# Rail-only: A Low-Cost High-Performance Network for Training LLMs with Trillion Parameters


Weiyang Wang
MIT CSAIL
weiyangw@mit.edu

Manya Ghobadi
MIT CSAIL
ghobadi@mit.edu

Kayvon Shakeri
Meta
kvon@meta.com

Ying Zhang
Meta
zhangying@meta.com

Naader Hasani
Meta
naaderh@yahoo.com



***Abstract*—This paper presents a low-cost network architecture for training large language models (LLMs) at hyperscale. We study the optimal parallelization strategy of LLMs and propose a novel datacenter network design tailored to LLM's unique communication pattern. We show that LLM training generates sparse communication patterns in the network and, therefore, does not require any-to-any full-bisection network to complete efficiently. As a result, our design eliminates the spine layer in traditional GPU clusters. We name this design a *Rail-only* network and demonstrate that it achieves the same training performance while reducing the network cost by 38% to 77% and network power consumption by 37% to 75% compared to a conventional GPU datacenter. Our architecture also supports Mixture-of-Expert (MoE) models with all-to-all communication through forwarding, with only 8.2% to 11.2% completion time overhead for all-to-all traffic. We study the failure robustness of Rail-only networks and provide insights into the performance impact of different network and training parameters.**


## I. Introduction

Large Language Models (LLMs) are among the most complex and computationally intensive Deep Neural Networks (DNNs). The GPT3 model from 2020 already requires 355 GPU-years on Nvidia's V100 GPUs [1], [2], while the recent GPT4 model is estimated to have trillions of parameters and takes months to train [3], [4]. As Moore's law slows down, the growth rate of LLM size and computation requirement exceeds the advancement of accelerators, making hyper-scale GPU datacenters inevitable. Our conversations with lead machine learning architects in the industry indicate that the next-generation LLMs likely require over 30,000 GPUs of computing power to finish training within a reasonable time.

GPU manufacturers have invested in high-bandwidth platforms, such as NVLink [5] and Infinity Fabric [6], to enable efficient multi-GPU training. These platforms provide several Tbps of bandwidth within a few GPUs but are not scalable. To connect multiple GPU platforms, state-of-the-art approaches rely on traditional lossless network solutions, such as RDMA over converged ethernet (RoCE) or Infiniband. In particular, today's GPU clusters employ an architecture called a "Rail-optimized" network. This architecture is derived from the classical Clos network [7] to provide any-to-any connectivity to *all GPUs* in a training cluster.

However, scaling a Clos network to tens of thousands of GPUs is challenging. Previous work demonstrated that large-scale lossless networks are prone to deadlocking and PFC storms [8]–[12], degrading the performance. Furthermore, as the scale increases, Clos architectures become prohibitively costly [13]. For instance, today, a full-bisection Clos fabric interconnecting 30,000 GPUs with 400 Gbps network capacity costs $200 million. At the same time, deploying such a network requires provisioning for $\sim$4.6 megawatts of peak power consumption. Consequently, datacenter providers resort to over-subscription to tame costs and energy consumption, worsening deadlocking and degraded performance problems.

In this paper, we show that efficiently training LLMs *does not* require any-to-any connectivity across all GPUs in the network, even for DNNs with sparsely gated Mixture-of-Expert (MoE) layers, which generate all-to-all communication (§III-C). As a result, 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 (§III). We demonstrate that with an optimal parallelization strategy, an LLM training workload requires high-bandwidth any-to-any connectivity *only within a small subset of GPUs*, and each subset fits within a single GPU platform, such as an Nvidia DGX server. Across the platforms, most communication occurs between *a few GPU pairs with the same rank* throughout the cluster. As a result, the conventional any-to-any approach for building Clos networks adds unnecessary complexity, cost, and power consumption for distributed LLM training.

Motivated by the above observations, we then propose a low-cost network architecture that accurately reflects LLM communication requirements, called *Rail-only* (§IV). Instead of forming a Clos to support any-to-any communication, as advocated by major GPU manufacturers [14], our architecture removes the spine layer of switches and only connects sets of GPUs with significant network traffic. Hence, compared to the state-of-the-art Rail-optimized design, our network 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 for all-to-all communication. We also analyze our design's fault-tolerance properties and provide recovery methods from failure cases (§IV-C).

We evaluate the performance of our Rail-only network architecture using an analytical formulation and provide insights into the performance impact of different network and training parameters. We compare the cost and power

consumption of a Rail-only network to a full-bisection bandwidth any-to-any Clos network and show that our LLM-centric network architecture reduces the network cost and power by 38%–76% and 37%–75%, respectively (§V-E). Moreover, we show that a Rail-only network achieves the same performance as a Rail-optimized cluster for LLMs without MoE layers. Finally, we demonstrate that a Rail-only interconnect only incurs 8.2%–11.2% throughput overhead for LLMs with MoEs that require all-to-all traffic.

## II. Background

### A. Intra-platform Connectivity: High-bandwidth Domain

The rise of resource-intensive ML workloads led to the dominance of GPU-centric platforms optimized for multi-GPU workloads. To accommodate the communication demand, these platforms use high-bandwidth local interconnects within a local domain of GPUs. Depending on the manufacturer, these GPU-centric platforms differ in computing FLOPs, GPU and CPU architectures, or even physical interconnect technology. However, these platforms all share a unifying property: they provision *several Tbps of internal bandwidth* across GPUs.

For instance, Nvidia's DGX H100 server [14] consists of eight H100 GPUs interconnected with NVSwitches, providing 3.6 Tbps of non-blocking bandwidth internally. The GB200 NVL72 computer announced recently connects 36 GB200 Superchips with fifth-generation NVLink within a rack at 7.2 Tbps per GPU [15]. The AMD MI300X platform, on the other hand, employs AMD's Infinity Fabric to connect eight MI300X accelerators in a full-mesh topology with 3.6 Tbps of bandwidth per GPU [6]. Similar platforms such as Nvidia's DGX GH200 Super Computer have utilized multi-tiered NVSwich topologies to scale the platform's size up to 256 GPUs while maintaining 3.6 Tbps full-bisection intra-GPU bandwidth [16]. This paper refers to a platform with Tbps internal bandwidth connectivity as a "high-bandwidth (HB) domain", and the corresponding interconnect as HB interconnects (HBI).

### B. Inter-platform Connectivity: NIC Domain

While GPU-centric platforms provide high internal bandwidth using NVLink or Infinity Fabric technologies, they can only 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 the NICs of different platforms. This paper refers to the inter-platform network as the "NIC domain."

The state-of-the-art interconnection in the NIC domain is based on a well-known network architecture called a *Rail-optimized network* [17]. This architecture is ubiquitously used for high-performance computing (HPC) workloads. As we discuss next, Rail-optimized networks are better suited for DNNs than conventional CPU-centric datacenter networks. However, given that Rail-optimized networks are primarily designed for HPC workloads, they miss a significant opportunity to further leverage the unique traffic patterns of LLM training workloads (§III).

First, let us consider a conventional datacenter design specialized to serve unpredictable and bursty CPU-heavy workloads. This architecture, known as a Clos network [7], [18], provides any-to-any connectivity between server pairs. Clos networks are well-studied in the system and networking community and are the de facto infrastructure for storage, cloud, and map-reduce workloads.

The Rail-optimized network for GPU training clusters evolves from the datacenter Clos network [17], [19], illustrated in Figure 1. For a GPU platform with an HB domain of size $K$, there are $K$ total rails, where a *rail* comprises GPUs with the same local rank that belong to different HB domains [20]. A Rail-optimized network places these GPUs under the same set of switches, which we denote as rail switches. Figure 1 highlights rail one and rail $K$ in red and yellow colors, respectively. Connecting same-rank GPUs to the same rail switches ensures the lowest possible latency across them. Such connectivity is desirable because an optimal DNN parallelization strategy concentrates its NIC domain traffic between GPUs with the same local rank [17].

Rail-optimized architectures enjoy low latency between GPUs in the same rail. 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. This network ensures that any pair of GPUs in different HB domains can still communicate at the network line rate of hundreds of Gbps. For instance, traffic between `GPU 1, Domain 1` and `GPU 1, Domain 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 the 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 ToR switch, it overlooks a fundamental question: Are the spine switches necessary? In the next section, we analyze LLM training traffic in greater detail to explore the potential for a spineless network architecture design.

## III. LLM Traffic Pattern Analysis

### A. Traffic Pattern of MegatronLM

We now analyze the traffic pattern generated by LLMs with hybrid data, tensor, and pipeline parallelism by computing the network transfer sizes from the model hyperparameters and the parallelization strategy. We first look at a series of GPT models with 145.6 billion, 310.1 billion, 539.6 billion, and 1 trillion parameters described in Table 1 of MegatronLM [21] paper, distributed across up to 3072 GPUs. The models are distributed in a cluster of up to 384 DGX A100 servers with an HB domain of size eight. Our analysis uses the same parallelization strategy from MegatronLM to ensure optimal GPU utilization. We use the ring-based collective communication since it is bandwidth-optimal and the default algorithm in NCCL.

There are three primary types of communication: AllGather and ReduceScatter traffic from tensor parallelism (TP), AllReduce traffic from data parallelism (DP), and point-to-point

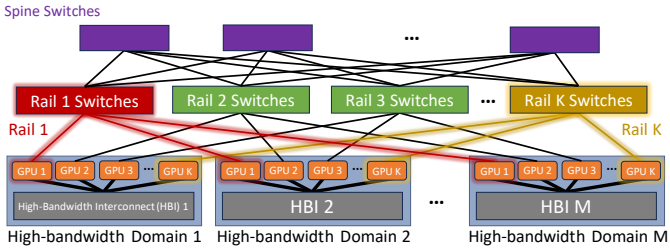



Fig. 1. A GPU datacenter with Rail-optimized, any-to-any Clos networks [19].

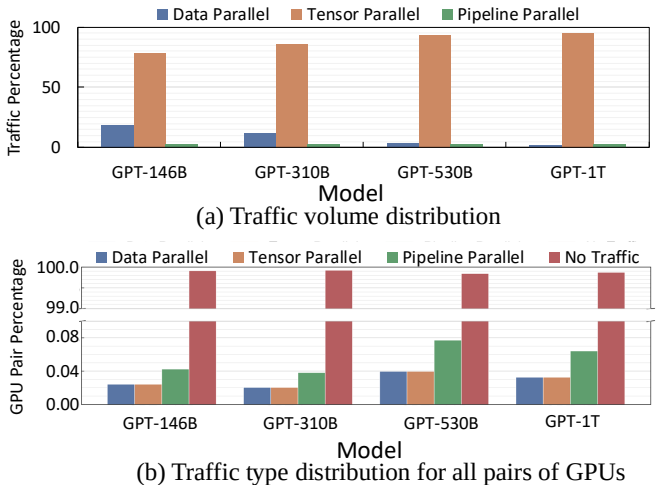


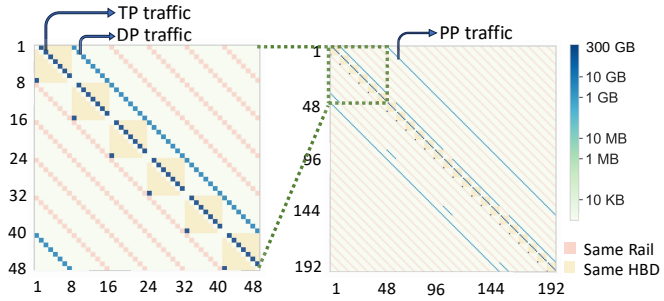

Fig. 2. (a) The traffic volume from different parallelization dimensions; (b) The communication type across all GPU pairs.

traffic from pipeline parallelism (PP). Figure 2a illustrates the volume percentage for each type of communication for one training iteration. Figure 2b shows the communication type distribution across all GPU pairs.

The TP traffic happens within GPUs participating in a TP rank, which occupies an HB domain. The DP and PP traffic happen in the NIC domain, and their volume is significantly smaller than TP traffic, as illustrated by Figure 2a. While traffic from different parallelism does not overlap between different pairs of GPUs, Figure 2b indicates that over 99% of GPU pairs carry *no traffic* and less than 0.04% of GPU pairs carry TP traffic. Simultaneously, Figure 2a suggests TP traffic accounts for over 75% of the total transmitted data. Recall that TP traffic stays within HB domains, suggesting efficient usage of HB domain bandwidth and low demand in the NIC domain. This pattern is consistent across all GPT models we studied, indicating that building a GPU datacenter with any-to-any connectivity on top of HB domains for LLM models is excessive.

### *B. Traffic in the NIC Domain for LLMs*

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 for training the GPT-1T model. In this plot, every




(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)

Fig. 3. Traffic heatmaps for GPT-1T in MegatronLM [21]. Highlights show GPUs in the same HB domains and rails.

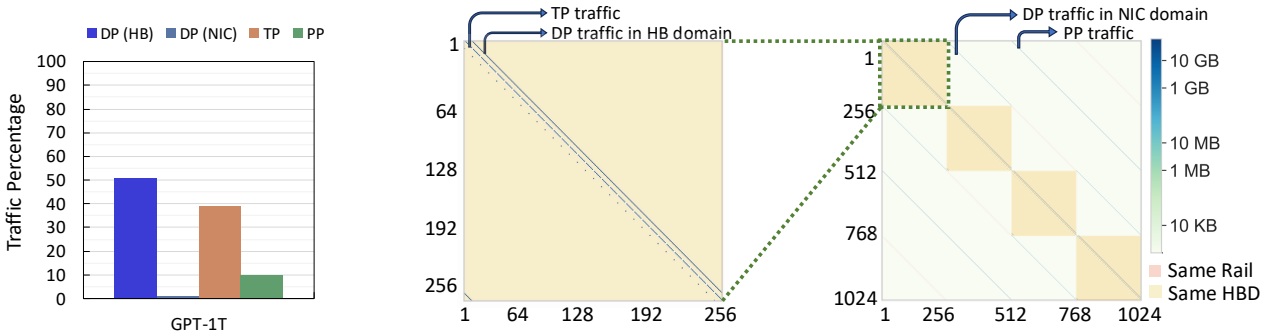
consecutive set of eight GPUs resides within the same HB domain (highlighted in orange), and GPUs with a distance of 8 between them belong to the same rail (highlighted in pink). Figure 3a demonstrates the traffic pattern within one pipeline stage, while Figure 3b shows the traffic across the first four pipeline stages. The traffic volume is significant (∼300 GB across GPU pairs) in an HB domain, while the communication drops to only about 6 GB in the NIC domain. Furthermore, the transfers in the NIC domain never traverse 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 in *within the rails*. By design, the only traffic exiting the HB domains is point-to-point traffic from pipeline parallelism or collective communication traffic (AllGather, ReduceScatter, and AllReduce) from TP and DP.

For PP, due to the symmetry of LLM parallelization, each pipeline stage contains the same number of GPUs. As a result, the pipeline stages can always be placed such that traffic across stages traverses the GPUs of the same rank in the NIC domain only, hence staying within the same rail.

TP and DP can induce collective communication traffic in both the HB and the NIC domains. The examples from MegatronLM always have TP and DP contained within HB and NIC domains only, respectively. While such a partition is common for LLMs, it is not ubiquitous. For example, 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 could use *hierarchical collective communication* algorithms that achieve near-optimal performance. Below, we introduce these algorithms, analyze their performance, and illustrate that their traffic in the NIC domain stays within rails.

Hierarchical collective communication algorithms are designed for a multi-tiered network topology. We introduce the hierarchical AllGather algorithm here and note that for the other collectives happening in LLM training, ReduceScatter achieves the same performance by inverting the schedule of AllGather, and AllReduce is equivalent to a ReduceScatter followed by an AllGather. We focus on the bandwidth performance and ignore the latency, as the data transmission




(a) Traffic volume distribution

(b) GPT-1T GH200 traffic matrix, GPU 1 to 256 (one HB domain)

(c) GPT-1T GH200 traffic matrix GPU 1 to 1024 (two pipeline stages)

Fig. 4. Traffic distribution and heatmaps for GPT-1T, distributed on 16 DGX GH200s. Note that DP (NIC) accounts for 0.8% of the total traffic percentage. The "Same-Rail" legend on Figure 4 appears for GPUs whose ranks are 256 apart.

is significant during LLM training; thus, the communication runtime is bandwidth-dominated. Table I lists the variables used in this section. We assume the GPUs conducting an AllGather operation are arranged into an $x \times y$ grid, where each $x$ GPU belongs to the same HB domain and across $y$ total HB domains. The basic 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 is of size $D$, then the amount of data exchanged in the network by all GPUs is $D(y-1)/x$. This operation effectively creates larger data shards for the GPUs to rerun AllGather within each HB domain. Therefore, each HB domain conducts an AllGather in the second phase, inducing a total transfer of $D(x-1)$. Assume the $x$ GPUs within an HB domain have bandwidth capacity $C_F$ and $y$ GPUs in the NIC domain have bandwidth $C_S$, then the total AllGather runtime is

$$\texttt{AGtime}(D, x, y, C_F, C_S) = \frac{(y-1)D}{xyC_S} + \frac{(x-1)D}{xC_F} \quad (1)$$

Like PP communication, by appropriately mapping the logical $x \times y$ GPUs to the GPU cluster, this algorithm only induces traffic for GPUs within the same rail.

Figure 4a shows the traffic pattern of training GPT-1T with hierarchical collective communication spanning both NIC and HB domains, differing from the MegatronLM cases. We compute and analyze the traffic pattern of training the GPT-1T model, with a batch size of 4096, distributed in a cluster composed of 16 Nvidia DGX GH200 supercomputers [16] (4096 GPUs). Each DGX GH200 supercomputer comprises a two-tier NVSwitch architecture, facilitating 3.6 Tbps GPU-to-GPU bandwidth across 256 H100 GPUs. Additionally, each GPU has a Connect-X7 HCA Infiniband network interface [16], which provides 400 Gbps network bandwidth in/out of each GPU. In this setup, each DGX GH200 supercomputer forms an HB domain. Figure 4 illustrates the traffic volume percentage and heatmap in this setting. The parallelization strategy has a total data parallel degree of 64, which spans 32 GPUs in each HB domain and two HB domains across the network. Figures 4b and 4c illustrate the traffic heatmap of the hierarchical AllReduce algorithm, which splits the AllReduce traffic among each DP group. Note that the network traffic stays within a rail (GPUs with a distance of 256 apart). The hierarchical algorithm efficiently utilized the bandwidth in the HB domain to carry $98\%$ of the AllReduce traffic, while the network domain carries the other $2\%$.

### C. All-to-All Traffic for Mixture-of-Expert Models

LLMs with sparsely gated Mixture-of-Expert (MoE) layers exhibit a different traffic pattern than the models described above. MoE layers provide an alternative way to increase the size of LLMs without introducing significant additional computational requirements. With MoEs, part of the model is replaced by a set of "expert" neural networks, where a *gating network* routes each token to different experts, thereby only activating part of the model. The typical parallelization strategy for MoEs is expert parallelism, where each expert is distributed to a different GPU in the cluster.

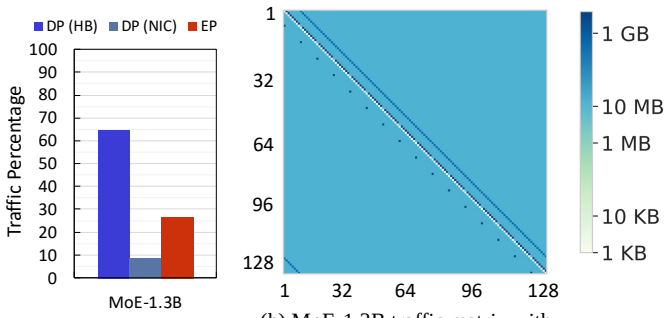
Unlike traditional LLMs, MoEs with expert parallelism require each expert to communicate with the rest of the model, creating a dense communication pattern. The exact traffic heatmap depends on the gating network and the token distribution. In this section, we assume a uniform token distribution for simplicity. Figure 5 shows the traffic matrix of training the MoE-1.3B model from DeepSpeedMoE [22], with 16 DGX A100 servers. The model contains 128 experts. The static part of the model uses DP, while the MoE part uses expert parallelism. 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 in the rail-optimized network crucial, as the traffic across GPUs on different rails will traverse through spine switches. However, as we show in the next section, we do not have to rely on the spine switches: using the HB domain as a forwarding step achieves near-optimal performance.

## IV. Rail-only Network Design

Based on the observations above, this section proposes *Rail-only*, a novel network architecture that diverts from the any-to-any GPU connectivity paradigm. We first introduce the

TABLE I
VARIABLES USED IN SECTION III AND IV-B.

| | |
|---|---|
| $x$ | GPU grid dimension in HB domains. |
| $y$ | GPU grid dimension in NIC domains. |
| $C_F$ | Bandwidth of HB domains. |
| $C_F$ | Bandwidth of NIC domains. |
| $D$ | Data transfer size between a pair of GPUs. |
| $T_{a2a}^{Rail-opt}$ | All-to-all traffic completion time for Rail-optimized networks. |
| $T_{a2a}^{Rail-only}$ | All-to-all traffic completion time for Rail-only networks. |



(a) Traffic volume distribution

(b) MoE-1.3B traffic matrix with uniform token distribution

Fig. 5. Traffic volume distribution and heatmap for the MoE-1.3B model in DeepSpeedMoE [22], assuming uniform token distribution.

architecture design of a Rail-only network. We then discuss routing policy and fault-tolerance properties of Rail-only interconnects.

### A. Architecture Design

Figure 6 illustrates our *Rail-only* network architecture. Compared to a conventional Rail-optimized GPU cluster, shown in Figure 1, our Rail-only network keeps the HB domains but omits the full-bisection connectivity *for all GPUs* in the NIC domain. Instead, we only ensure that *GPUs within each rail* are connected with a full-bisection network.

A straightforward illustration of our proposed architecture is to remove the spine switches (Figure 1) and re-purpose 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 in the network.

### B. Routing in Rail-only Networks

Our Rail-only network architecture removes network connectivity across GPUs with different ranks in different rails. Such communication is still possible by forwarding the data through HB domains. For instance, a message from `GPU 1, Domain 1` to `GPU 2, Domain 2` in Figure 6 can first route through the first HB domain to `GPU 2, Domain 1` and then be transmitted to the final destination through the network. Previous work has shown that such a routing scheme induces a *bandwdith-tax* [13], [23], [24], where the physical traffic increases in the network due to forwarding. However,

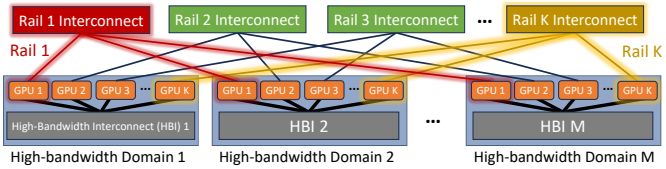


Fig. 6. Our proposal: a *Rail-only* network providing full-bisection connectivity to individual rails.

in this section, we show that due to the bandwidth asymmetry between the HB and the NIC domain, the performance degradation caused by bandwidth tax is negligible.

We use LLMs with MoE layers described in Section III-C as an example, as it generates a challenging all-to-all communication pattern. At first glance, this traffic pattern is challenging for the Rail-only network. Most of the all-to-all traffic requires forwarding. However, since the HB domain is much faster than the NIC domain, forwarding network traffic on the HB domain incurs a slight overhead. Below, we derive the slow-down factor for uniform all-to-all traffic using a two-step forwarding routing algorithm for the Rail-only network. Note that this strategy has already been implemented in NCCL as "PCIe $\times$ NVLink" (PXN) to avoid congestion in cases where the spine layer of the Rail-optimized network is oversubscribed [17].

We use the same variables defined in Table I in the following derivation. Consider a grid of $x \times y$ GPUs where $x$ GPUs are placed in an HB domain, and a Rail-only or Rail-optimized network connects $y$ HB domains. Recall that the HB domain has bandwidth $C_F$ while the NIC domain has bandwidth $C_S$ per GPU pair. For a Rail-optimized network, every GPU sends traffic directly to its destinations through the HB and full-bisection NIC domains. Assuming each pair of GPU communicates traffic of size $D$, the total all-to-all completion time is:

$$T_{a2a}^{Rail-opt} = \max\left(\frac{(x-1)D}{C_F}, \frac{x(y-1)D}{C_S}\right) = \frac{x(y-1)D}{C_S} \tag{2}$$

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 each rail, each GPU runs a second all-to-all *in the HB domain* to finish the transfers with an effective shard size of $xD$. Note that the second step here contains the *bandwidth-tax*. The total transmission time is

$$T_{a2a}^{Rail-only} = \frac{y(x-1)D}{C_F} + \frac{x(y-1)D}{C_S} \tag{3}$$

The two terms differ by $y(x-1)D/C_F$, which is the cost of forwarding within HB domains. When $y(x-1) \approx x(y-1)$, the slow-down factor is approximately $C_S/C_F$, which equals to 8.2% and 11.2% for the DGX A100 and DGX H100 generations of GPU platforms, respectively. This factor is low because of the bandwidth asymmetry between the two domains. Furthermore, this slow-down only applies to the all-to-all communication, which accounts for 27% of the total traffic as shown in Figure 5. Therefore, this small overhead is negli-

gible by Amdah's law. We note that such properties also make our network design suitable for other classes of DNN models.

### C. Fault Tolerance Properties of Rail-only Networks

Fault tolerance is crucial for large GPU clusters with long-lasting LLM training jobs. This section investigates the fault tolerance properties of Rail-only networks compared to traditional Rail-optimized networks.

**Link and switch failures.** Suppose a rail switch or a link fails. GPUs connected to the failed switch or link become unavailable for both Rail-optimized and Rail-only network architectures, rendering the two designs identical regarding fault tolerance on switches and links. However, our design requires fewer switches, which naturally reduces the points of failure. Datacenter operators can add redundant capacity by including extra rail switches, and our design remains more cost-effective than the any-to-any network design.

**GPU platform failure.** For a GPU cluster composed of DGX-like servers, each server is an HB domain. When a server fails, the network operator migrates the task to another healthy server. The Rail-only connectivity remains the same for the new server. For a GB200-like cluster, a super-chip module is analogous to a server; thus, the failure mode is the same as a single GPU failure, which we will discuss next.

**Single GPU failures with idle GPU in the HB domain.** We discuss two distinct scenarios separately for single GPU failures. The first case is when another idle GPU presents the same HB domain as the failed one. In this case, a Rail-optimized network directly replaces the failed GPU with the idle one without breaking the HB domain integrity. One possible solution is to leverage optical reconfigurable switches for the Rail-only network. To improve robustness, we add a small number of optical switches between the GPU and rail switches to allow the dynamic reconfiguration of rails. When a GPU fails, the optical switch reconfigures to bring a healthy, idle GPU to replace the failed one. This approach is conceptually similar to the failure recovery mechanism of network designs that primarily uses optical-reconfigurable switches [13], [25], [26]. We leave an in-depth analysis of rail-only with optical switch to future work.

**Single GPU failure in fully occupied HB domains.** Another failure mode occurs when a GPU fails in a fully occupied HB domain and requires a substitute GPU from different HB domains. In this case, the Rail-only design prevents migrating the task on the failed GPU to another idle one in the cluster, which is possible in a Rail-optimized network. However, such a solution is undesirable even with the Rail-optimized network. The new GPU no longer belongs to the same HB domain as the failed one, creating a bottleneck that slows the HB domain into a NIC domain. Instead, we propose two solutions. For platforms with small HB domains, we migrate the tasks on the entire HB domain with the failed GPU to a new one. For larger HB domains (e.g., DGX GH200 supercomputers with $K = 256$), these HB domains comprise a multi-tiered topology with an optical core-layer [16]. One potential approach is to add optical switches, like in the previous failure case. When a GPU failure occurs, the optical switch reconfigures, replacing a small set of GPUs (including the failed one) with healthy ones, thus maintaining the integrity of an HB domain.

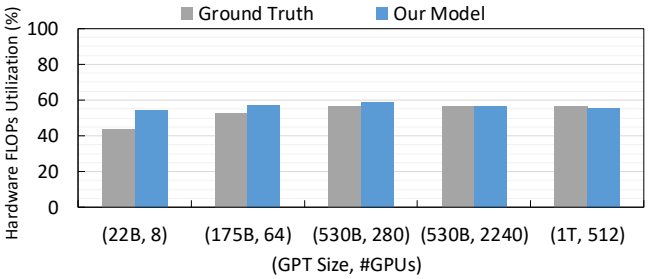



Fig. 7. HFU comparison between the ground truth [28] and our formulation.

## V. Evaluation

### A. Iteration Time Modeling

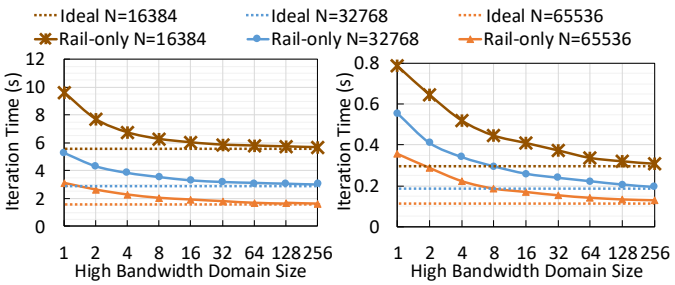
We evaluate our Rail-only network design's performance through an analytical model of the training iteration time. Our analytical model considers the critical path for LLM training with TP, DP, and PP, similar to the approach in Calculon [27].

We demonstrate the accuracy of our analytical model by comparing its estimation of hardware FLOPs utilization (HFU) to that of published results in the literature. The HFU refers to the hardware's floating point operation performed in an iteration over the peak floating point operations. Prior work provided the full set of hyperparameters in their evaluation setup, enabling us to compare the estimated HFUs from our analytical model to the ground truth [28]. In our evaluations, we assume a cluster of 1 to 280 DGX A100 servers. To compute the total required FLOPs for training per iteration of a DNN model, we use the same formula proposed by Korthikanti et al. [28].

Figure 7 compares the HFUs approximated by our analytical model with the ground truth for different GPT models and cluster scales. For GPT-1T, our computed HFU only differs from the ground truth by $1.8\%$. The discrepancy between our analytical model and the ground truth comes from our idealistic modeling of GPU computation and communication, assumptions on how computation and communication overlap, and underestimation of memory overhead. Such discrepancy goes up as the model size decreases. For the GPT-22B model, our estimation error is $10.8\%$ compared to ground truth MFU. The rest of this section utilizes our analytical model to estimate the training iteration time of Rail-only interconnects.

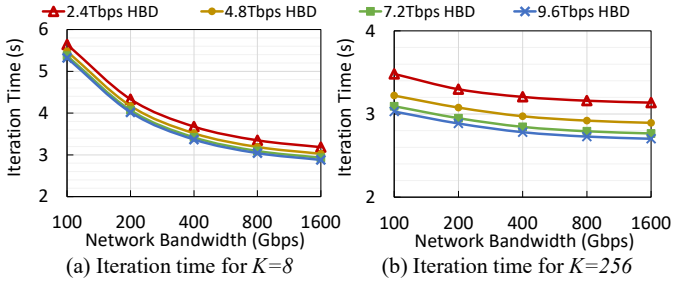
### B. What is the Optimal Size of an HB domain?

Intuitively, increasing HB domain size reduces the inter-platform network overhead during training. This section answers the following question: *what is the optimal size of an HB domain in a Rail-only interconnect?* In Figure 8, we vary the HB domain size ($K$) and plot the training iteration time for GPT-1T and GPT-146B MegatronLMs for clusters with 16384, 32768, and 65536 H100 GPUs. The global



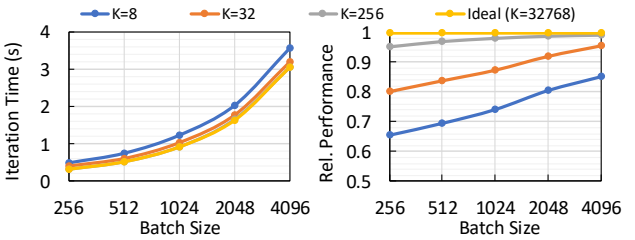

(a) GPT-1T (b) GPT-146B

Fig. 8. Iteration time as HB domain size changes.




(a) Iteration time for *K=8* (b) Iteration time for *K=256*

Fig. 9. Iteration time of GPT-1T as HB domain bandwidth and network bandwidth varies for different HB domain sizes.




(a) Iteration time (b) Relative performance to Ideal

Fig. 10. Iteration time and relative performance to the ideal case, as batch size changes, for GPT-1T.

batch size in this evaluation 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 found in our analytical model, using the bandwidth and computational ability parameters of DGX GH200. In addition, to capture the ideal iteration time, we also compute training iteration time where a full-bisection monolithic NVLink fabric connects every GPU (i.e., the case where $K = N$, where $N$ is the total GPU count).

As depicted in Figure 8, iteration times decrease as the HB domain size increases, indicating that larger HB domains reduce the network overhead during training. In both GPT models, the iteration time achieved with an HB domain size of 256 is nearly optimal. 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 the HB domain size increases. For the larger GPT-1T model, the train iteration time plateaus above 32 GPUs due to Amdhahl's law, suggesting diminishing returns from the increased cost of bigger HB domains.

For a smaller GPT-146B model, shown in Figure 8b, the marginal gain of increasing HB domain size is higher than that of GPT-1T. Providing an HB domain of size eight reduces the iteration time by $43.3\%$ compared to the HB domain of size one, while increasing the HB domain size from 8 to 256 further achieves a $30.6\%$ reduction in iteration time. The more significant marginal gain for smaller LLMs incurs more communication overhead when distributed to the same cluster than larger models. This effect arises from how computation and communication costs scale as LLM grows. The communication requirement increases linearly with the model's hidden size and sequence length. On the other hand, the model FLOPs increase quadratically with these two parameters, as indicated by previous work [21]. Therefore, we conclude that the optimal HB domain size depends on the size of the GPT model.

### C. Impact of HB domain and Network Bandwidth

The available bandwidth of HB and NIC domains determines the communication time during training. We analyze the impact of these bandwidths on the iteration time of a GPT model with one trillion parameters. Figure 9a and 9b show the iteration time variation for different HB domain bandwidths (different lines) and network bandwidths in the rails (on the $x$-axis) for HB domain size $K = 8$ and 256, respectively. As expected, the iteration time decreases when either bandwidth increases. However, the $K = 8$ case is less sensitive to the HB domain bandwidth. Increasing per-GPU bandwidth by a factor of four (from 2.4 Tbps to 9.6 Tbps) only improves the iteration time by $8.0\%$ on average for $K = 8$, compared to the improvement of $13.3\%$ for $K = 256$. On the other hand, larger HB domain sizes are more pronounced on network bandwidth improvement. Increasing the bandwidth from 100 Gbps to 400 Gbps, results in $35.9\%$ improvement for $K = 8$ but only $8.0\%$ for $K = 256$. Hence, improving the HB domain bandwidth is more beneficial than the network bandwidth for LLMs as future HB domain size increases.

### D. Impact of Batch Size on Network Design

While the batch size is typically an ML-centric metric for optimization, our analysis indicates that the impact of batch size on the total training time goes beyond the total number of iterations required for convergence. To further understand such impact, we analyze the iteration time of a GPT-1T model on a 32768 GPU cluster while changing the HB domain size from $K = 8$ to 32768. In this study, we vary the global batch size from 256 to 4096. Figure 10a plots the change in iteration time as the batch size varies. The iteration time exhibits a similar trajectory for all HB domain sizes; however, the *relative performance* (the ratio of the iteration time for an HB domain size to that of the ideal case) improves as the batch size increases. Figure 10b represents this trend. When $K = 256$, the relative performance increases from $95\%$ to $99\%$ as the batch size increases from 256 to 4096 sequences. The

iteration time advantage is prominent when the HB domain size is small. For $K = 8$, increasing the batch size from 256 to 4096 improves the relative performance from $65\%$ to $85\%$, suggesting a larger batch size is preferable for a cluster with a smaller HB domain. Prior studies have shown that LLM training benefits from a larger batch size [1], [29], especially for bigger models, making it a perfect fit for our Rail-only design.

### *E. Network Cost and Power Analysis*

Our Rail-only network architecture judiciously reduces the network resources for LLM training by eliminating unused network hardware. This section compares our proposed approach's network cost and power with the state-of-the-art Rail-optimized GPU clusters. We calculate the number of switches ($N_{SW}$) and transceivers ($N_{TR}$) required for each network design and derive the network equipment cost and peak power consumption based on numbers reported in prior work and by vendors [13], [30], [31].[1] We enumerate the number of switches and transceivers required to build the state-of-the-art network architecture and our proposed architecture in Table II, accounting for variable cluster sizes and network switch radix. For each architecture, we build a minimum layer Clos network with the provided switch radix and calculate the minimum number of required switches and transceivers to achieve the desired connectivity. The total cost of each architecture is

$$\text{Total cost} = Price_{SW} \times N_{SW} + Price_{TR} \times N_{TR} \quad (4)$$

while the power is

$$\text{Total power} = Pwr_{SW} \times N_{SW} + Pwr_{TR} \times N_{TR} \quad (5)$$

The last two columns of Table II provide the cost and power savings of a Rail-only interconnect over the state-of-the-art. Our Rail-only design notably reduces the network cost by 38% to 77% (117 to 234 million dollars) and power consumption by 37% to 75% (1.7 to 6.9 megawatts) compared to the state-of-the-art design while achieving equivalent performance. This reduction stems from eliminating spine layer switches and decreasing the number of switch tiers within each rail. Surprisingly, switches with a radix of 64 provide the worst-case cost and power reduction in both cluster sizes. In this case, the state-of-the-art design requires a three-tier Clos network, while the Rail-only design requires two tiers for each rail. Still, our design only requires three-quarters of the total number of switches while achieving the same performance as the state-of-the-art design.

TABLE II
NUMBER OF SWITCHES AND TRANSCEIVERS FOR DIFFERENT CLUSTERS.

| #GPUs | Switch Radix | #Switches ($N_{SW}$) | | #Transceivers ($N_{TR}$) | | Savings | |
|---|---|---|---|---|---|---|---|
| | | SOTA | Rail-only | SOTA | Rail-only | Cost | Pwr |
| 32768 | 64 | 2560 | 1536 | 196608 | 131072 | 38% | 37% |
| | 128 | 1280 | 256 | 196608 | 65536 | 77% | 75% |
| | 256 | 384 | 128 | 131072 | 65536 | 62% | 60% |
| 65536 | 64 | 5120 | 3072 | 393216 | 262144 | 38% | 37% |
| | 128 | 2560 | 1536 | 393216 | 262144 | 38% | 37% |
| | 256 | 1280 | 256 | 393216 | 131072 | 77% | 75% |

## VI. RELATED WORK

**LLM trend.** The current growth rate of LLM computational and speed requirement outpaces the advancements in AI accelerators and network speed as Moore's law slows down, necessitating hyper-scale clusters and more efficient interconnects [32], [33]. The MegatornLM line of work pioneers LLM parallelization [21], [28], [34]. Our position to remove any-to-any network connectivity complements MegatronLM. We also acknowledge ongoing efforts to reduce language models' size and resource requirements without compromising performance [35]. These works complement our design as our design reduces network resources and maintains performance even for smaller language models and clusters. Similarly, research directions that aim to directly reduce the amount of communication through quantization and compression, like DeepSpeed Zero++, also complement our approach [36].

**LLM Inference.** This paper explores the training workload of LLMs, yet inference represents another significant part of the LLM product cycle. Inference demands fewer resources as it involves moving less data through the LLM and only computes the forward pass and multiple passes to generate response tokens [37]. Pope et al. developed specific parallelism for inference on TPU-v4 architecture [38]. For our design, each HB domain becomes an inference-serving domain with low latency, and the Rail-only connections help load-balance multiple inference domains. We leave a detailed investigation of LLM inference to future work.

**Multi-job training.** It is common for a GPU cluster to train multiple smaller jobs simultaneously. Existing works focus on Clos-based GPU clusters and provide techniques for better fairness and shorter job-completion time [39]–[42]. While this paper focuses on training a single LLM on a large cluster, the Rail-only network design is also suitable for a multi-job setting. The entire cluster can be arbitrarily partitioned by tiling into smaller rectangular partitions, similar to the case of TPU-v4 [25]. Each partition then independently executes a smaller training job.

**ML infrastructures and other ML workloads.** Prior works illustrated the benefits of co-designing hardware and software for ML models. For instance, Google's TPU cluster is optimized for training large models with 3D parallelism on TPUs [25], while Meta's Neo focuses on training recommendation models with large embedding tables [43]. Our work focuses on designing a cost-efficient network to train LLMs efficiently. Although our proposed Rail-only architecture focuses on network design specifically for LLMs, our design is efficient for many other DNN workloads when combined with other efforts, as the forwarding overhead is low (§IV-B). Recent works attempt to make the parallelization strategy and collective communication algorithms bandwidth-aware for any DNN model [44]–[46], producing traffic patterns ideal for the Rail-only network.

[1]Cost: $Price_{TR}$ = \$199 per transceiver, $Price_{SW}$ = \$694 per switch port for 400 Gbps; Power: $Power_{TR}$ = 9W per transceiver, $Power_{SW}$ = 18W per switch port

## VII. Conclusion

In this paper, we examine and analyze the traffic pattern of LLM training with hybrid parallelism. We propose a novel *Rail-only architecture* that aligns with LLMs' distinct characteristics and demands. Our architecture 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

We thank anonymous Hot Interconnects reviewers for their feedback. The MIT-affiliated authors are supported by DARPA FastNICs 4202290027, NSF SHF-2107244, NSF CAREER-2144766, NSF PPoSS-2217099, NSF CNS-2211382, NSF FuSe-TG-2235466, Sloan fellowship FG-2022-18504, ACE and CUbiC centers sponsored by Semiconductor Research Corporation (SRC) and DARPA under the JUMP 2.0 program.

## References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] L. Labs, "Openai's gpt-3 language model: A technical overview," 2020. [Online]. Available: https://lambdalabs.com/blog/demystifying-gpt-3

[3] OpenAI, "Gpt-4 technical report," 2023.

[4] The-Decoder, "Gpt-4 has a trillion parameters - report," 2023. [Online]. Available: https://the-decoder.com/gpt-4-has-a-trillion-parameters/

[5] Nvidia, "Nvlink and nvswitch: The building blocks of advanced multi-gpu communication—within and between servers." 2023. [Online]. Available: https://www.nvidia.com/en-us/data-center/nvlink/

[6] AMD, "Amd instinct mi300x platform," 2023. [Online]. Available: https://www.amd.com/en/products/accelerators/instinct/mi300/platform.html

[7] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 63–74. [Online]. Available: https://doi.org/10.1145/1402958.1402967

[8] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 202–215. [Online]. Available: https://doi.org/10.1145/2934872.2934908

[9] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl, A. Bhagat, G. Bhaskara, T. Brokhman, L. Cao, A. Cheema, R. Chow, J. Cohen, M. Elhaddad, V. Ette, I. Figlin, D. Firestone, M. George, I. German, L. Ghai, E. Green, A. Greenberg, M. Gupta, R. Haagens, M. Hendel, R. Howlader, N. John, J. Johnstone, T. Jolly, G. Kramer, D. Kruse, A. Kumar, E. Lan, I. Lee, A. Levy, M. Lipshteyn, X. Liu, C. Liu, G. Lu, Y. Lu, X. Lu, V. Makhervaks, U. Malashanka, D. A. Maltz, I. Marinos, R. Mehta, S. Murthi, A. Namdhari, A. Ogus, J. Padhye, M. Pandya, D. Phillips, A. Power, S. Puri, S. Raindel, J. Rhee, A. Russo, M. Sah, A. Sheriff, C. Sparacino, A. Srivastava, W. Sun, N. Swanson, F. Tian, L. Tomczyk, V. Vadlamuri, A. Wolman, Y. Xie, J. Yom, L. Yuan, Y. Zhang, and B. Zill, "Empowering azure storage with RDMA," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 49–67. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/bai

[10] T. Schneider, O. Bibartiu, and T. Hoefler, "Ensuring deadlock-freedom in low-diameter infiniband networks," in *2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI)*, 2016, pp. 1–8.

[11] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 779–805. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/goyal

[12] S. Hu, Y. Zhu, P. Cheng, C. Guo, K. Tan, J. Padhye, and K. Chen, "Deadlocks in datacenter networks: Why do they form, and how to avoid them," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 92–98. [Online]. Available: https://doi.org/10.1145/3005745.3005760

[13] W. Wang, M. Khazraee, Z. Zhong, M. Ghobadi, Z. Jia, D. Mudigere, Y. Zhang, and A. 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)*. Boston, MA: USENIX Association, Apr. 2023, pp. 739–767. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/wang-weiyang

[14] NVIDIA, "Dgx h100 computer," 2023. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-h100/

[15] ——, "Gb200 nvl72 computer," 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/gb200-nvl72/

[16] Nvidia, "Nvidia dgx gh200," 2023. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-gh200/

[17] ——, "Doubling all2all performance with nvidia collective communication library 2.12," 2022. [Online]. Available: https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/

[18] Meta, "Introducing data center fabric, the next-generation facebook data center network," 2014. [Online]. Available: https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[19] Nvidia, "Nvidia dgx superpod: Next generation scalable infrastructure for ai leadership, reference architecture," 2023. [Online]. Available: https://docs.nvidia.com/dgx-superpod-reference-architecture-with-dgx-h100-systems.pdf

[20] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits, "Using multirail networks in high-performance clusters," in *Proceedings 2001 IEEE International Conference on Cluster Computing*, 2001, pp. 15–24.

[21] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," 2021.

[22] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," 2022.

[23] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 267–280. [Online]. Available: https://doi.org/10.1145/3098822.3098838

[24] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/mellette

[25] N. P. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles, C. Young, X. Zhou, Z. Zhou, and D. Patterson, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," 2023.

[26] L. Poutievski, O. Mashayekhi, J. Ong, A. Singh, M. Tariq, R. Wang, J. Zhang, V. Beauregard, P. Conner, S. Gribble, R. Kapoor, S. Kratzer, N. Li, H. Liu, K. Nagaraj, J. Ornstein, S. Sawhney, R. Urata, L. Vicisano, K. Yasumura, S. Zhang, J. Zhou, and A. Vahdat, "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 66–85. [Online]. Available: https://doi.org/10.1145/3544216.3544265

[27] M. Isaev, N. Mcdonald, L. Dennison, and R. Vuduc, "Calculon: a methodology and tool for high-level co-design of systems and large language models," in *Proceedings of the International Conference for*

*High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607102

[28] V. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," 2022.

[29] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," 2020.

[30] Nvidia, "Nvidia docs hub: Mma4z00-ns400 400gb/s single-port osfp 400gb/s multimode sr4 50m connectivity scenarios," 2023. [Online]. Available: https://docs.nvidia.com/networking/display/mma4z00ns400/connectivity+scenarios#:~:text=The%20400Gb%2Fs%20transceiver%20has,maximum%20or%208%20Watts%20typical.

[31] ——, "Nvidia docs hub: Qm9700/qm9790 1u ndr 400gb/s infiniband switch systems user manual specifications," 2023. [Online]. Available: https://docs.nvidia.com/networking/display/qm97x0pub/specifications

[32] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 782–797. [Online]. Available: https://doi.org/10.1145/3387514.3406221

[33] OpenAI, "Openai: Ai and compute," 2023. [Online]. Available: https://openai.com/research/ai-and-compute

[34] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," 2020.

[35] Databricks, "Hello dolly: Democratizing the magic of chatgpt with open models," 2023. [Online]. Available: https://www.databricks.com/blog/2023/03/24/hello-dolly-democratizing-magic-chatgpt-open-models.html

[36] Microsoft, "Deepspeed zero++: A leap in speed for llm and chat model training with 4x less communication," 2023. [Online]. Available: https://www.microsoft.com/en-us/research/blog/deepspeed-zero-a-leap-in-speed-for-llm-and-chat-model-training-with-4x-less-communication/

[37] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez, and I. Stoica, "AlpaServe: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

[38] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, A. Levskaya, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," 2022.

[39] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 595–610. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/xiao

[40] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU cluster manager for distributed deep learning," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 485–500. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/gu

[41] Y. Zhao, Y. Liu, Y. Peng, Y. Zhu, X. Liu, and X. Jin, "Multi-resource interleaving for deep learning training," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 428–440. [Online]. Available: https://doi.org/10.1145/3544216.3544224

[42] S. Rajasekaran, M. Ghobadi, and A. Akella, "Cassini: Network-aware job scheduling in machine learning clusters," 2023.

[43] D. Mudigere, Y. Hao, J. Huang, Z. Jia, A. Tulloch, S. Sridharan, X. Liu, M. Ozdal, J. Nie, J. Park, L. Luo, J. A. Yang, L. Gao, D. Ivchenko, A. Basant, Y. Hu, J. Yang, E. K. Ardestani, X. Wang, R. Komuravelli, C.-H. Chu, S. Yilmaz, H. Li, J. Qian, Z. Feng, Y. Ma, J. Yang, E. Wen, H. Li, L. Yang, C. Sun, W. Zhao, D. Melts, K. Dhulipala, K. Kishore, T. Graf, A. Eisenman, K. K. Matam, A. Gangidi, G. J. Chen, M. Krishnan, A. Nayak, K. Nair, B. Muthiah, M. khorashadi, P. Bhattacharya, P. Lapukhov, M. Naumov, A. Mathews, L. Qiao, M. Smelyanskiy, B. Jia, and V. Rao, "Software-hardware co-design for fast and scalable training of deep learning recommendation models," 2023.

[44] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, J. E. Gonzalez, and I. Stoica, "Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 559–578. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[45] C. Unger, Z. Jia, W. Wu, S. Lin, M. Baines, C. E. Q. Narvaez, V. Ramakrishnaiah, N. Prajapati, P. McCormick, J. Mohd-Yusof, X. Luo, D. Mudigere, J. Park, M. Smelyanskiy, and A. Aiken, "Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 267–284. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/unger

[46] L. Zhao and A. Krishnamurthy, "Bandwidth optimal pipeline schedule for collective communication," 2023.

## APPENDIX

An accurate and detailed model provides fundamental guidance for choosing the right parallelization strategy and network design. Previous research has mathematically modeled the computational time and memory requirements for various LLM training parallelization strategies [21], [28], [34]. Nevertheless, these works omit a detailed derivation for communication time during a training iteration considering the network infrastructure. This section presents an analytical model of the training iteration time, incorporating both the parallelization strategy and the training infrastructure. This formulation is the foundation for evaluating the Rail-only network design in Section V. Table III outlines the parameters used in this analysis. The section assumes mixed-precision training with 16-bit model parameters, activations, and gradients.

### A. Critical Path Analysis

Figure 11 displays the 1F1B pipeline schedule without interleaving. Given the uniform structure of LLMs under the PTD-P parallelism, both forward and backward execution times for each micro-batch across GPUs remain the same. This consistency allows us to identify a critical path in a training iteration, highlighted by a red arrow in Figure 11. This path further decomposes into three parts: the pipeline bubble time, the computation and communication time in the last stage (LS) of the pipeline, and finally, the parameter synchronization time after the pipeline flush. Note that the bubbling and the last pipeline stage are strictly disjointed. However, parameter synchronization traffic can start immediately for each transformer layer after the last micro-batch finishes processing, overlapping itself with the backward propagation. Another potential overlapping happens within the last stage, between the PP and TP traffic across different micro-batches. For simplicity, we provide conservative modeling of the iteration time, in which we start parameter synchronization after all computation finishes and disallow cross micro-batch TP and PP traffic overlapping.

TABLE III
PARAMETERS UTILIZED IN THE CALCULATION IN THIS SECTION.

| Name | Description |
|---|---|
| $p, t, d$ | Pipeline, Tensor and Data parallelism dimensions, respectively |
| $p_h, t_h, d_h$ | The portion $p, t, d$ mapped into an HB domain, respectively |
| $p_l, t_l, d_l$ | The portion $p, t, d$ mapped into the network domain, respectively |
| $h$ | LLM Embedding dimension (hidden size) |
| $s$ | Sequence length |
| $v$ | Number of interleaved stages |
| $K$ | HB domain size |
| $l$ | Number of transformer block layers |
| $C_F$ | HB domain bandwidth |
| $C_S$ | GPU Network bandwidth |
| $S_T$ | Number of parameters in a transformer block |
| $b$ | Micro-batch size per pipeline |
| $m$ | Number of micro-batches per iteration |

With these observations, the iteration time is

$$T_{iter} = T_{bubble} + T_{LS} + T_{sync} \tag{6}$$

This model also holds for the interleaved schedule. Interleaved schedules reduce the pipeline bubble time while requiring additional communication within the last stage of the pipeline. We factor such cost into $T_{bubble}$ and $T_{LS}$ in Eq. 6. The rest of this section dissects each term with explicit computation and communication cost.

### B. Analytical Iteration Time Modeling

This section provides a quantitative analysis, considering each term's computation and communication cost in Eq. 6.

**Pipeline bubble time**. For the pipeline bubble time, we break down the cost into the communication and the computation as $T_{bubble} = T^{comp}_{bubble} + T^{comm}_{bubble}$. Assume a micro-batch size's total compute time (forward and backward pass) is $t(b)$. With interleaved scheduling of $v$, the computation time spent in the pipeline is

$$T^{comp}_{bubble} = \frac{(p-1)t(b)}{v} \tag{7}$$

Narayanan et al. observed that the computational efficiency of GPU varies with $b$ [21]; therefore, it is best to profile $t(b)$ for each micro-batch size in practice. For simplicity, we provide an alternative measurement analytically by modeling the computational requirement (FLOPs) of an LLM and GPU's computation speed in Appendix **??**.

For the communication, each micro-batch induces $D^p_{\mu b} = 2bhs/t$ bytes of traffic across two pipeline stages when sequence parallelism is enabled together with TP. Such transfer will happen for a total of $2(p-1)$ times throughout the pipeline filling and emptying, where $2(p_s-1)$ times will happen in the network domain and $2p_s(p_f-1)$ times in HB domains. Hence, the pipeline bubble communication time is

$$T^{comm}_{bubble} = \frac{2(p_s-1)D^p_{\mu b}}{C_S} + \frac{2p_s(p_f-1)D^p_{\mu b}}{C_F} \tag{8}$$

Unlike the computation time, the communication time for bubbling is unaffected by the interleaved scheduling.

**Last stage time.** Similar to the bubble time, we analyze the computation and computation costs separately in the last stage of the pipeline. The last stage has $m$ micro-batches going through it, and therefore the computational cost is

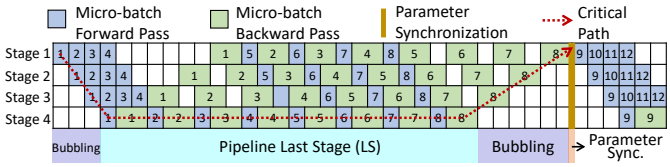



Fig. 11. 1F1B pipeline schedule and its critical path.

$$T^{comp}_{LS} = mt(b) \tag{9}$$

For communication, TP and PP both generate network traffic across GPUs. We first focus on the TP traffic: for each micro-batch, four AllGather and four ReduceScatter happen in total for each layer of the transformer layer when sequence parallelism is applied. Since ReduceScatter is the inverse of AllGather and generates the same amount of traffic, this analysis will focus on AllGather only. Recall $\texttt{AGtime}(D, x, y, C_F, C_S)$ (Eq. 1) is the time of running AllGather on data of size $D$ bytes on an $x \times y$ grid of GPUs with HB domain and network bandwidth $C_F$, $C_S$ respectively. The amount of data to run AllGather for each micro-batch is $D^t_{\mu b} = 2bhs$ per transformer block, and since each pipeline stage holds $l/p$ transformer blocks, the total runtime for all $m$ micro-batches is $8lm\texttt{AGtime}(D^t_{\mu b}, t_h, t_l, C_F, C_S)/p$.

Next, we look at the pipeline traffic in the last stage. The pipeline traffic can overlap with the computation, even with interleaved scheduling; however, with conservative modeling, we assume that GPUs do not perform computation when sending and receiving pipeline parallel traffic. Each interleaved part of each micro-batch at least sends or receives $D^p_{\mu b}$ bytes of traffic both on the forward and backward pass and every micro-batch needs to traverse the network domain whenever $p_s > 1$. Therefore, we model the pipeline communication overhead in the last stage as $2mvD^p_{\mu b}/C_*$ where $C_* = C_S$ if $p_s > 1$, else $C_* = C_F$. Adding the tensor and pipeline communication time together,

$$T^{comm}_{LS} = \frac{8lm\texttt{AGtime}(D^t_{\mu b}, t_h, t_l, C_F, C_S)}{p} + \frac{2mvD^p_{\mu b}}{C_*} \tag{10}$$

**Parameter Synchronization.** Finally, we have the parameter synchronization time, consisting of an AllReduce operation of the model parameters in the first stage of the pipeline. We only model the communication time of the AllReduce operation since the computation of AllReduce incurs minimal overhead. We follow the same hierarchical collective algorithm described in Section IV. For a $d = d_h \times d_l$ way DP, the amount of data to AllReduce is $D^d = 2lS_T/pt$. An AllReduce induces twice the runtime as an AllGather for the same amount of data; therefore,

$$T_{sync} = 2\texttt{AGtime}(D^d, d_h, d_l, C_F, C_S) \tag{11}$$

Together, Eq. 8, 7, 9, 10 and 11 provide a closed-form expression for Eq. 6 as the training iteration time for an

LLM. While this section only presents the transmission delay, in our evaluation, we also consider the propagation delay (network latency) for the communication times to get the best iteration time accuracy.

### C. Constraints in Choosing Parameters

In addition to the cost model, we derive the set of constraints of the hyperparameters that describe a complete parallelization strategy. We derive a program that exhaustively generates all the possible parallelization configurations in Appendix **??**. The iteration time model from Eq. 6 then serves as the cost of an optimization problem to derive the optimal parallelization strategy. In the next section, we evaluate the accuracy of this modeling and use it as the basis for analyzing our Rail-only network design for training LLMs.

### I. Bandwidth Time of AllGather Algorithm for Rail-Optimized and Rail-Only Network

Zhao et al. [46] proved that a bandwidth-optimal AllGather schedule exists for arbitrary network topology. This section examines the best-case AllGather time for a grid of $x \times y$ GPUs in the rail-optimized and rail-only network. Following Eq. 1 in Zhao et al. [46], we derive an analytical expression of the bandwidth AllGather time for these two types of networks.

Let $\mathbb{M}_{xy}$ be the space of all boolean matrices of size $x \times y$ except all ones and all zeros. A matrix $A \in \mathbb{M}_{xy}$ represents a specific partition of the GPUs in an $x \times y$ configuration. Then, the optimal AllGather time for one unit of data in a rail-optimized network is

$$\max_{A \in \mathbb{M}_{xy}} \frac{\max\limits_{A' \in \{A, \bar{A}\}} \sum\limits_{i,j} A'_{ij}}{\min\limits_{A'' \in \{A, \bar{A}\}} ((C_F + C_S) \sum\limits_{i,j} A''_{ij} - xR(A'')C_F)} \tag{12}$$

Here, $\bar{A}$ denotes the negation of the boolean matrix $A$. The numerator finds the partition that sends the largest data, which equals the sum of the binary entries of $A$ or $\bar{A}$. The denominator finds the partition with the lowest ingress and egress bandwidth. For each GPU included in the partition, the total bandwidth of the partition increases by $C_F + C_S$, hence the first term in the minimization. However, whenever the partition includes an entire row of GPUs (i.e., an HB domain), the bandwidth internal to this HB domain no longer contributes to the ingress or egress bandwidth. The function $R(A)$ counts the number of rows with all ones as $A$'s entries, implying one HB domain entirely inside the partition. The second term in the minimization removes this part of the bandwidth from the egress bandwidth.

For the rail-only network, going through the same analysis, we get the AllGather time of

$$\max_{A \in \mathbb{M}_{xy}} \frac{\max\limits_{A' \in \{A, \bar{A}\}} \sum\limits_{i,j} A'_{ij}}{\min\limits_{A'' \in \{A, \bar{A}\}} ((C_F + C_S) \sum\limits_{i,j} A''_{ij} - xR(A'')C_F - yC(A'')C_S)} \tag{13}$$

The formula remains the same as Eq. 12, except for the last term in the denominator. This term accounts for the fact that whenever an entire rail is included in the partition, this rail no longer contributes its bandwidth as the ingress or egress bandwidth of the partition. Here, the $C(A)$ function counts the number of columns with all ones as their entries, hence the number of entire rails inside the partition.

Intuitively, to maximize either expression, the choice of $A$ should be skewed (i.e., having a large portion of GPU in one partition) so that $\sum\limits_{i,j} A'_{ij}$ on the numerator is large but $\sum\limits_{i,j} A''_{ij}$ on the denominator is small. In addition, the GPU choice should be arranged such that the denominator can exclude as much bandwidth from the partition as possible. For Eq. 12 and Eq. 13, one such configuration is obtained when the partition has $y - 1$ HB domains. In this case. $R(A'') = 1$ and $C(A'') = 0$ which yield an AllGather time of $(y - 1)/C_S$ per unit of data for both networks. We postulate that for $C_F >> C_S$ and $x \geq y$, this choice of partition yields the lower bound (thus bandwidth optimal) AllGather time for both of this network, as perturbing the partition by adding or removing single GPUs or entire HB domains only relaxes the bound. We leave concrete proof of the optimality in future work.