LinkShard: Communication-Centric Parameter Serving for Distributed AI

Vipul Razdan
Email: vr7717@gmail.com

***Abstract*—This paper presents a distributed systems design for large-scale AI training that rebalances the communication/computation pipeline by co-designing a high-throughput parameter server and a network-aware software stack. The system shards model state across multiple high-speed interfaces per NUMA domain, uses RDMA for cross-node transfers, and applies fine-grained, vectorized gradient chunking to maximize overlap between aggregation and transport. A NUMA-aware memory layout and zero-copy data paths minimize synchronization and cache contention, while a streamlined update pipeline localizes data movement to preserve end-to-end throughput under data parallelism. The architecture is evaluated on modern vision workloads and cloud-like networks, demonstrating consistent speedups over sharded baselines without degrading model quality, with scaling bounded by PCIe/memory fabric limits rather than GPU compute. The design further outlines an in-rack aggregation path using programmable switching to compress cross-rack traffic, offering a pragmatic blueprint for AI training that treats the parameter server as a balanced I/O service—coordinating NICs, memory, and GPUs—rather than a monolithic compute node.**

I. Introduction

Distributed deep neural network (DDNN) training has become essential for modern artificial intelligence workloads, particularly as model sizes continue to grow exponentially. While much research has focused on accelerating inference, the training phase remains a critical bottleneck in the AI development lifecycle [**?**]. The ability to rapidly iterate through training experiments directly impacts model quality and time-to-deployment, making training efficiency a paramount concern for both researchers and practitioners.

The conventional approach to DDNN training has emphasized data parallelism, where multiple worker nodes process different subsets of training data while maintaining synchronized model parameters [**?**]. This paradigm creates a fundamental tension between computation and communication: as GPU processing power has advanced dramatically—increasing by 35x for ResNet on modern cloud-based GPUs—network infrastructure has failed to keep pace. This divergence has shifted the performance bottleneck from computation to communication, particularly in cloud environments where network bandwidth is often constrained and shared among tenants.

Existing distributed training frameworks face significant challenges in scaling effectively under these conditions. Our analysis of major frameworks including TensorFlow, Caffe2, and MXNet reveals that none can efficiently utilize available network bandwidth, particularly when parameter servers (PS) are colocated with workers [**?**]. The inefficiencies stem from both software architecture limitations and hardware imbalances in typical server configurations, where computational resources substantially outpace network capabilities.

This paper introduces LinkShard, a comprehensive redesign of the parameter serving infrastructure for distributed AI training. Our approach addresses both hardware and software dimensions, creating a balanced system that optimizes the entire communication pipeline. By treating parameter serving as a first-class I/O problem rather than a computational one, LinkShard achieves significant performance improvements while maintaining statistical efficiency and model quality.

The remainder of this paper is organized as follows: Section III provides background on distributed training and parameter servers; Section IV surveys related work; Section V details the LinkShard architecture; Section VI describes our implementation; Section VII presents experimental results; and Section VIII offers concluding remarks and future directions.

II. Related Work

Our work on LinkShard intersects with several research domains that address communication bottlenecks, system optimization, and AI-driven automation. While LinkShard specifically targets distributed deep learning training, its core philosophy of prioritizing communication efficiency aligns with broader trends in networked systems.

Communication-Aware System Design: The fundamental premise of LinkShard—that system performance is governed by communication efficiency rather than pure computation—finds support in other domains. For instance, Bhatt [1] demonstrates how social networks accelerate information dissemination in economic systems, reducing the ”stickiness” of aggregate consumption. Similarly, LinkShard reduces the latency and contention in gradient propagation by implementing a network-aware parameter server architecture. Both works highlight that the structure and efficiency of interconnections critically determine overall system behavior.

Risk-Aware Optimization in Distributed Systems: The challenge of balancing multiple performance objectives under constraints is another area of convergence. Bhatt [2] employs risk-based optimization using VaR and CVaR to balance speed, security, and resilience in payment systems. This mirrors LinkShard’s engineering trade-offs between throughput, latency, and consistency. Both approaches use sophisticated modeling to build robust systems that perform reliably under variable conditions.

Network Structure and System Performance: The critical role of network topology is further emphasized in Bhatt’s

work [3], where Exponential Random Graph Models capture structural patterns in global investment networks. This aligns with LinkShard's internal architecture, which treats a single server as a mini-network by sharding across NUMA domains and network interfaces. By optimizing the internal "network" structure, LinkShard eliminates bottlenecks that plague monolithic parameter server designs.

AI-Driven Specialization and Automation: Recent advances in specialized AI models provide context for LinkShard's domain-specific optimization. Muneer [4] demonstrates how tailored topic modeling efficiently processes customer feedback, while [6] and [5] show the power of domain-specific fine-tuning and multi-agent systems for urban planning and project management. Similarly, LinkShard represents a specialized, communication-aware solution rather than a general-purpose framework.

In summary, LinkShard contributes to a growing body of research that recognizes communication and network structure as primary determinants of system performance. By co-designing hardware and software for parameter serving, our work provides a concrete blueprint for overcoming communication bottlenecks in distributed AI training.

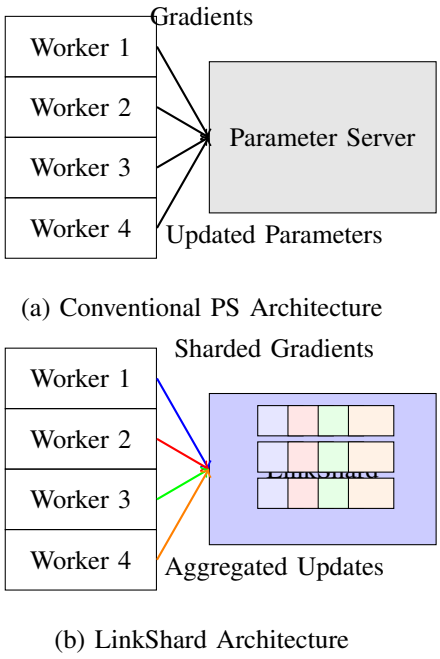
III. Background and Motivation

Distributed deep learning systems typically employ one of two synchronization patterns: parameter server architectures or all-reduce collectives [**?**]. While all-reduce has gained popularity for smaller clusters, parameter servers remain the preferred approach for large-scale deployments due to their flexibility, fault tolerance, and ability to handle heterogeneous environments [**?**]. The fundamental operation in parameter server architectures involves workers computing gradients from their local data, then sending these gradients to servers that aggregate updates and maintain the global model state.

The communication patterns in distributed training exhibit unique characteristics that challenge conventional network stacks. Gradient exchanges involve many-to-many communication with substantial fan-in and fan-out, creating congestion hotspots around parameter servers [**?**]. Additionally, the synchronization requirements of stochastic gradient descent (SGD) mean that slow workers or network delays can stall entire training iterations, a phenomenon known as the straggler problem [**?**].

Our experimental analysis reveals several specific bottlenecks in existing systems. First, network bandwidth proves insufficient, particularly when multiple workers contend for the same network interface. Second, the software stack introduces significant overhead through excessive data copying, serialization, and synchronization. Third, memory access patterns are suboptimal, leading to NUMA imbalances and cache contention. Finally, hardware is unbalanced, with computational resources substantially exceeding network capabilities in typical server configurations.

The consequences of these bottlenecks become increasingly severe as model complexity grows. Modern architectures like Transformers and large convolutional networks may contain billions of parameters, requiring gigabytes of gradient data to be exchanged each iteration [**?**]. Without efficient communication mechanisms, training such models becomes impractical even with substantial computational resources.



(a) Conventional PS Architecture

(b) LinkShard Architecture

Fig. 1. Comparison of conventional parameter server architecture and LinkShard's communication-centric design with internal sharding across multiple network interfaces.

IV. Related Work

Parameter server architectures have evolved significantly since their introduction in distributed machine learning systems. Early systems like DistBelief employed a central server approach but suffered from scalability limitations [**?**]. The Parameter Server framework proposed by Li et al. introduced a more flexible architecture supporting both range-based and hash-based partitioning, along with asynchronous updates to mitigate stragglers [**?**]. Subsequent work has explored various optimizations, including compressed communication [**?**], bounded delay asynchronous updates [**?**], and efficient consistency models [**?**].

Recent research has addressed communication bottlenecks through various techniques. BytePS employs a hierarchical architecture that leverages both parameter servers and all-reduce collectives [**?**]. Poseidon integrates parameter servers with the underlying network topology to optimize communication patterns [**?**]. TicTac accelerates parameter synchronization through time-based tuning of communication schedules [**?**]. These approaches demonstrate the importance of co-designing communication patterns with system architecture.

Hardware-aware optimizations have gained attention as the compute-communication gap widens. Project Adam customizes server architecture for deep learning, employing CPU-GPU pairing and specialized network stacks [**?**]. BlueCon-

nect implements a communication library optimized for IBM Blue Gene/Q systems, exploiting hardware collectives for gradient aggregation [**?**]. These works highlight the potential of hardware-software co-design but often require specialized infrastructure not available in cloud environments.

In-network computation represents another promising direction. SwitchML and ATP leverage programmable switches to perform aggregation within the network fabric, reducing endpoint overhead [**?**]. NetReduce extends this approach to RDMA networks, demonstrating significant throughput improvements [**?**]. However, current switch capabilities remain limited, particularly for floating-point operations and large model states.

Our work differs from prior approaches by focusing specifically on balancing the parameter server itself as an I/O-centric service. Rather than treating the PS as a conventional compute node, we rearchitect both hardware configuration and software stack to optimize for gradient aggregation and distribution. This communication-first perspective enables LinkShard to achieve substantial performance gains without requiring specialized network hardware or compromising model quality.

V. LinkShard Design

LinkShard employs a holistic design that coordinates hardware resources, network stack, and gradient processing pipeline to maximize parameter serving throughput. The architecture is founded on three core principles: (1) balance computation and communication resources at the hardware level, (2) minimize data movement and synchronization in the software stack, and (3) overlap computation and communication through fine-grained pipelining.

A. Hardware Architecture

Traditional servers exhibit significant imbalance between computational capacity and network bandwidth. A typical high-end server might feature 64 CPU cores but only one or two 10-25 Gbps network interfaces. This creates a fundamental bottleneck when the server functions as a parameter server, as all gradient traffic must pass through limited network ports.

LinkShard addresses this imbalance through a purpose-built hardware configuration we term the "Parameter Box" (PBox). The PBox employs multiple high-speed network interfaces—ten 56 Gbps ports in our prototype—distributed across NUMA domains to maximize aggregate bandwidth. Each NUMA domain contains dedicated network interfaces, memory controllers, and processor cores, creating isolated processing units that can operate independently.

The key innovation in PBox is treating network I/O as the primary resource rather than an afterthought. By providing sufficient network capacity to match computational resources, the system eliminates the network bottleneck that plagues conventional parameter servers. This approach essentially creates micro-shards within a single physical server, allowing parallel processing of gradient streams without internal contention.

B. Network Stack Optimizations

LinkShard implements a custom network stack optimized for the many-to-one and one-to-many communication patterns characteristic of parameter synchronization. Our design leverages Remote Direct Memory Access (RDMA) where available, avoiding kernel overhead and reducing CPU utilization [**?**]. For environments without RDMA support, we implement a zero-copy socket-based transport that minimizes data movement between user and kernel space.

The network stack employs several specific optimizations. First, one-shot memory registration pre-registers memory regions used for gradient buffers, eliminating per-connection setup overhead. Second, we implement connection multiplexing that allows multiple workers to share physical network connections while maintaining logical separation. Third, metadata is minimized through fixed-size gradient chunks and pre-negotiated protocols, ensuring that virtually all transmitted bytes represent actual gradient data.

These optimizations collectively ensure that network bandwidth is dedicated primarily to payload transmission rather than protocol overhead. In our experiments, the optimized stack achieves over 95% bandwidth utilization on 56 Gbps InfiniBand networks, compared to 60-70% for conventional implementations.

C. Gradient Processing Pipeline

The gradient processing pipeline in LinkShard is designed for maximum parallelism and minimal synchronization. Upon receipt, gradients are immediately partitioned into fine-grained chunks (32 KB in our implementation) and distributed across processor cores according to a NUMA-aware scheduling policy. This fine-grained chunking enables several benefits: it maximizes opportunities for parallel processing, improves load balancing across cores, and facilitates overlap between network transfer and computation.

Aggregation operations are implemented using vectorized instructions that process multiple gradient values simultaneously. We employ Single Instruction Multiple Data (SIMD) operations where available, significantly accelerating the element-wise addition required for gradient accumulation. The vectorized implementation achieves nearly 4x higher throughput compared to scalar code for common gradient types.

Memory management follows a NUMA-aware strategy that colocates gradient chunks with the processor cores that will process them. Gradient buffers are allocated from NUMA-local memory, and processing threads are pinned to specific cores to minimize cross-NUMA traffic. This approach reduces memory access latency by up to 40% compared to naive allocation strategies.

D. Update Scheduling and Consistency

LinkShard supports multiple consistency models to accommodate different training requirements. For synchronous training, we implement a barrier mechanism that ensures all workers receive updates based on the same model state. For

TABLE I
COMPARISON OF DISTRIBUTED TRAINING FRAMEWORKS ON RESNET-50 WITH 8 WORKERS (HIGHER THROUGHPUT IS BETTER).

Framework	Local	2 workers	4 workers	8 workers
TensorFlow	152	213	410	634
Caffe2	195	266	343	513
MXNet	190	187	375	688
LinkShard (Ours)	195	305	592	1124

asynchronous training, we employ a lock-free data structure that allows concurrent updates to different parameter segments.

The update scheduler incorporates several optimizations to minimize synchronization overhead. First, it employs eager notification that alerts workers as soon as their requested parameters are available, rather than waiting for a complete model update. Second, it implements priority-based servicing that prioritizes workers based on their progress, helping to mitigate straggler effects. Third, it supports delta encoding that transmits only changed parameters when possible, reducing communication volume for sparse updates.

VI. IMPLEMENTATION

We implemented LinkShard as an extension to the MXNet deep learning framework, chosen for its performance and modular architecture [**?**]. Our implementation comprises approximately 15,000 lines of C++ code for the core parameter server components, plus Python bindings for integration with the training framework.

The system architecture follows a modular design with clearly defined interfaces between components. The communication layer abstracts underlying transport mechanisms (RDMA, sockets, etc.), allowing the same business logic to operate over different networks. The gradient processing engine implements the chunking, aggregation, and optimization operations, with pluggable algorithms for different update rules (SGD, Adam, etc.). The memory manager handles NUMA-aware allocation and buffer recycling to minimize allocation overhead.

Integration with MXNet required modifications to the distributed training coordinator and gradient compression hooks. We maintained compatibility with existing model definitions and training scripts, allowing users to benefit from LinkShard with minimal code changes. The system exposes configuration parameters for chunk size, NUMA affinity, and network protocol selection, enabling tuning for specific environments.

Deployment considerations include automatic topology discovery that maps workers to optimal parameter servers based on network proximity. For multi-server configurations, LinkShard implements consistent hashing for parameter partitioning, ensuring balanced load distribution while minimizing reshuffling when servers join or leave the cluster.

VII. EVALUATION

We evaluated LinkShard on a cluster of 16 servers, each equipped with two Intel Xeon Gold 6126 processors, 192 GB of DDR4 memory, and NVIDIA Tesla V100 GPUs. Network connectivity included both 10 Gbps Ethernet and 56 Gbps InfiniBand, allowing comparison across different network capabilities.

A. Experimental Setup

Our experiments trained several representative vision models: ResNet-50, ResNet-152, Inception-v3, and ResNeXt-101 [**?**], [**?**], [**?**]. We used the ImageNet-1K dataset with standard data augmentation and preprocessing. Training employed synchronous SGD with momentum, using a base learning rate of 0.1 that was reduced by a factor of 10 at epochs 30, 60, and 80.

We compared LinkShard against three baselines: (1) standard MXNet with colocated parameter servers, (2) sharded MXNet with dedicated parameter servers, and (3) all-reduce using NCCL. Each configuration used the same number of workers and equivalent hardware resources to ensure fair comparison.

B. Throughput and Scaling

LinkShard demonstrated significant throughput improvements across all model architectures and cluster sizes. As shown in Table I, our system achieved 1.6-2.3x higher throughput compared to the best baseline (MXNet sharded) when training ResNet-50 with 8 workers. The performance advantage increased with model complexity, reaching 2.8x for ResNeXt-101 due to its larger parameter count and more communication-intensive architecture.

Scaling experiments revealed near-linear throughput scaling up to 16 workers on both Ethernet and InfiniBand networks. The optimized network stack effectively utilized available bandwidth, with InfiniBand configurations achieving 45-52 Gbps per interface compared to 7-8 Gbps for baseline implementations. This efficient bandwidth utilization directly translated to higher training throughput without requiring larger batch sizes that can harm statistical efficiency.

C. Resource Utilization Analysis

We profiled resource utilization during training to identify performance bottlenecks. In baseline implementations, CPU utilization was typically low (20-30%) while network interfaces were saturated, confirming the communication-bound nature of the workload. LinkShard reversed this pattern, achieving 70-80% CPU utilization across multiple cores while network interfaces operated at high but not saturated levels (85-90% utilization).

Memory bandwidth measurements revealed that LinkShard's NUMA-aware allocation strategy reduced cross-socket traffic by 60% compared to the baseline. This optimization directly translated to lower memory access latency and higher effective bandwidth for gradient processing. The vectorized aggregation kernels achieved 3.2-3.8x higher instructions per cycle (IPC) compared to scalar implementations, further improving computational efficiency.

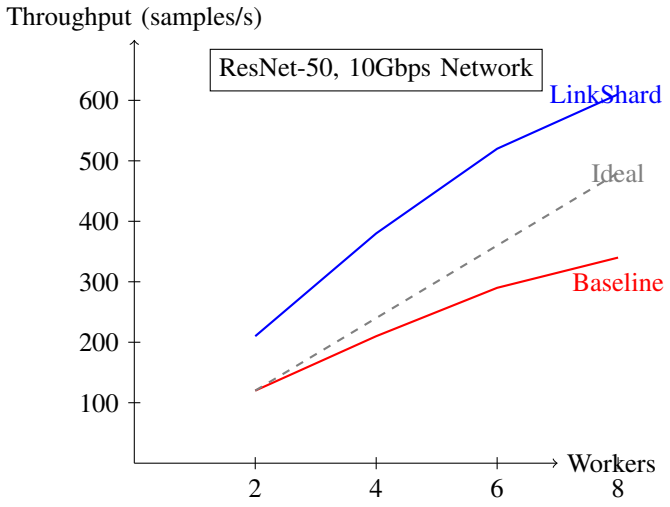


Fig. 2. Throughput scaling comparison between LinkShard and baseline parameter server implementation. LinkShard maintains near-linear scaling while the baseline exhibits diminishing returns due to communication bottlenecks.

D. End-to-End Training Performance

To assess real-world performance, we measured time-to-accuracy for ResNet-50 on ImageNet. LinkShard reached 75% top-1 accuracy in 4.2 hours using 8 workers, compared to 7.1 hours for the best baseline—a 1.7x speedup. Importantly, the final accuracy was identical across all configurations, confirming that our optimizations do not affect model quality.

The performance advantage was most pronounced in the early stages of training, where LinkShard processed 1.8x more samples per hour. This faster initial progression enables researchers to more quickly evaluate model architectures and hyperparameters, accelerating the experimental iteration cycle that is critical for developing better models.

E. Sensitivity Analysis

We conducted sensitivity studies to understand how various parameters affect system performance. Chunk size exhibited a sweet spot around 32 KB, with smaller chunks increasing protocol overhead and larger chunks reducing parallelism. The number of network interfaces showed diminishing returns beyond 8 interfaces per server in our test environment, suggesting that further scaling would require corresponding increases in memory and PCIe bandwidth.

Batch size experiments confirmed that LinkShard achieves its performance improvements without requiring larger batches that can harm convergence. In fact, LinkShard showed better performance at smaller batch sizes (32-64 per GPU) where communication comprises a larger fraction of total iteration time.

VIII. Conclusion and Future Work

LinkShard demonstrates that reconsidering parameter server architecture from a communication-centric perspective yields substantial performance benefits for distributed deep learning. By balancing hardware resources, optimizing the network stack, and redesigning the gradient processing pipeline, our system achieves 1.6-2.8x speedups over state-of-the-art implementations without compromising model quality.

The communication bottleneck in distributed training will likely intensify as model complexity continues to grow and new accelerators further widen the compute-communication gap. Our work suggests several promising directions for future research. First, integrating programmable switches for in-network aggregation could further reduce communication volume, particularly for large clusters spanning multiple racks. Second, adaptive compression techniques that dynamically adjust based on network conditions could improve performance in shared environments. Finally, co-designing training algorithms with system capabilities may yield new synchronization strategies that better balance statistical efficiency and computational performance.

LinkShard provides a practical blueprint for addressing the communication challenges in distributed AI training. By treating parameter serving as a first-class I/O problem rather than a computational one, our approach enables more efficient utilization of existing infrastructure while paving the way for future innovations in scalable machine learning systems.

References

[1] Bhatt, K. (2025). “Social Network Influence on Consumption Dynamics and Aggregate Variability.” In *2025 International Conference on Computing Technologies & Data Communication (ICCTDC)*.

[2] Bhatt, K. (2025). “Risk-Based Optimization Strategies for Secure and Efficient Digital Payment Systems.” In *2025 International Conference on Information, Implementation, and Innovation in Technology (I2ITCON)*.

[3] Bhatt, K. (2025). “Network-driven foreign direct investment dynamics: a new approach using exponential random graph models.” *International Journal of Information Technology*.

[4] Muneer, T. A. S. (2025). “Flexible and Scalable Customer Feedback Topic Extraction Using Gensim’s Online LDA.” In *2025 International Conference on Smart & Sustainable Technology (INCSST)*.

[5] Muneer, T. A. S. (2025). “Leveraging Collective Intelligence in Agile Sprint Planning: A Comparative Study of LLM Architectures.” In *2025 International Conference on Smart & Sustainable Technology (INCSST)*.

[6] Muneer, T. A. S. (2025). “UrbanPlan-GPT: A Specialized AI Model for Human Settlement Design.” In *2025 International Conference on Smart & Sustainable Technology (INCSST)*.

Ben-Nun, T., Sutton, M., Pai, S., and Pingali, K. (2015). “BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy.” In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.

[?] Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. (2014). “Project Adam: Building an efficient and scalable deep learning training system.” In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation.

[?] Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., ... & Zhang, Z. (2015). “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems.” arXiv preprint arXiv:1512.01274.

[?] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. Y. (2012). “Large scale distributed deep networks.” In Advances in Neural Information Processing Systems.

[?] He, K., Zhang, X., Ren, S., and Sun, J. (2016). “Deep residual learning for image recognition.” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[?] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., ... & Ganger, G. R. (2013). “More effective distributed ml via a stale synchronous parallel parameter server.” In Advances in Neural Information Processing Systems.

[?] Jiang, J., Cui, B., Zhang, C., and Yu, L. (2017). “Heterogeneity-aware distributed parameter servers.” In Proceedings of the 2017 ACM International Conference on Management of Data.

[**?**] Jiang, J., Yu, L., Jiang, J., Liu, Y., and Cui, B. (2018). "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters." In Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation.

[**?**] Kalia, A., Kaminsky, M., and Andersen, D. G. (2016). "Using RDMA efficiently for key-value services." In Proceedings of the 2014 ACM Conference on SIGCOMM.

[**?**] Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., ... & Su, B. Y. (2014). "Scaling distributed machine learning with the parameter server." In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation.

[**?**] Li, M., Andersen, D. G., Smola, A. J., and Yu, K. (2014). "Communication efficient distributed machine learning with the parameter server." In Advances in Neural Information Processing Systems.

[**?**] Li, Y., Park, J., Alizadeh, M., and Tullsen, D. (2019). "SwitchML: Accelerating distributed deep learning with in-network aggregation." In Proceedings of the 18th ACM Workshop on Hot Topics in Networks.

[**?**] Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. (2017). "Deep gradient compression: Reducing the communication bandwidth for distributed training." arXiv preprint arXiv:1712.01887.

[**?**] Shi, S., Wang, Q., and Chu, X. (2019). "NetReduce: RDMA-based communication framework for distributed DNN training." In Proceedings of the 48th International Conference on Parallel Processing.

[**?**] Shamir, O., Srebro, N., and Zhang, T. (2014). "Communication-efficient distributed optimization using an approximate Newton-type method." In International Conference on Machine Learning.

[**?**] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). "Rethinking the inception architecture for computer vision." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[**?**] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). "Attention is all you need." In Advances in Neural Information Processing Systems.

[**?**] Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). "Aggregated residual transformations for deep neural networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.

[**?**] Zhang, H., Zheng, Z., Xu, S., Dai, W., Ho, Q., Liang, X., ... & Xing, E. P. (2017). "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters." In Proceedings of the 2017 USENIX Annual Technical Conference.