beyond a single platform, operators rely on traditional network technologies, such as Ethernet or Infiniband, to connect different platforms' network interface cards (NICs). This article refers to the interplatform network as the "NIC domain."

Figure 2 illustrates the architecture of a Rail-optimized network. In these networks, each GPU platform with an HB domain of size *K* has *K* total *rails*, where a rail refers to all GPUs with the same local rank in different HB domains. A Rail-optimized network places these GPUs under the same set of switches, which we denote as rail switches. Figure 2 highlights different

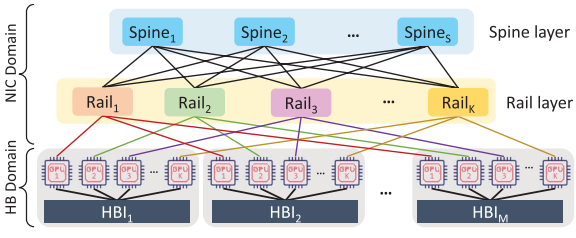


**FIGURE 2.** Illustration of a conventional Rail-optimized network architecture.[5]


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

rails in different colors. Connecting same-rank GPUs to the same rail switches ensures the lowest possible latency. Such connectivity is desirable because an optimal deep neural network (DNN) parallelization strategy concentrates NIC domain traffic on GPUs with the same local rank.

The rest of the network employs layers of spine switches to connect the rail switches to form a full-bisection any-to-any Clos network topology. A Rail-optimized network ensures any pair of GPUs in different HB domains can communicate at the network line rate. For instance, traffic between `GPU 1, Domains 1` and `2` traverses through rail switch 1 only, while traffic between `GPU 1, Domain 1,` and `GPU 2, Domain 2` goes through the respective rails and spine switches.

While the Rail-optimized network architecture takes advantage of the strong locality of DNN training traffic by connecting the same-rank GPUs with the same top-of-rack switch, it overlooks a fundamental question: *Are the spine switches necessary?* Next, we analyze LLM training traffic in greater detail to explore the potential for a spineless network architecture design.

## LLM TRAFFIC PATTERN ANALYSIS

### Traffic Pattern of MegatronLM

We now analyze the traffic pattern generated by LLM training with hybrid data parallelism (DP), tensor parallelism (TP), and pipeline parallelism (PP) by computing the network transfer sizes from the models' hyperparameters and parallelization strategy. We study the GPT-1T model with 1 trillion parameters described in MegatronLM.[4] The models are distributed in a cluster of 384 DGX A100 servers with an HB domain of size eight. We use a ring-based collective communication (CC) to ensure bandwidth optimality.

### LLM Traffic in the NIC Domain

The parallelization strategy employed in MegatronLM induces an insignificant amount of traffic in the NIC domain compared to the HB domains. Figure 3 shows the traffic heatmap of the first pipeline stage and the first four pipeline stages for training the GPT-1T model. Every consecutive set of eight GPUs resides within the same HB domain (highlighted in orange), and GPUs with a distance of 8 between them belong to the same rail (in pink). Figure 3(a) demonstrates the traffic pattern within one pipeline stage; Figure 3(b) shows the traffic across the first four pipeline stages. The traffic volume is significant (~300 GB across GPU pairs) in an HB domain, but it drops to only about 6 GB in the NIC domain. Furthermore, the traffic in the NIC domain never traverses through the spine switches; these network transfers only happen within a rail.

We argue that *all* LLMs without sparse MoE layers distributed with an optimal parallelization strategy always induce sparse, low-volume traffic *within the rails*. By design, only point-to-point traffic from PP or CC traffic from TP and DP exits the HB domains.

For PP, given the symmetry of LLM parallelization, each pipeline stage contains the same number of GPUs. As a result, traffic across stages can be rearranged to traverse the GPUs of the same local rank in the NIC domain only, hence staying within the same rail.

TP and DP can induce CC traffic in both HB and NIC domains. The models in MegatronLM always contain TP and DP within HB and NIC domains, respectively.

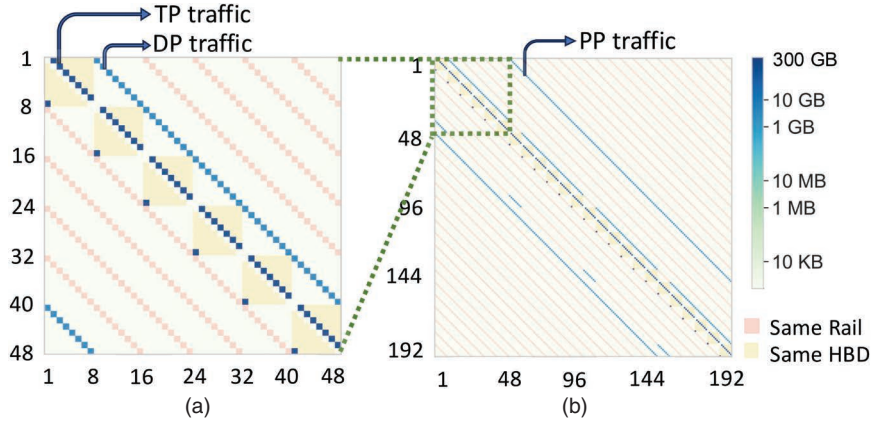


**FIGURE 3.** Traffic heatmaps for GPT-1T in MegatronLM.[4] Highlights show GPUs in the same HB domains and rails. (a) GPT-1T MegatronLM traffic matrix GPU 1 to 48 (one pipeline stage). (b) GPT-1T MegatronLM traffic matrix GPU 1 to 192 (four pipeline stages).


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

While such a partition is common for LLMs, it is not ubiquitous. Training a smaller model using only DP causes all GPUs to participate in the same DP rank and, thus, the same AllReduce operation across both HB and NIC domains. In these cases, the training cluster can use *hierarchical CC* algorithms to achieve near-optimal performance. In the following, we introduce the hierarchical AllGather algorithm and show its traffic in the NIC domain stays within the rails.

Assume the GPUs conducting an AllGather operation are arranged in an $x \times y$ grid, where each $x$ GPU belongs to the same HB domain across $y$ total HB domains. The hierarchical AllGather executes in two phases. First, the algorithm collects partial data for each rail of GPUs without transferring data in the HB domain. If the total data to run AllGather are of size $D$, then the amount of data exchanged in the network by all GPUs is $D(y-1)/x$. This operation creates larger data shards for the GPUs to rerun AllGather within each HB domain. Each HB domain conducts an AllGather in the second phase, inducing a total transfer of $D(x-1)$. Note that traffic never exits the rail in the NIC domain. Note that it is possible to pipeline the two phases for near-optimal performance. Recent works on synthesizing communication algorithms focus on generating optimal CC schedules for a given network schedule and, therefore, are also capable of generating collectives without cross-rail traffic. The performance penalty is minor due to omitting the spine network because of the bandwidth asymmetry across the HB and NIC domains.

### All-to-All Traffic for MoE Models

LLMs with sparsely gated MoE layers exhibit a different traffic pattern than the models described above. MoE layers increase the size of LLMs without introducing significant additional computational requirements. With MoEs, part of the models are replaced by a set of expert neural networks, where a gating network routes each token to different experts. The typical parallelization strategy for MoEs is expert parallelism (EP), where each expert is distributed to a different GPU in the cluster.

Unlike traditional LLMs, MoEs with EP require each expert to communicate with the rest of the model, creating a dense communication pattern. The exact traffic heatmap depends on the token distribution. We assume a uniform token distribution for simplicity. Figure 4 shows the traffic matrix for training the MoE-1.3B model from DeepSpeedMoE,[6] with 16 DGX A100 servers. The model contains 128 experts. The static part uses DP, while the MoE part uses EP. Since each GPU contains a different expert, a uniform token distribution generates a uniform all-to-all traffic pattern across the network. At first glance, such traffic patterns make the spine switch crucial, as traffic across GPUs on different rails will traverse through spine switches. However, as we show next, we do not have to rely on spine switches: using the HB domain as a forwarding step achieves near-optimal performance.

## RAIL-ONLY NETWORK DESIGN

*Rail-only* is a novel network architecture that deviates from the any-to-any GPU connectivity paradigm.

### Architecture Design

Like a conventional Rail-optimized GPU cluster, a Rail-only network keeps the HB domains but omits the NIC domain's full-bisection connectivity. Instead, it ensures the *GPUs within each rail* are connected with a full-bisection network.

> *MoE LAYERS INCREASE THE SIZE OF LLMs WITHOUT INTRODUCING SIGNIFICANT ADDITIONAL COMPUTATIONAL REQUIREMENTS.*

A straightforward illustration of our proposed architecture is to remove the spine switches and repurpose the uplinks connecting rail switches to the spine as downlinks to GPUs. Hence, a dedicated Clos network connects each rail. Compared to the Rail-optimized architecture, the Rail-only design saves the number of switches and transceivers by building multiple smaller Clos networks for individual rails, requiring fewer layers of switches.

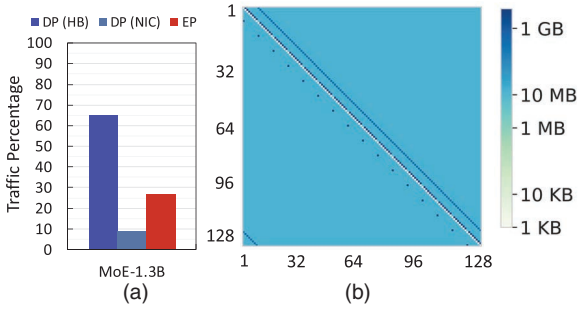


**FIGURE 4.** Traffic volume distribution and heatmap for the MoE-1.3B model in DeepSpeedMoE,[6] assuming uniform token distribution. (a) Traffic volume distribution and (b) MoE-1.3B traffic matrix with uniform token distribution.


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

## Routing in Rail-Only Networks

Interrail communication in a Rail-only network is possible by forwarding data through HB domains. For instance, in a Rail-only network, messages from `GPU 1, Domain 1` to `GPU 2, Domain 2` (Figure 5) are routed through the first HB domain to `GPU 2, Domain 1`, and then transmitted to the final destination through the network. This strategy has already been implemented in the Nvidia Collective Communication Library as "PCIe×NVLink" (PXN).[2] Such a routing scheme induces a *bandwidth-tax*,[7] where the physical traffic increases in the network due to forwarding. Consider the LLMs with MoE layers described in the "All-to-All Traffic for MoE Models" section. At first glance, this traffic pattern is challenging for the Rail-only network. Most all-to-all traffic requires forwarding. However, forwarding network traffic incurs a small overhead since the HB domain is much faster than the NIC domain.

Consider a GPU cluster with HB domain bandwidth $B_{HB}$, connected with a Rail-only or Rail-optimized network with bandwidth $B_{NIC}$. In the Rail-optimized network, every GPU sends traffic directly to its destinations through the HB and full-bisection NIC domains. For the Rail-only network, the two-step algorithm first runs an all-to-all *within each rail*, preparing each GPU to have all data to send on its rail. Then, within a rail, each GPU runs a second all-to-all *in the HB domain* to finish the transfers with a larger data shard.

The second phase forwarding induces a *bandwidth-tax*,[7] and the slowdown factor is approximately $B_{NIC}/B_{HB}$ due to retransmitting the forwarded traffic. This factor equals 8.2% and 11.2% for the DGX A100 and DGX H100 generations of GPU platforms, respectively. Furthermore, this slow-down only applies to all-to-all communication, which accounts for 27% of the total traffic (Figure 4). The low overhead of the Rail-only network renders it suitable for application beyond LLM training, as other DNN training, inference, and some GPU-based HPC workloads exhibit similar traffic localities.

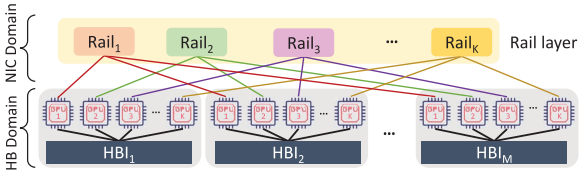


**FIGURE 5.** Our proposed *Rail-only* network provides full-bisection connectivity to individual rails.

## Fault Tolerance Properties of Rail-Only Networks

This section investigates the fault tolerance properties of Rail-only networks.

- *Link and switch:* Failures in switches or links render connected GPUs unavailable in both Rail-optimized and Rail-only networks. However, our design reduces potential failure points.
- *GPU platform:* In DGX-like clusters, each server acts as an HB domain. Upon server failure, tasks migrate to a healthy server without disrupting Rail-only connectivity. In GB200-like clusters, superchip module failures resemble single GPU failures, discussed next.
- *Single GPU:* When an idle GPU exists in the same HB domain as a failed one, we can employ optical reconfigurable switches to enable similar recovery by dynamically reconfiguring rails in a Rail-only network. This approach aligns with optical switch-based fault recovery mechanisms.[8] For fully occupied HB domains, the resolution depends on their size. In smaller HB domains, migrating all tasks within the failed domain addresses the issue. For larger HB domains, we also employ optical switches to dynamically reconfigure a subset of GPUs, preserving HB domain integrity after failures.

## ITERATION TIME MODELING

This section analyzes the training iteration time of LLMs by incorporating the impact of the parallelization strategy and the training infrastructure. We use

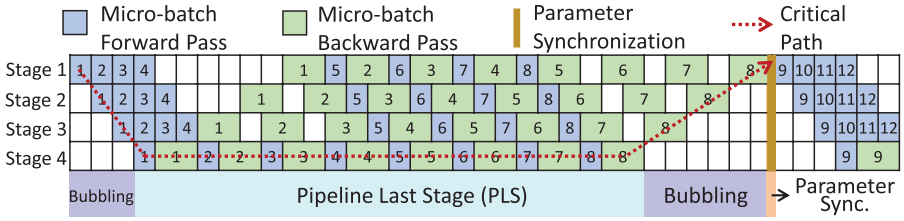


**FIGURE 6.** The 1F1B pipeline schedule and its critical path.


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

this analytical model to evaluate Rail-only networks in the "Evaluation" section. Other tools, like Calculuon,[9] can also be used for this purpose.

A typical LLM training session uses the one-forward-one-backward (1F1B) pipeline schedule, displayed in Figure 6. Given the uniform structure of LLMs under DP, PP, and TP, the forward and backward execution times for each microbatch across GPUs remain the same.[4] This consistency allows us to identify the critical path in a training iteration, shown as the red arrow in Figure 6. This path decomposes into three parts:

- *Pipeline bubble time*: Time to fill and drain the pipeline of micro-batches
- *Pipeline last stage (PLS) time*: Time in the last pipeline stage to perform consecutive 1F1B computations
- *Parameter synchronization time*: Time to synchronize parameters for DP.

The rest of this section dissects each term with explicit computation and communication times.

## Pipeline Bubble Time

We break down the pipeline bubble time into communication and computation times. The computation occurs in each stage except the last one, with one forward and one backward pass for each microbatch. Therefore, the compute time for a bubble is the number of pipeline stages except the last one, multiplied by the computation time for each microbatch. The computation time of a microbatch can be estimated through the floating point operations per second (FLOP) requirement or profiled on hardware with a single GPU.

To determine the communication time, let the amount of traffic induced by each microbatch be $D_p$ across two pipeline stages. Such a transfer occurs across every pipeline stage throughout the pipeline filling and emptying. Therefore, the communication volume is $2D_p$ multiplied by the number of pipeline stages, except for the last stage, from which we can derive the communication time.

## PLS Time

We analyze the computation and communication times separately in the last stage of the pipeline. Let the total number of microbatches be $m$. The last stage has $m$ microbatches going through consecutively; therefore, the PLS computation time is $m$ times the computation time of a single microbatch.

For communication, the TP and PP strategies both generate network traffic across GPUs. For TP, each parallelized attention and feed-forward operator in the transformer block induces one AllGather and one ReduceScatter collective.[10] Let $\texttt{AGtime}(D)$ denote the completion time of AllGather on data of size $D$, and let $D_t$ denote the amount of AllGather data for each operator of a microbatch. The total communication time is the sum of the CC time, $\texttt{AGtime}(D_t)$, for all operators within a single pipeline stage.

For PP, the pipeline traffic may overlap with the computation. To be conservative, we assume GPUs do not perform computation when sending and receiving PP traffic. Each microbatch sends or receives $D_p$ bytes of traffic on forward and backward passes. Thus, the pipeline communication volume in PLS is twice the number of microbatches times $D_p$.

## Parameter Synchronization Time

In our modeling, the parameter synchronization time consists of an AllReduce operation of LLM parameters in the first stage of the pipeline. We only model the communication time of the AllReduce operation as its computation incurs minimal overhead. Let $D_d$ denote the amount of AllReduce data. Since AllReduce is equivalent to a ReduceScatter followed by an AllGather, it induces twice the runtime as AllGather for the same amount of data, the parameter synchronization time becomes $2\texttt{AGtime}(D_d)$. This time overlaps with backpropagation.

# EVALUATION

## Iteration Time Modeling

We demonstrate the accuracy of our analytical model by comparing its estimation of the iteration time to

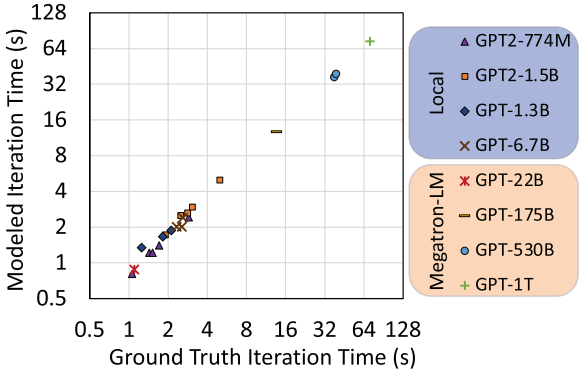


**FIGURE 7.** Iteration time comparison of ground truth and our formulation for different GPT models.


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

testbed measurements. We use two types of testbed measurements: for small GPTs, we directly measure the training iteration time on a testbed with four to 12 servers, each with one Nvidia A100 GPU, for different parallelization strategies and batch sizes. For large GPTs, we use the measurements provided in prior work.[10] In particular, we use our model to estimate the iteration times by taking the hyperparameters in their evaluation setup.

Figure 7 illustrates the difference between the ground truth iteration times and the iteration times by our analytical model, where each legend represents a particular GPT model with a different parallelization strategy and training batch size on our testbed and from MegatronLM. On average, our model's estimations are within 10% of the ground truth. The discrepancy comes from our idealistic modeling of GPU computation and communication, assumptions on how computation and communication overlap, and underestimation of memory overhead. Next, we use our analytical model to estimate the training iteration time of Rail-only interconnects.

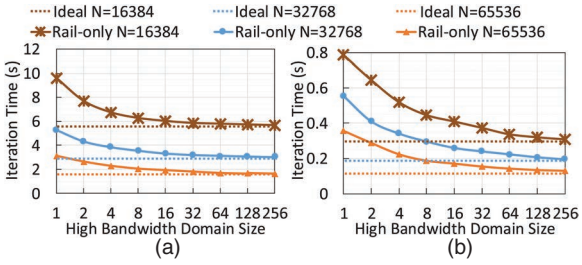


**FIGURE 8.** Iteration time as HB domain size changes. (a) GPT-1T and (b) GPT-146B.

## What Is the Optimal Size of an HB Domain?

Intuitively, increasing HB domain size reduces the interplatform network overhead. In Figure 8, we vary HB domain size ($K$) and plot training iteration time for GPT-1T and GPT-146B MegatronLMs for clusters with 16,384, 32,768, and 65,536 H100 GPUs. The global batch size is 4096 and 1024 for GPT-1T and GPT-146B, respectively. We enumerate all possible parallelization strategies for each cluster size and use the optimal parallelization strategy with the bandwidth and computational ability parameters of DGX GH200. To capture the ideal iteration time, we compute training iteration time when a full-bisection NVLink fabric connects every GPU.

Figure 8 illustrates the change of iteration time with HB domain sizes. Compared to the ideal case (all GPUs are under a monolithic HB domain), GPT-146B with an HB domain of 256 is 4.1% slower, while GPT-1T is 0.9% slower. However, the *marginal gain* decreases as HB domain size increases. For the larger GPT-1T model, the training iteration time plateaus above 32 GPUs due to Amdhahl's law, as the bottleneck shifts to computation, suggesting diminishing returns from the high cost of building larger HB domains.

## Network Cost and Power Analysis

Our Rail-only network architecture reduces the network resources for LLM training by eliminating unused network hardware. Following prior work,[11] we calculate the number of switches and transceivers required for each network design and derive network equipment cost and peak power consumption, for variable cluster sizes and network switch radix, based on numbers

**TABLE 1.** Number of switches and transceivers for different clusters.

| No. GPUs | Switch Radix | No. Switches ($N_{SW}$) | | No. Transceivers ($N_{TR}$) | | Savings | |
|---|---|---|---|---|---|---|---|
| | | SOTA | Rail-Only | SOTA | Rail-Only | Cost | Power |
| 32,768 | 64 | 2560 | 1536 | 196,608 | 131,072 | 38% | 37% |
| | 128 | 1280 | 256 | 196,608 | 65,536 | 77% | 75% |
| | 256 | 384 | 128 | 131,072 | 65,536 | 62% | 60% |
| 65,536 | 64 | 5120 | 3072 | 393,216 | 262,144 | 38% | 37% |
| | 128 | 2560 | 1536 | 393,216 | 262,144 | 38% | 37% |
| | 256 | 1280 | 256 | 393,216 | 131,072 | 77% | 75% |

SOTA: state of the art.


Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.

reported by prior research and by vendors.[11,12,a] We build a minimum layer Clos network for each architecture with the provided switch radix and calculate the number of required switches and transceivers.

The last two columns of Table 1 provide the cost and power savings of a Rail-only interconnect over the Rail-optimized architecture. Our design reduces network cost by 38% to 77% (117 to 234 million dollars) and power consumption by 37% to 75% (1.7 to 6.9 MW) while achieving equivalent performance. Switches with a radix of 64 provide the worst-case reduction in both cluster sizes. Even in this case, the Rail-only design requires two tiers of switches for each rail, consuming three-quarters of the total number of switches compared to the Rail-optimized network that requires three tiers of switches.

## CONCLUSION

We analyze the traffic pattern of LLM training with hybrid parallelism. Our novel Rail-only architecture aligns with LLMs' distinct traffic characteristics. It leads to 38% to 77% cost reductions and 37% to 75% power savings while maintaining identical performance to the state-of-the-art GPU networks.

## ACKNOWLEDGMENTS

The authors are supported by Defense Advanced Research Projects Agency FastNICs 4202290027, National Science Foundation (NSF) CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, and Sloan Fellowship FG-2022-18504.

## REFERENCES

1. A. Grattafiori, "The Llama 3 herd of models," 2024, *arXiv:2407.21783*.
2. K. Mandakolathur and S. Jeaugey, "Doubling all2all performance with NVIDIA collective communication library 2.12," *Nvidia*, Feb. 28, 2022. [Online]. Available: https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/
3. C. Guo et al., "RDMA over commodity Ethernet at scale," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 202–215, doi: 10.1145/2934872.2934908.
4. D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2021, pp. 1–15, doi: 10.1145/3458817.3476209.
5. "NVIDIA DGX SuperPOD." NVIDIA Docs. Accessed: Sep. 20, 2024. [Online]. Available: https://docs.nvidia.com/https:/docs.nvidia.com/dgx-superpod-reference-architecture-dgx-h100.pdf
6. S. Rajbhandari et al., "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale," in *Proc. Mach. Learn. Res.*, 2022, vol. 162, pp. 18,332–18,346.
7. W. M. Mellette et al., "RotorNet: A scalable, low-complexity, optical datacenter network," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 267–280, doi: 10.1145/3098822.3098838.
8. N. Jouppi et al., "TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2023, pp 1–14, doi: 10.1145/3579371.3589350.
9. M. Isaev, N. Mcdonald, L. Dennison, and R. Vuduc, "Calculon: A methodology and tool for high-level co-design of systems and large language models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2023, pp 1–14, doi: 10.1145/3581784.3607102.
10. V. Korthikanti et al., "Reducing activation recomputation in large transformer models," 2022, *arXiv:2205.05198*.
11. W. Wang et al., "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," 2023, *arXiv:2202.00433*.
12. "NVIDIA Docs Hub: QM9700/QM9790." Nvidia**.** 2023. Accessed: Sep. 20, 2024. [Online]. Available: https://docs.nvidia.com/networking/display/qm97x0pub

[a]Cost: $199 per transceiver, $694 per switch port for 400 Gbps; power: 9 W per transceiver, 18 W per switch port.

**WEIYANG WANG** is a Ph.D. degree student at the Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, 02139, USA. His research interests include network architectures for machine learning clusters and reconfigurable networks. Wang received his S.M. degree from the Massachusetts Institute of Technology. Contact him at weiyangw@mit.edu.

**MANYA GHOBADI** is an associate professor at the Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, 02139, USA. Her research interests include computer networks, building efficient systems for machine learning, and high-performance cloud infrastructure. Ghobadi received her Ph.D. degree from the University of Toronto. Contact him at ghobadi@mit.edu.

Authorized licensed use limited to: MIT. Downloaded on May 10,2025 at 18:55:13 UTC from IEEE Xplore. Restrictions apply.