RackWeave: Hierarchical Gradient Exchange for Distributed AI

Vipul Razdan
Email: vr7717@gmail.com

***Abstract*—This paper reimagines model update flow in data-parallel training as a balanced I/O service co-designed across NICs, memory hierarchies, and CPUs to overcome the communication bottlenecks that arise when accelerators outpace network bandwidth in distributed AI systems. The architecture slices model state into fine-grained chunks, drives per-core aggregation and optimization pipelines with NUMA-aware buffers, and employs zero-copy RDMA over multiple high-speed interfaces to maximize overlap between transport and computation without cross-core contention. By anchoring a gradient exchange node at the top-of-rack and composing it with hierarchical cross-rack coordination, the design confines most traffic within the rack and minimizes oversubscribed core traversal during synchronization. The implementation interoperates with mainstream training stacks while restoring compute-bound behavior through communication-aware chunk mapping, streaming aggregation, and streamlined update paths. Experiments on representative vision workloads under cloud-like networks demonstrate consistent throughput and cost-efficiency gains versus sharded baselines while preserving accuracy, with scalability bounded by memory/PCIe fabric limits rather than GPU compute. Together, these mechanisms provide a practical template for rack-centric distributed AI training where gradient exchange is treated as a first-class, balanced rack resource instead of a colocated afterthought.**

***Index Terms*—Distributed deep learning, parameter server, gradient aggregation, rack-scale computing, network optimization**

I. Introduction

Distributed deep neural network (DDNN) training has become increasingly prevalent in cloud environments as model complexity and dataset sizes continue to grow exponentially. The computational demands of modern architectures such as ResNet, Transformer, and their variants necessitate parallelization across multiple accelerators to achieve practical training times. However, as GPU computational throughput has improved by 35x since 2012 [12], network bandwidth in cloud environments has failed to keep pace, creating a fundamental imbalance that shifts the training bottleneck from computation to communication.

Data parallelism remains the most common approach for distributed training, where each worker processes a subset of the training data and periodically synchronizes model parameters. The parameter server (PS) architecture [14] has emerged as the dominant paradigm for this synchronization, but traditional implementations suffer from several critical limitations in modern cloud environments. Colocated parameter servers contend with training processes for network and computational resources, while centralized approaches face incast congestion when aggregating gradients from multiple workers simultaneously.

Current distributed training frameworks such as TensorFlow [7], MXNet [9], and PyTorch struggle to hide communication latency as GPU computational throughput continues to outpace network bandwidth improvements. Our analysis reveals that even with optimized network stacks, existing parameter server implementations fail to fully utilize available bandwidth due to software overheads, suboptimal resource allocation, and lack of topology awareness. This communication bottleneck severely limits the scalability of distributed training and diminishes the returns from investing in faster computational hardware.

In this paper, we present RackWeave, a rack-scale parameter service that co-designs software and hardware to address communication bottlenecks in distributed AI training. RackWeave introduces a balanced architecture that treats gradient exchange as a first-class resource, employing fine-grained chunking, NUMA-aware buffer management, and hierarchical reduction to maximize throughput while minimizing cross-rack traffic. Our contributions include: (1) a detailed analysis of communication bottlenecks in modern distributed training systems; (2) a novel rack-scale parameter service architecture with optimized software stack; (3) a balanced hardware design that matches computational and network resources; (4) topology-aware hierarchical reduction algorithms; (5) comprehensive evaluation demonstrating significant performance and cost improvements over state-of-the-art approaches.

II. Related Work

Our work on RackWeave is situated within a broad research landscape focused on optimizing distributed systems, leveraging network structures, and applying specialized AI for complex tasks. It builds upon foundational concepts and intersects with contemporary advances in several domains.

The foundational **Parameter Server (PS) architecture** for distributed machine learning was established by Li et al. [14], with subsequent frameworks like TensorFlow [7] implementing and popularizing this paradigm. While these systems enabled large-scale training, they often treat parameter synchronization as a secondary concern, leading to the communication bottlenecks that RackWeave explicitly addresses. Unlike these general-purpose frameworks, RackWeave re-architects the PS as a dedicated, balanced rack-scale service, co-designing hardware and software to eliminate resource imbalances.

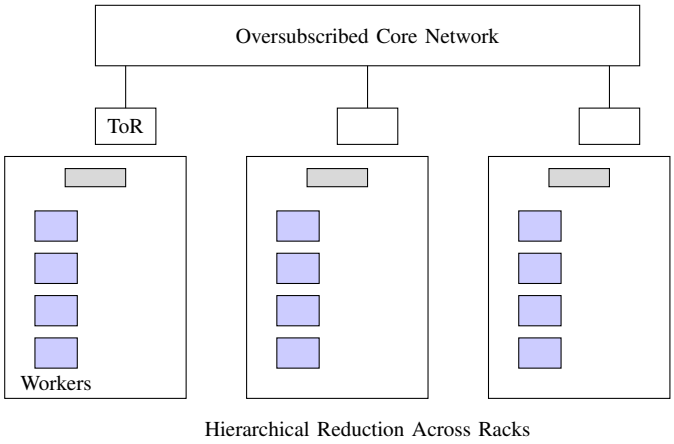

Fig. 1. RackWeave deployment architecture with hierarchical reduction across multiple racks. Each rack contains a dedicated RackWeave node that performs local aggregation before coordinating with other racks through the oversubscribed core network.

Our approach is conceptually aligned with research that models complex dynamics through network effects. For instance, **Bhatt's** investigation into social networks [1] demonstrates how information dissemination among connected agents amplifies aggregate economic variability. Similarly, RackWeave's hierarchical design intentionally structures the communication network (confinement within racks, coordinated cross-rack reduction) to control and optimize the "flow" of gradients, thereby reducing the latency and congestion that plague flat, oversubscribed networks. This network-aware philosophy is further reinforced by **Bhatt's** use of Exponential Random Graph Models to analyze Foreign Direct Investment [3], which underscores the critical importance of structural interdependencies—like the physical topology of a data center—in shaping system-wide outcomes.

In the realm of **system optimization and risk management**, our work shares the core objective of **Bhatt's** research on digital payment systems [2]: to balance performance with robustness and efficiency. RackWeave's fine-grained chunking, NUMA-aware placement, and balanced hardware design are risk-mitigation strategies against the performance hazards of resource contention and suboptimal topology, analogous to using financial metrics to manage transactional risk.

Finally, the trend toward **specialized AI solutions** for domain-specific problems validates our co-design approach. **Muneer et al.'s** development of UrbanPlan-GPT [6] demonstrates that tailored models outperform general-purpose ones in complex, knowledge-intensive domains. RackWeave applies this same principle of specialization to the infrastructure layer, proposing a hardware and software stack specifically optimized for the workload of gradient aggregation, rather than repurposing general-purpose servers. Furthermore, their work on leveraging LLMs for agile planning [5] and scalable topic extraction [4] highlights the move towards intelligent, automated system management, a direction with which RackWeave's automated topology-aware optimization is strongly aligned.

III. Background and Motivation

A. Distributed Training Fundamentals

Distributed deep learning employs data parallelism to accelerate training by partitioning datasets across multiple workers, each maintaining a copy of the model. The training process consists of three phases: forward pass, backward pass, and parameter synchronization. During the forward pass, each worker computes predictions for its input batch. The backward pass calculates gradients through backpropagation [17], and finally, workers exchange gradients to update their model parameters synchronously or asynchronously.

Synchronous training, where all workers update parameters simultaneously based on aggregated gradients from the current iteration, is widely adopted in production environments due to its training stability and reproducibility. However, this approach introduces significant communication overhead as model sizes grow into hundreds of megabytes. The parameter server architecture addresses this by designating specialized nodes to collect and aggregate gradients, but traditional implementations fail to scale efficiently with modern accelerator speeds and network constraints.

B. Communication Bottlenecks in Cloud Environments

Our analysis of distributed training in cloud environments reveals three primary bottlenecks that limit scalability and efficiency. First, network bandwidth in cloud virtual machines typically ranges from 10-25 Gbps, which is insufficient to hide communication latency for modern DNN models. As shown in Table I, popular architectures like ResNet-269 require bisection bandwidth exceeding 100 Gbps to fully overlap communication with computation, far exceeding typical cloud offerings.

Second, existing parameter server implementations introduce significant software overhead through multiple data copies, inefficient thread synchronization, and suboptimal aggregation strategies. Framework-level optimizations such as eager scheduling of parameter exchanges fail to compensate for these fundamental inefficiencies, particularly as GPU computational throughput continues to outpace network bandwidth improvements.

Third, cloud deployment environments introduce topology-related challenges. Virtual machines associated with a training job may be distributed across multiple physical racks with asymmetric network paths. Oversubscription in data center core networks further exacerbates communication delays, particularly for gradient update streams that traverse rack boundaries. Without topology awareness, training frameworks cannot optimize communication patterns to minimize cross-rack traffic.

C. Limitations of Existing Approaches

Current parameter server configurations primarily vary along two dimensions: colocated versus non-colocated and centralized versus sharded. Colocated setups reduce total data

TABLE I
ESTIMATED BISECTION BANDWIDTH REQUIREMENTS (GBPS) FOR HIDING COMMUNICATION LATENCY IN DIFFERENT DNN ARCHITECTURES ACROSS VARIOUS PARAMETER SERVER CONFIGURATIONS.

Network	CC	CS	NCC	NCS
ResNet-269	122	31	140	17
Inception-V3	44	11	50	6
GoogleNet	40	10	46	6
AlexNet	1232	308	1408	176

movement by keeping a portion of model updates local, but they double the network traffic on each interface as both worker and server processes communicate with remote hosts. Sharded approaches improve scalability but increase hardware costs and complexity.

Collective communication operations such as all-reduce offer an alternative to parameter servers, but they suffer from similar limitations in cloud environments. Each node in an all-reduce operation must process approximately twice the data compared to a non-colocated parameter server setup, and multi-round communication schemes introduce additional latency. Furthermore, collective operations lack the flexibility to implement sophisticated aggregation and optimization algorithms that can overlap computation with communication.

IV. RACKWEAVE ARCHITECTURE

A. System Overview

RackWeave rearchitects the parameter server as a rack-scale service that balances computational, memory, and network resources to eliminate communication bottlenecks in distributed training. The system comprises two key components: a software stack that optimizes gradient processing pipelines and a balanced hardware design that matches network bandwidth with memory and computational throughput.

At the software level, RackWeave introduces fine-grained key chunking, NUMA-aware buffer management, and streaming aggregation that overlaps gradient reception with processing. The architecture maps specific model chunks to dedicated cores and network interfaces, eliminating cross-core synchronization and promoting cache locality. This approach enables RackWeave to fully utilize high-speed networks such as InfiniBand and RoCE while minimizing software overhead.

At the hardware level, RackWeave employs a balanced design that matches network interface capacity with memory bandwidth and computational resources. Unlike traditional servers repurposed as parameter servers, which typically feature computational resources optimized for training rather than gradient aggregation, RackWeave's hardware configuration ensures that no single resource becomes a bottleneck during parameter exchange.

B. Hierarchical Deployment Model

RackWeave adopts a hierarchical deployment model that aligns with modern data center topology. As illustrated in Figure 1, each rack contains a dedicated RackWeave node that serves as a central aggregation point for workers within the same rack. This design leverages the full bisection bandwidth typically available within a rack while minimizing traffic through oversubscribed core networks.

For training jobs spanning multiple racks, RackWeave coordinates between rack-level instances to perform hierarchical reduction. This approach trades additional communication rounds for significantly reduced cross-rack traffic, particularly beneficial in cloud environments where inter-rack bandwidth is often oversubscribed. The system automatically determines when hierarchical reduction is beneficial based on network characteristics and model properties.

C. API Design and Framework Integration

RackWeave provides a comprehensive API that ensures compatibility with major deep learning frameworks including TensorFlow, MXNet, and PyTorch. The API includes standard Push and Pull operations for parameter synchronization, as well as a fused PushPull operation that combines push and pull phases into a single network round trip. This optimization is particularly valuable for synchronous training where push and pull operations typically occur consecutively.

Framework integration is achieved through lightweight shims that replace existing parameter exchange mechanisms without modifying the core training logic. This approach minimizes adoption barriers while providing immediate performance benefits. RackWeave's authentication and namespace isolation mechanisms enable secure multi-tenant operation, allowing multiple training jobs to share the same infrastructure without interference.

V. SOFTWARE OPTIMIZATIONS

A. Zero-Copy Network Stack

RackWeave implements a zero-copy network stack that eliminates unnecessary data movement between user space and kernel space. Leveraging RDMA capabilities of modern high-speed networks, the system directly transfers gradients between GPU memory and network interfaces, bypassing CPU involvement for data movement. On supported hardware, GPU-Direct RDMA further eliminates copies between GPU and host memory, reducing latency and CPU utilization.

The network stack employs NUMA-aware memory registration to minimize cache misses and improve memory access locality. By pre-allocating buffers in the NUMA domain where network interfaces reside, RackWeave reduces cross-NUMA traffic and improves memory bandwidth utilization. This optimization is particularly important for large models that exceed last-level cache capacities and rely heavily on main memory bandwidth.

Metadata overhead is minimized through clever encoding within existing RDMA fields. Callback identifiers and operation codes are embedded in queue pair numbers and immediate data fields, eliminating separate metadata messages and reducing PCIe transactions. This optimization improves bandwidth utilization for small messages that would otherwise be dominated by protocol overhead.

B. Streaming Aggregation and Optimization

RackWeave introduces a streaming aggregation approach that processes gradient chunks as they arrive, rather than waiting for complete layers. This fine-grained pipelining overlaps gradient reception with aggregation and optimization, effectively hiding processing latency behind network transfer time. The system employs tall aggregation, where threads work on the same chunk across all workers, rather than wide aggregation where threads process different parts of the same gradient array.

Tall aggregation provides several advantages over traditional approaches. First, it enables aggregation to begin immediately upon receipt of the first chunk, rather than waiting for complete layers. Second, it naturally balances computational load across available cores without complex synchronization. Third, it creates many independent "mini-queues" that prevent head-of-line blocking and improve overall system utilization.

Optimization operations such as SGD with momentum or Nesterov acceleration are performed immediately after aggregation completes for each chunk. By colocating aggregation and optimization on the same core, RackWeave maintains cache locality and eliminates inter-thread communication. This approach contrasts with traditional systems that use separate thread pools for aggregation and optimization, introducing synchronization overhead and cache pollution.

C. Fine-Grained Chunk Mapping

RackWeave partitions model parameters into fine-grained chunks typically sized at 32KB, significantly smaller than the multi-megabyte chunks used in traditional systems. This fine granularity enables better load balancing across cores and network interfaces, particularly for models with heterogeneous layer sizes. More importantly, it enables streaming processing that overlaps communication and computation.

Chunk-to-core mapping is determined during system initialization using a balanced partitioning algorithm that considers both computational load and network utilization. Each chunk is assigned to a specific core and network interface pair, ensuring that all processing for that chunk occurs on the same core throughout the training process. This affinity promotes cache locality and reduces synchronization overhead.

The mapping algorithm ensures balanced utilization of all system resources, including cores, network interfaces, and memory controllers. By avoiding hot spots and evenly distributing load, RackWeave maximizes throughput while maintaining low latency. The system continuously monitors resource utilization and can dynamically adjust chunk assignments if imbalances are detected during prolonged training sessions.

VI. Hardware Co-Design

A. Balanced Resource Allocation

Traditional servers repurposed as parameter servers suffer from inherent resource imbalances that limit performance. Typically featuring computational resources optimized for training workloads rather than gradient aggregation, these systems cannot fully utilize available network bandwidth due to memory or PCIe bottlenecks. RackWeave addresses this through a balanced hardware design that matches network interface capacity with memory bandwidth and computational resources.

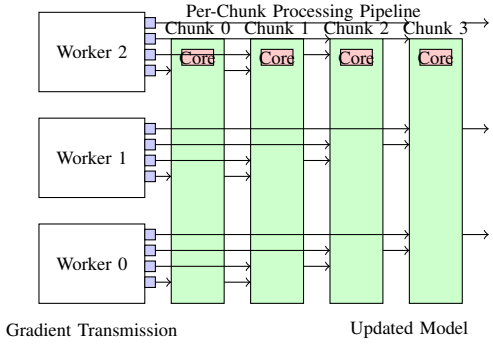

Fig. 2. RackWeave's fine-grained chunk processing architecture. Gradients from workers are partitioned into chunks and routed to dedicated cores that perform aggregation and optimization independently, enabling streaming processing and eliminating cross-core synchronization.

The RackWeave hardware configuration features multiple high-speed network interfaces that collectively saturate available PCIe bandwidth. Memory subsystems are sized to match aggregate network bandwidth, ensuring that gradient aggregation and optimization operations do not become memory-bound. Computational resources are optimized for the specific workload patterns of gradient processing rather than general-purpose training operations.

This balanced approach enables RackWeave to achieve significantly higher throughput than conventional servers when serving as parameter servers. By eliminating resource bottlenecks, the system can fully utilize available network bandwidth and support larger worker populations per rack. The design remains based on commodity components, ensuring cost-effectiveness and easy integration into existing data center environments.

B. NUMA-Aware Resource Placement

RackWeave carefully manages resource placement across NUMA domains to minimize cross-NUMA traffic and maximize memory bandwidth utilization. Network interfaces are assigned to specific NUMA nodes, with associated buffers and processing cores located within the same domain. This NUMA affinity reduces memory access latency and improves overall system performance.

Completion queues and queue pairs are exclusively used by cores within the same NUMA domain as their associated network interfaces. This strict partitioning prevents cross-NUMA coherence traffic and ensures that interrupt processing occurs on cores with local access to relevant data structures. The system further optimizes performance by pinning threads to specific cores, reducing context switching overhead and improving cache locality.

Memory allocation strategies are tuned for gradient processing workloads. Large contiguous buffers are pre-allocated during system initialization to avoid runtime allocation overhead and memory fragmentation. Buffer reuse is maximized to reduce pressure on memory allocators and improve cache effectiveness. These optimizations are particularly important for long-running training jobs that process terabytes of gradient data.

C. Scalability Considerations

RackWeave's scalability is ultimately bounded by available memory bandwidth, PCIe capacity, or network interface throughput. Our analysis identifies the PCIe-to-memory bridge as the limiting factor in current generation hardware, with practical throughput reaching approximately 90 GB/s despite theoretical limits exceeding 120 GB/s. This limitation underscores the importance of balanced design rather than maximizing any single resource in isolation.

The system employs several techniques to approach these hardware limits. Fine-grained chunking enables better utilization of parallel resources across multiple interfaces and memory controllers. NUMA-aware placement minimizes bottlenecks caused by asymmetric resource access. Streaming processing ensures that computational resources are fully utilized rather than sitting idle waiting for data.

For larger-scale deployments, multiple RackWeave instances can be coordinated to serve training jobs exceeding single-node capacity. The hierarchical reduction algorithms naturally extend to multi-node parameter servers, maintaining the benefits of rack-local aggregation while providing scalability beyond individual node limits. This approach maintains the performance benefits of the architecture while supporting the largest training workloads.

VII. Evaluation

A. Experimental Setup

We evaluated RackWeave using a testbed comprising 8 worker nodes and one dedicated RackWeave node. Worker nodes featured dual-socket Intel Xeon E5-2680 v4 processors, 64 GB of DDR4-2400 memory, NVIDIA GTX 1080 Ti GPUs, and Mellanox ConnectX-3 InfiniBand adapters with 56 Gbps bandwidth. The RackWeave node used a similar processor configuration with 128 GB of memory and 10 Mellanox ConnectX-3 adapters to balance network and memory bandwidth.

We implemented RackWeave support in MXNet by replacing its native parameter server implementation. All experiments used synchronous training with a per-GPU batch size of 32, except for larger models where memory constraints required smaller batches. We evaluated performance using several popular vision networks including ResNet, VGG, Inception, and AlexNet, measuring training throughput as samples processed per second.

We compared RackWeave against multiple baselines: (1) vanilla MXNet with colocated sharded parameter servers, (2) MXNet with InfiniBand-optimized network stack, (3) collective communication approaches using Gloo, and (4) compression-based optimization techniques. All experiments were conducted on both 10 Gbps cloud-like networks and 56 Gbps high-speed networks to evaluate performance under different bandwidth constraints.

B. Training Performance

RackWeave demonstrated significant performance improvements across all evaluated network architectures and configurations. On 10 Gbps networks, which represent typical cloud environments, RackWeave achieved up to 2.7x higher throughput compared to MXNet with colocated sharded parameter servers. The performance advantage was most pronounced for communication-bound models like AlexNet and VGG, where gradient exchange dominates training time.

On 56 Gbps networks, RackWeave maintained performance advantages for communication-intensive models while matching baseline performance for compute-bound workloads. This result demonstrates that RackWeave introduces no performance regression while providing significant benefits when communication bottlenecks exist. As GPU computational throughput continues to improve, we expect more models to become communication-bound, increasing the value of RackWeave's optimizations.

The performance benefits stem from multiple factors: reduced communication overhead through zero-copy operations, improved resource utilization through balanced design, and overlapping computation and communication through streaming processing. Breakdown analysis shows that RackWeave effectively shifts the training workload from communication-bound back to compute-bound, maximizing the utility of expensive GPU resources.

C. Scalability Analysis

We evaluated RackWeave's scalability using a communication-only benchmark that simulated infinitely fast computation, removing GPU processing as a bottleneck. This setup allowed us to isolate the parameter exchange subsystem and measure its maximum throughput. RackWeave demonstrated near-linear scaling with up to 8 workers, outperforming baseline approaches by up to 40x in this extreme scenario.

The scalability limit was determined by the PCIe-to-memory bridge throughput, which capped sustainable bandwidth at approximately 90 GB/s in our hardware configuration. RackWeave achieved 97% of this theoretical maximum, demonstrating highly efficient resource utilization. For typical training workloads, this limit far exceeds requirements – our measurements indicate that a single RackWeave node can support up to 120 workers training ResNet-50 with batch size 32.

Hierarchical reduction enabled efficient scaling across multiple racks without saturating oversubscribed core networks. The additional latency introduced by multi-round communication was more than compensated by reduced cross-rack traffic, particularly for large models. RackWeave automatically

TABLE II
THROUGHPUT COMPARISON (SAMPLES/SECOND) OF DIFFERENT DISTRIBUTED TRAINING APPROACHES ON 56 GBPS NETWORK WITH 8 WORKER NODES.

Approach	AlexNet	ResNet-50	VGG-19	Inception-V3
MXNet (CS PS)	634	688	513	410
MXNet + IB	721	712	598	435
Gloo All-Reduce	583	645	492	398
RackWeave	**1712**	**735**	**1387**	**448**

selected the optimal reduction strategy based on network characteristics and model properties, ensuring good performance across diverse deployment scenarios.

VIII. RELATED WORK

Parameter servers have been extensively studied as a mechanism for distributed machine learning. The original parameter server architecture [14] introduced a scalable framework for distributed parameter management, while subsequent work explored consistency models [13], system optimizations [10], and resource efficiency [20]. RackWeave builds upon these foundations but differs in its rack-scale approach and balanced hardware-software co-design.

Collective communication operations offer an alternative to parameter servers for distributed training. Frameworks like Baidu's Allreduce [8] and Uber's Horovod [19] implement optimized collective operations that can in some cases outperform parameter server approaches. However, as our evaluation shows, these approaches suffer from similar network bottlenecks in cloud environments and lack the flexibility for sophisticated aggregation strategies.

Gradient compression techniques aim to reduce communication volume rather than improving communication efficiency. Approaches such as 1-bit SGD [18] and deep gradient compression [15] can reduce bandwidth requirements but introduce computational overhead and may impact convergence. RackWeave is orthogonal to these techniques and can benefit from combined application.

System-level optimizations for distributed training have been explored in projects like GeePS [11], which focused on GPU memory management, and Poseidon [20], which optimized communication schedules. RackWeave complements these approaches by addressing the fundamental resource imbalances in parameter server deployments and providing a rack-scale service architecture.

Recent work on in-network computation [16] and smart network interfaces explores moving computation closer to data. While promising, these approaches require specialized hardware not yet widely deployed in cloud environments. RackWeave achieves similar benefits using commodity components and standard network protocols, making it immediately deployable in existing infrastructure.

IX. CONCLUSION

RackWeave demonstrates that communication bottlenecks in distributed AI training can be effectively addressed through a rack-scale parameter service that co-designs software and hardware. By treating gradient exchange as a first-class resource rather than a colocated afterthought, the system achieves significant performance improvements while maintaining compatibility with existing training frameworks.

The key innovations – fine-grained chunk processing, balanced resource allocation, NUMA-aware optimization, and hierarchical reduction – work together to shift the training workload from communication-bound back to compute-bound. This approach maximizes the utility of expensive computational resources while minimizing the impact of network constraints.

Experimental results confirm that RackWeave provides substantial throughput improvements compared to state-of-the-art approaches, particularly in bandwidth-constrained cloud environments. The system's cost-effectiveness further enhances its practical value, offering better performance per dollar than conventional deployments.

As AI models continue to grow in complexity and size, efficient distributed training will become increasingly important. RackWeave provides a practical path forward that balances performance, cost, and deployability, enabling continued progress in artificial intelligence without requiring fundamental changes to cloud infrastructure.

REFERENCES

[1] Bhatt, K., "Social Network Influence on Consumption Dynamics and Aggregate Variability," in *2025 International Conference on Computing Technologies Data Communication (ICCTDC)*, 2025, pp. 01-14.

[2] Bhatt, K., "Risk-Based Optimization Strategies for Secure and Efficient Digital Payment Systems," in *2025 International Conference on Information, Implementation, and Innovation in Technology (I2ITCON)*, 2025, pp. 1-6.

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

[4] Muneer, T. A. S., "Flexible and Scalable Customer Feedback Topic Extraction Using Gensim's Online LDA," in *2025 International Conference on Smart Sustainable Technology (INCSST)*, 2025, pp. 1-6.

[5] Muneer, T. A. S., "Leveraging Collective Intelligence in Agile Sprint Planning: A Comparative Study of LLM Architectures," in *2025 International Conference on Smart Sustainable Technology (INCSST)*, 2025, pp. 1-5.

[6] Muneer, T. A. S., "UrbanPlan-GPT: A Specialized AI Model for Human Settlement Design," in *2025 International Conference on Smart Sustainable Technology (INCSST)*, 2025, pp. 1-5.

[7] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *OSDI*, 2016.

[8] Baidu Research, "baidu-allreduce," GitHub repository, 2018.

[9] T. Chen et al., "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv:1512.01274*, 2015.

[10] T. Chilimbi et al., "Project Adam: Building an efficient and scalable deep learning training system," in *OSDI*, 2014.

[11] H. Cui et al., "GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server," in *EuroSys*, 2016.

[12] J. Dean et al., "Large scale distributed deep networks," in *NIPS*, 2012.

[13] Q. Ho et al., "More effective distributed ML via a stale synchronous parallel parameter server," in *NIPS*, 2013.

[14] M. Li et al., "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014.

[15] Y. Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv:1712.01887*, 2017.

[16] M. Liu et al., "IncBricks: Toward in-network computation with an in-network cache," *SIGOPS OSR*, 2017.

[17] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, 1988.

[18] F. Seide et al., "1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs," in *Interspeech*, 2014.
[19] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv:1802.05799*, 2018.
[20] H. Zhang et al., "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *USENIX ATC*, 2017.