Datacenter Topologies for Communication-efficient LLM Training

Anonymous Author(s)

Abstract

Modern *Large Language Model*s (LLMs) are trained on large clusters of computational nodes (e.g., GPUs), requiring frequent and costly communication. In practice, these clusters are deployed in datacenters whose network interconnect follows a particular topology, which determines tradeoffs among communication overhead, total link count, and the average shortest-path distance between nodes—the focus of this paper. We study common communication patterns in distributed LLM training and, leveraging these insights, compare how well different datacenter topologies support them. Specifically, we examine Fat-tree, Dragonfly+, and HyperX, and show that HyperX can achieve minimal hop counts for these communication patterns.

1 Introduction

Training large language models (LLMs) has become a major endeavor in recent years, involving massive computational workloads that are often distributed across hundreds to thousands of accelerators (e.g., GPUs/TPUs) [6, 15]. Table 1 summarizes the reported number of training accelerators for GPT-3, PaLM, and LLaMA, showcasing its actual scale.

Table 1: Reported accelerator counts for training selected large language models.

Model	Year	# Accelerators	Type
LLaMA (65B)	2023	2048 [15]	NVIDIA A100
PaLM (540B)	2022	6144 [6]	TPU v4
GPT-3 (175B)	2020	~1024 [3]	NVIDIA A100

Note: The original GPT-3 paper [4] does not report the number of GPUs/TPUs used for training; the ~1024 A100 figure reported in Table 1 is an estimate/assumption used in subsequent work [3].

At this scale, communication overhead becomes an inevitable bottleneck. During training, synchronization operations generate highly structured, high-volume traffic patterns between compute nodes [9, 10, 12, 13]. As a result, the efficiency of the interconnection topology directly affects model throughput, training time, and power consumption.

While compute capabilities of GPUs have scaled rapidly, network interconnects have not kept pace in terms of cost-efficiency and scalability. Consequently, designing or selecting an optimal network topology becomes a fundamental challenge in large-scale LLM deployments. The physical topology must be capable of delivering sufficient bandwidth to support the communication pattern implied by the training algorithm—captured by a *transport matrix*—while minimizing latency and infrastructure cost.

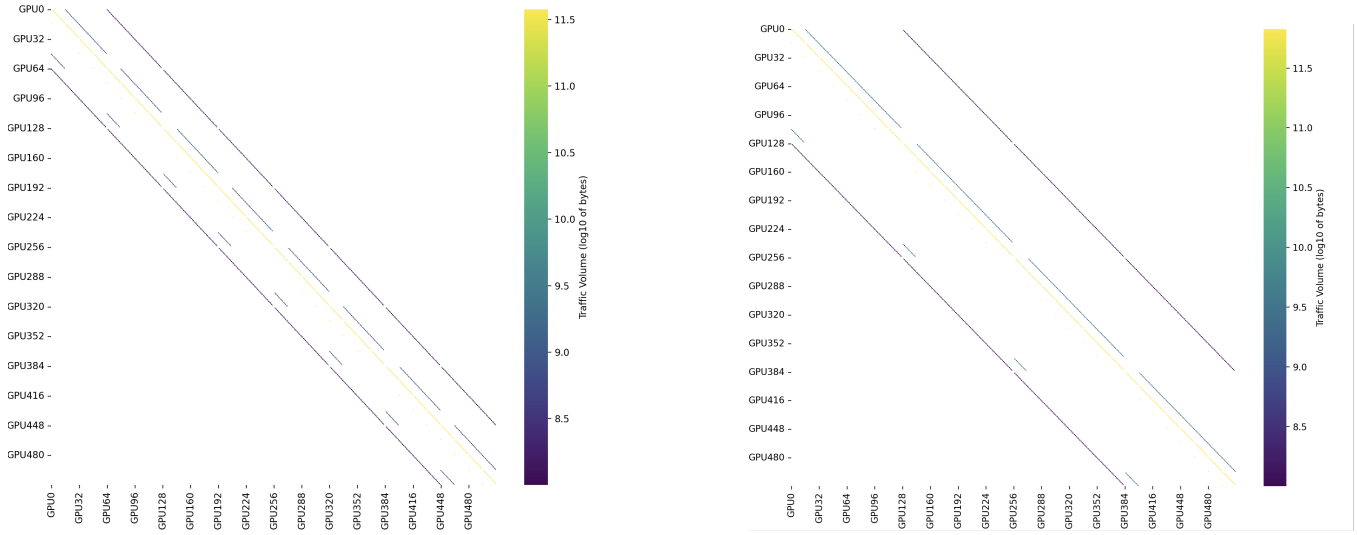
Following the large number of involved GPUs, to minimize training time and maximize hardware utilization, LLM training typically leverages a combination of three parallelization strategies. Large neural network layers are split across multiple GPUs. Since layer computations like matrix multiplications can be partitioned across the weight matrix, each GPU stores and computes a slice of the layer. After the local computation, GPUs must synchronize their partial results.

In LLM training, GPUs are often organized in a pipeline form with multiple stages with disjoint sets of GPUs. Stages often have similar numbers of GPUs such that within each stage, GPUs are organized as multiple groups of a fixed size. This structure, in turn, dictates the specific communication requirements, which vary during different steps of the training process. Communication is required (1) within each group of GPUs in the same stage, (2) between different groups in the same stage and (3) between GPUs in adjacent stages.

In this paper, we study the suitability of various datacenter topologies [1, 2, 8, 11, 14] to serve communication in LLM training. To do so, we dive into basic properties of the communication requirements. Our key contribution is showing that the communication patterns in LLM training implied by its pipeline parallelization map directly to the dimensions of a 3D direct-connect topology, making HyperX an ideal, hop-count optimal topology for this workload. This investigation is particularly timely given the growing scale and economic impact of LLM training infrastructure, and the need for principled, data-driven decisions in system architecture [5, 7].

2 Communication Properties for Real Workloads

In the first part of our study, we shed light on common communication patterns in LLM training. To do so, we simulate the communication in various models with variable number of parameters and a range of GPU counts. This understanding is



(a) PP=8 stages, each of 64 GPUs **(b) PP=4 stages, each of 128 GPUs**

Figure 1: Transport matrices for the pipeline configurations of 512 GPUs in eight stages (PP=8) and in four stages (PP=4). In both configurations, tensor parallelism group size is set as TP 16.

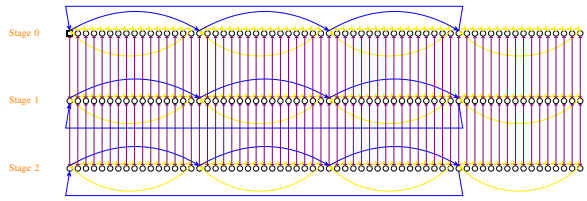


Figure 2: Communication within and across stages for 512 GPUs in eight stages (PP=8) with TP=16 (as in Figure 1(a)), based on the pipeline structure and TP group organization. The first $64 \cdot 3 \quad 192$ **GPUs, organized in three among the eight stages are shown. Arrows in purple indicate bidirectional communication. Arrows in blue are shown only for the first GPU in each TP group but leave all GPUs in the TP group.**

crucial towards examining the suitability of various datacenter topologies for efficient communication.

Setup. We generated synthetic training workloads using the Alibaba Cloud *AICB* tool, which serves as input to the large-scale simulator *SimAI* [17]. Each workload was characterized by four key parameters:

- **Model architecture** - E.g., GPT-3 or LLaMA, each associated with a number indicating the number of parameters in the model such as 7B, 65B and 405B;
- **Number of GPUs** - The total number of allocated GPUs;
- **Pipeline parallelism** (PP) - The number of pipeline stages in which GPUs are organized;
- **Tensor parallelism (TP) group size** - The size of a group of GPUs within each stage.

Jointly, these values imply the **number of model replicas** (DP) (namely, how many times the model is replicated across distinct data-parallel groups) as

$$DP \quad \frac{\text{Total number of GPUs}}{PP \text{ stages} \times TP \text{ group size}}.$$

All other training parameters, such as batch size and optimizer settings, were held fixed to isolate the effects of the parallelism configuration. Specifically, all workloads were trained for a single epoch with a micro-batch size of 1 and a sequence length of 4096 tokens, using a GPT-style tokenizer with a vocabulary size of 50,257 and maximum position embeddings of 4096. All models used FP16 precision (2 bytes per element) under the `Megatron-LM` framework, with no mixture-of-experts (MoE) layers (`expert_model_parallel_size`=1, `num_experts`=1). The global batch size was set to 2×world_size (e.g., 1024 for 512 GPUs), and for `LLaMA` models, `FlashAttention` and `SwiGLU` activations were enabled. Using these standardized workloads together with *SimAI*, we analyzed the form of communication among GPUs.

Transport Matrix. In a transport matrix D for a training process, each entry $D_{i,j}$ represents the total volume of data transferred from GPU i to GPU j during the course of training. To obtain realistic communication patterns, we examined

transport matrices under diverse configurations. These configurations varied in the total number of GPUs, the pipeline parallelism number of stages PP and the TP group size.

We observe in the transport matrices the following traffic patterns, ordered by decreasing order of their significance:

- Within TP-group: The most dominant communication is within each TP group, in a ring-based form between adjacent nodes. This refers to the operations AllGather, ReduceScatter and AllReduce.
- Within stage: Communication between adjacent TP groups within the same pipeline stage. This refers to handling activations or layer configurations.
- Cross stage: Communication between adjacent stages through corresponding GPUs. This also refers to activations forwarding, here between stages.

The amount of communication can differ among the three patterns by an order of magnitude, particularly between the TP pattern and the other two.

Consider two configurations of 512 GPUs in pipelines: (a) with PP=8 stages, each of 64 GPUs; (b) with PP=4 stages, each with 128 GPUs. For the two configurations, Figure 1 illustrates the transport matrices (of size 512×512). Group size is set as TP=16 in both matrices such that each stage of the model is split into groups of TP=16 GPUs. The matrices are shown on a logarithmic scale (with values of zero displayed in white). A row refers to a source GPU and a column to a destination GPU. There are similar but also different parts in the matrices. In both matrices, as shown in yellow, a high portion of the traffic is sent from one GPU to the following GPU (in each group of 16 GPUs in a cyclic manner). In addition, in (a) with PP=8 stages (each of 64 GPUs), we see a smaller but still significant amount of traffic sent between a GPU to the GPU with index +16 within the same stage, in a cyclic manner (namely modulo 64 which is the size of each stage). A slightly lower amount of traffic is sent to the GPU with index +64, namely to the corresponding GPU in the next stage. Similarly, there is also a similar amount of traffic to the corresponding GPU in the previous stage with index -64. On the other hand, in (b) with PP=4 stages (each of 128 GPUs), we also observe a significant amount of traffic sent between a GPU to GPU with index +16 within the same stage, in a cyclic manner (namely modulo 128 here). A slightly lower amount of traffic is sent to GPU with indices +128 and -128, namely to the corresponding GPU in the next stage and previous stages.

Figure 2 further illustrates the communication pattern in the pipeline structure for the case of PP=8 stages of 64 GPUs each, again with TP=16 as for which the transport matrix was shown in Figure 1(a). The figure refers to the first three among the eight stages with $64 \cdot 3$ 192 GPUs. Within each stage, there are four groups of TP=16 consecutive GPUs. The figure uses similar colors to the transport matrix, illustrating communication within a TP group in a ring form in yellow, within the same stage between corresponding nodes in adjacent TP groups in blue, and in violet, the lighter communication between corresponding nodes in the adjacent stages. For simplicity only part of the blue arrows are shown.

3 Mapping GPUs into Topologies based on Transport Matrices

The communication requirements of a transport matrix relies on a datacenter which is organized in various topologies. Designing such service efficiently has two main steps: (1) Setting the topology; (2) Mapping the various GPUs to the topology. In doing so, we focus on minimizing the mean hop count between communicating nodes. We overview major datacenter topologies and analyze their basic properties.

3.1 Datacenter Network Topologies

We focus on the following three datacenter topologies:

(i) Fat-tree [2]. The Fat-tree topology is a hierarchical network built from multiple layers of switches as described in Figure 3. At the bottom layer, each switch connects directly to a group of compute nodes (e.g., GPUs). Higher layers of switches aggregate connections from lower layers, with bandwidth increasing toward the root of the tree.

In LLM training, it is common to employ a three-tier Fat-tree. The first tier typically represents the GPUs, while the upper tiers consist of switches. A common design choice is to match the number of nodes in the middle tier with the number of GPUs within each HBI, a rationale that will be further elaborated later in this work. By contrast, the number of nodes in the top tier (the spines) is more flexible: increasing the number of spine switches provides higher aggregate bandwidth, but also raises the overall cost of the network.

(ii) Dragonfly+ [14][8] is an emerging topology aiming to increase the effectiveness of switch radix and decrease network diameter, latency and cable cost. As shown in Figure 4, in this hierarchical topology switches are collected into groups. Groups are connected to each other directly, namely there is a pair of connected switches in every two groups. The topology within a group is based on a fully connected bipartite graph with switches organized in two levels. Switches in the two levels are called leaves (where GPUs are located) and spines (connected to spines in other groups).

(iii) HyperX [1] [11]. The HyperX topology is a multi-dimensional direct network. Nodes are arranged in a grid-like structure across several dimensions, and each node connects directly to other nodes along every dimension it participates in as shown in Figure 5. A GPU is located at each node.

The HyperX topology is determined by the number of dimensions which has a typical value of three. The hop distance

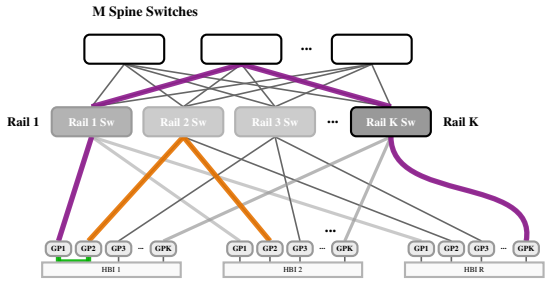


Figure 3: Fat-tree Network Topology. The architecture interconnects R HBI domains via K Rail switches and a top-layer Spine fabric (M in total). Each domain organizes K GPUs, enabling high-bandwidth local communication between GPUs in the HBI domain. Three routing paths are highlighted: (1) Intra-domain via HBI (Green); (2) Inter-domain via a Rail switch (Orange); and (3) Cross-rail via the Spine layer (Purple).

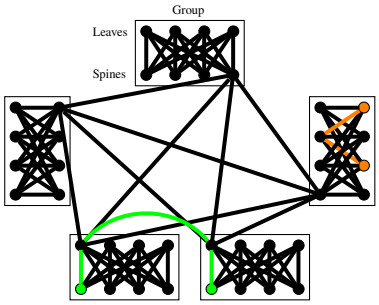


Figure 4: Dragonfly+ Topology (with five groups, switches of 8 ports). Each group is a fully connected bipartite. For each pair of groups, there are spines which are directly connected. Some edges (crossing groups) are omitted to avoid confusion. Each spine is connected to every other group by a single edge. Colors illustrate various ways to connect nodes.

equals the number of dimensions in which two nodes differ so the diameter equals the number of dimensions. A further design choice concerns how to distribute the sizes across dimensions. Parameters can be tuned to create on-demand networks with flexible properties.

3.2 Hop-Distance Heatmaps Across Topologies

To visualize the structural characteristics of each interconnect, we computed the pairwise *number of hops* between every pair of GPUs in an arbitrary configuration of the three evaluated topologies. Figures 6(a)–(c) show the resulting hop-distance heatmaps for Fat-tree, Dragonfly+ and HyperX, upon connecting 128 GPUs. Each matrix entry i, j represents the shortest

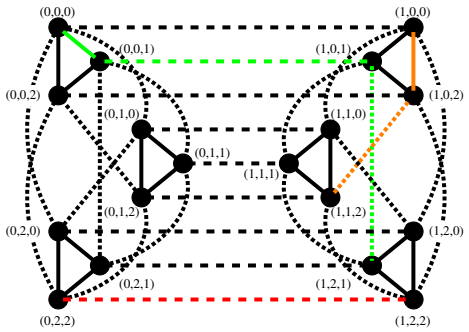


Figure 5: HyperX topology illustration of L 3 dimensions with S_1 2, S_2 S_3 3. There are N $\frac{L}{i1} S_i$ $2 \cdot 3 \cdot 3$ 18 nodes, each represented by three coordinates. Links along the different dimensions are shown in different line styles. Paths of various lengths are shown in different colors.

Table 2: Topologies and their main parameters.

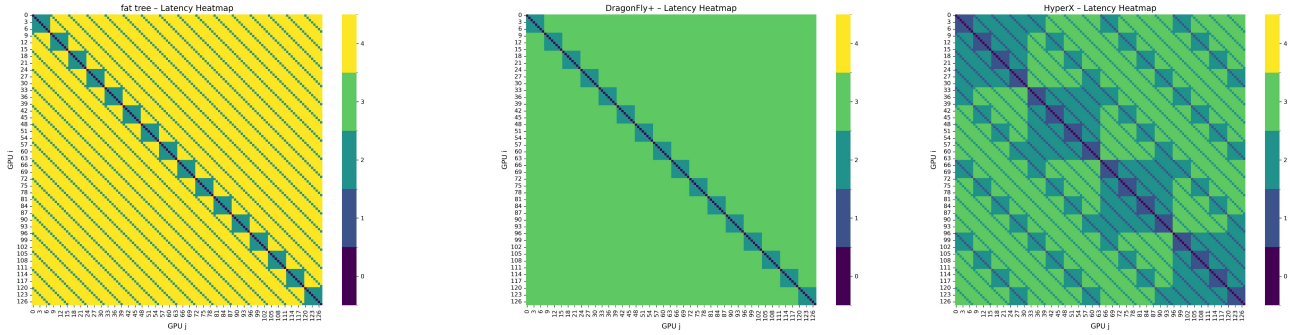
Topology	Main parameters	Values (by GPU count N)
Fat-tree	K GPU in HBI domain	TP group size (2,4,8,16,32)
	M spines	-
	R GPU per rail switch	NK
Dragonfly+	# GPUs per group	TP group size
	g groups	$\frac{\text{Total number of GPUs}}{\text{group size}}$
HyperX+	L 3 dimensions	3
	S_1 first dimension size	TP group size
	S_2 second dimension size	DP - Number of Model Replicas (2,4,8,16,32,64, 128,256,512)
	S_3 third dimension size	number of PP stages (1,2,4,8,16,32)

communication path length (in hops) between GPU i and GPU j, where darker colors indicate shorter (one-hop) connections and brighter colors indicate longer multi-hop routes. In addition, Figure 7 further illustrates the distribution of the number of hops across the three topologies.

Interpretation. The Fat-tree topology (Figure 6(a)) exhibits a highly regular, diagonal pattern—most GPU pairs are connected through four hops via shared spine switches.

In Dragonfly+ (Figure 6(b)), communication within local groups occurs in two hops, whereas inter-group communication requires three hops, forming visible block structures.

In contrast, the HyperX topology (Figure 6(c)) implies distance values ranging between one and three among different nodes, as implied by the number of dimensions set as three. The structure of the HyperX hop-distance heatmap closely resembles the corresponding transport matrix heatmap, and shown later in the paper. This similarity highlights that the communication is typically exchanged between directly connected neighbors, consistent with a single hop.



(a) Fat-tree topology **(b) Dragonfly+ topology** **(c) HyperX topology**

Figure 6: Hop-distance heatmaps showing the shortest number of hops between every pair of GPUs (i, j) in the three interconnect topologies, constructed over a system of 128 GPUs. Each topology assumes 8 GPUs per HBI domain (equivalently, one tensor-parallel group or server).

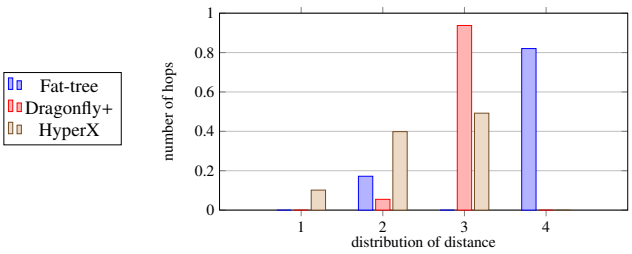


Figure 7: Distribution of Number of Hops in the three topologies.

4 Topology Setup and Workload Mapping

Table 2 summarizes main parameters of the topologies.

4.1 Fat-tree Topology Setup

For the Fat-tree topology, we adopted a mapping strategy designed to align the physical network hierarchy with the communication patterns of 3D parallelism. We explicitly define the domain size parameter, K, to equal the Tensor Parallelism (TP) degree. Consequently, each High-Bandwidth Domain encapsulates exactly one TP group, ensuring that the bandwidth-intensive synchronization required for tensor slicing occurs entirely over the local High-Bandwidth Interconnect (HBI).

A structural analysis of this mapping reveals a significant latency advantage: steady-state training traffic bypasses the top-tier network entirely. While TP communication is confined to the HBI, both DP gradient aggregation and PP point-to-point transfers occur via the Rail switches, which connect corresponding GPUs across domains. This ensures that all standard training flows avoid the Spine switches, strictly minimizing the hop count for every packet.

4.2 Dragonfly+ Topology Setup

In our implementation of the Dragonfly+ topology, GPUs are placed in accordance with the hierarchical structure of the network. Each tensor-parallel (TP) group is mapped to a single internal group within the topology, thereby exploiting the high bandwidth and rich intra-group connectivity characteristics of Dragonfly+. Within each such group, leaf switches are directly connected to the GPUs belonging to that TP group, while spine switches handle inter-group communication. This design ensures that intra-TP communication—required for tensor-parallel computation—remains localized within the same group, whereas inter-group communication (e.g., for data-parallel collectives) occurs through the spine-level links connecting the groups. In this way, the design balances local bandwidth utilization with reduced inter-group traffic, aligning naturally with the hierarchical structure of Dragonfly+ and the GPU parallelism model.

4.3 HyperX Topology Setup

For HyperX, we adopted a design tailored to the communication characteristics of common parallelism implementations—particularly the ring-based collectives used in most training frameworks. To emphasize this structure, no centralized switches were included in the HyperX interconnect.

The three dimensions of the 3D HyperX network correspond to the three parallelism modes:

- A first dimension **TP** connects GPUs tensor-parallel group to serve traffic within TP groups.
- A second dimension **DP** connects replicas of the model across data-parallel groups to serve traffic within stages.

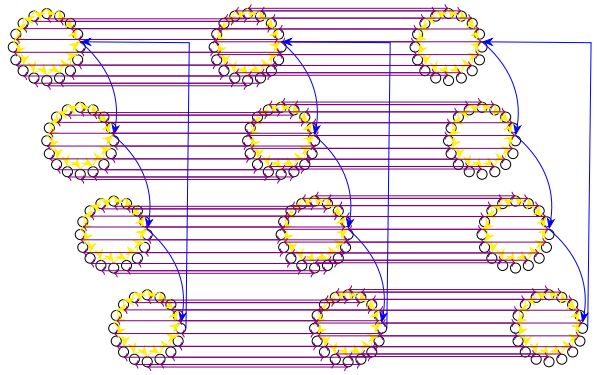


Figure 8: Illustration of the mapping into HyperX of the 192 GPUs in Figure 2.

- A third dimension **PP** connects corresponding GPUs participating in adjacent pipeline stages to serve traffic between adjacent stages.

Figure 8 illustrates a potential 3D HyperX connecting the 192 GPUs in Figure 2 as a topology with S_1 16, S_2 4, S_3 3. All non-zero values in the transport matrix refer to pairs of values with distance of a single hop.

Each GPU is directly connected to its immediate neighbors along these three dimensions, resulting in one-hop communication for all collective and point-to-point operations. We assume a ring-based implementation for TP and DP collectives, and peer-to-peer (P2P) communication for PP. The links along the TP dimension—connecting GPU i in TP group j to GPU $i1$ in the same group—are implemented using NVLink, providing high-bandwidth connections for tensor-parallel synchronization.

We acknowledge that, in current GPU server architectures, the number of direct interconnect links per GPU is physically limited and relatively small. However, inspired by the dual-plane design philosophy of high-performance network (HPN) topologies, we found it insightful to conceptually explore a setup where each GPU can maintain several direct, switchless links. This abstraction allows us to examine communication efficiency under an idealized, high-connectivity model that could inform future architectural design.

Indeed, not all links available in the HyperX are required here and the topologies can be further simplified by keeping along each dimension only links between adjacent GPUs.

5 Results and Communication Overhead Analysis

We evaluated the communication overhead across three interconnect topologies — Fat-tree, Dragonfly+, and HyperX — under multiple workload configurations. Each configuration was generated using the parameters described in Section 2,

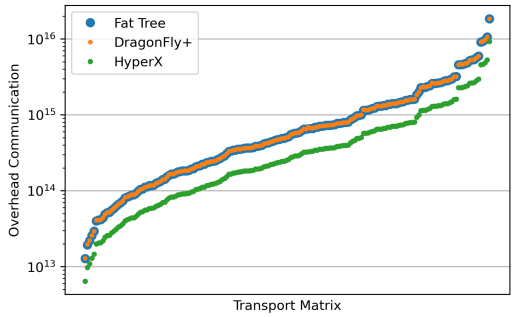


Figure 9: Communication Overhead Comparison Across Topologies (Logarithmic Scale). Each data point represents the measured overhead for a specific workload configuration. Overall, the HyperX topology exhibits an approximate 2× reduction in overhead compared to the alternative topologies.

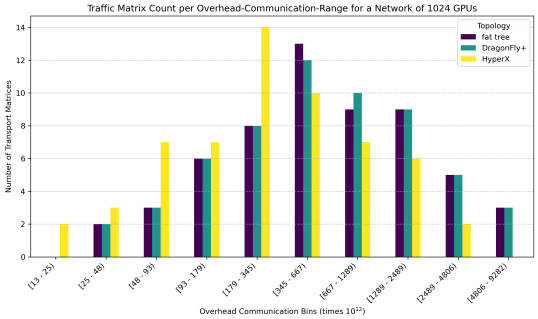


Figure 10: Traffic Matrix Distribution by Overhead Communication Bin across Varying Network Sizes for Network Size of 1024 GPUs.

covering variations in the number of pipeline stages (PP), tensor-parallel groups (TP) and data-parallel replicas (DP). Communication overhead was analyzed both per-parallelism dimension and in total.

5.1 Transport Matrices

We examined a total of 184 generated transport matrices across different parallelism settings. For these matrices the total number of GPUs (world size) ranges from 2^6 to 2^{12} while the number of GPUs per tensor-parallel (TP) group is among the values $2, 2^2, 2^3, 2^4, 2^5$. Smaller GPU counts and TP groups were sampled more sparsely. Together, these configurations span a wide range of scenarios needed to evaluate the interaction of TP, data parallelism, and pipeline parallelism in large-scale training. For the various settings, we found that the communication focuses on TP traffic, referring for each model to a portion of 98.7%-99.5%.

5.2 Overall Comparison Across Topologies

Figure 9 summarizes the total communication overhead aggregated across all types of parallelism. The HyperX topology consistently achieved the lowest overall overhead, serving as a lower bound for communication cost. The Fat-tree exhibits comparable behavior but with slightly higher overhead due to its multi-hop routing and shared switch contention. In contrast, Dragonfly+ shows larger variance and several high-outlier configurations, primarily caused by longer inter-group routes and non-uniform link utilization. Beyond the average communication overhead over all matrices, Figure 10 shows for the various topologies the count of traffic matrices whose communication overhead falls within defined logarithmic bins.

We also examine the *distribution of traffic volume as a function of the number of hops* (in the range of 1-4 hops) between communicating GPUs.

In the HyperX topology, all traffic is confined to one-hop links, as every GPU communicates directly with its peers in the ring-like interconnect, avoiding switch traversal entirely. Conversely, the Fat-tree topology exhibits a consistent two-hop pattern for all parallelism types, since every traffic during LLM training does not use the spine switches [16]. The Dragonfly+ topology presents a hybrid behavior: while traffic within TP-groups predominantly traverses two hops, within-stage (DP) and between-stages (PP) communication are evenly split between two- and three-hop paths due to inter-group routing. The resulting overall distribution, therefore, shows a visible dominance of two-hop communication, as the TP traffic volume substantially outweighs that of PP and DP. This observation aligns with our earlier findings that TP contributes the majority of communication load in large-scale transformer training, and it clarifies why the normalized latency of Dragonfly+ remains slightly above the theoretical two-hop baseline.

5.3 Summary of Observations

Overall, the results demonstrate:

- The **HyperX** topology minimizes communication overhead across all parallelism types due to its one-hop, switchless interconnect design and uniform link distribution.
- The **Fat-tree** topology consistently results in roughly double the overhead of HyperX, driven primarily by two-hop DP and TP communication through shared switches.
- The **Dragonfly+** topology exhibits the highest variability, typically exceeding the HyperX baseline by slightly more than 2×, reflecting its mix of two-hop and three-hop communication paths depending on the specific parallelism dimension and GPU placement.

6 Conclusions

This paper studies fundamental properties of the communication patterns in LLM training and examines the suitability of common datacenter topologies to serve them. We detail the three main components of such communication and study how their size is influenced by the model, number of parameters, and GPU count. Towards efficiently mapping GPUs to these topologies, we identify an interesting property of the HyperX topology that enables communication with ideal hop count for LLM training.

References

[1] Jung Ho Ahn et al. 2009. HyperX: Topology, routing, and packaging of efficient large-scale networks. In *ACM/IEEE SC*.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *ACM SIGCOMM*.

[3] Jehyeon Bang, Yujeong Choi, Myeongwoo Kim, Yongdeok Kim, and Minsoo Rhu. 2023. vTrain: A Simulation Framework for Evaluating Cost-effective and Compute-optimal Large Language Model Training. *CoRR* abs/2312.12391 (2023). arXiv:2312.12391 [cs.DC] doi:10.48550/arXiv.2312.12391

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901. arXiv:2005.14165 [cs.CL] doi:10.48550/arXiv.2005.14165

[5] Zina Chkirbene, Ridha Hamila, Ala Gouissem, and Unal Devrim. 2024. Large language models (LLM) in industry: A survey of applications, challenges, and trends. In *IEEE International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*.

[6] Aakanksha Chowdhery et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research* 24, 240 (2023), 1–113.

[7] Nikhil Kandpal and Colin Raffel. 2025. Position: The Most Expensive Part of an LLM should be its Training Data. *CoRR* abs/2504.12427 (2025).

[8] John Kim, William J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *International Symposium on Computer Architecture (ISCA)*.

[9] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M. Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[10] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *ACM SIGKDD*.

[11] Ori Rottenstreich. 2023. Path Diversity and Survivability for the HyperX Datacenter Topology. *IEEE Trans. Netw. Serv. Manag.* 20, 3 (2023), 2370–2385.

[12] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan

R. K. Ports, and Peter Richtárik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[13] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[14] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. 2017. Dragonfly+: Low Cost Topology for Scaling Datacenters. In *IEEE HiPINEB Workshop*.

[15] Hugo Touvron et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[16] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2024. Rail-only: A low-cost high-performance network for training LLMs with trillion parameters. In *IEEE Symposium on High-Performance Interconnects (HOTI)*.

[17] Xizheng Wang et al. 2025. SimAI: Unifying Architecture Design and Performance Tuning for Large-Scale Large Language Model Training with Scalability and Precision. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.