

AI Fabrics Perspective and Modeling

Open Compute Project Foundation (OCP)

AI HW/SW Co-Design Workgroup

July 2026

Contributors (in the alphabetic order)

- Jascha Achterberg, University of Oxford
- Boon Chong Ang, Intel
- Andy Balogh, Keysight
- Eric Borch, HPE
- Kevin Cameron, Cameron EDA
- Pradosh Datta, Keysight
- Heshm ElBakoury, Independent
- Sakul Gupta, Micron
- Jack Harwood, Dell
- Prakash Jain, MIPS
- Jai Kumar, Broadcom
- Winston Liu, Keysight
- Victor Lu, Independent
- Kingshuk Mandal, Keysight
- Pankaj Mehra, Ohio State University
- Bijan Nowroozi, Lightmatter
- Harsh Sikhwal, Keysight
- Durgesh Srivastava, MIPS
- Siamak Tavallaei, Samsung
- Dan Wilkinson, Synopsis
- Weifeng Zhang, Lenovo Research & OCP AI Co-Design

Executive Summary

This paper establishes a vendor-neutral framework for understanding and designing AI datacenter infrastructure, focusing on the specialized network fabrics required for large-scale AI training and inference. It introduces standardized terminology and characterization methods for "**AI Fabrics**" - purpose-built networks that connect thousands of GPUs and accelerators with ultra-low latency and extreme bandwidth. The framework categorizes infrastructure into scale-up, scale-out, and scale-across networks, along with integrated distributed storage services.

Organizations deploying AI infrastructure often face critical challenges of selecting appropriate technologies from competing vendors, optimizing the balance between compute power and network capability, and preventing communication bottlenecks that waste expensive GPU resources. This framework enables informed decision-making through simulation and modeling before committing capital expenditure. Crucially, the framework extends beyond initial architecture to support continuous operational optimization, utilizing real-time simulation (Digital Twins) to predict performance and dynamically align resources with evolving AI workloads. The InfraGraph modeling tool provides a standardized way to describe, compare, and optimize infrastructure designs, reducing time-to-deployment and improving ROI through better resource utilization.

This paper serves as a reference with key insights for stakeholders in the data center ecosystem, including AI infrastructure architects and data center planners designing next-generation systems, cloud service providers and enterprises building large-scale AI training clusters, technology vendors developing AI accelerators, switches, and interconnect solutions, and research institutions exploring frontier AI model development. This paper also identifies several open challenges including cross-layer co-optimization of model architecture with hardware placement, multi-tenant workload scheduling on shared fabrics, scalable mapping algorithms for 10,000+ device systems, and automated co-design workflows accessible to practitioners. This paper explores the full spectrum of AI infrastructure, illustrating how co-design principles must be applied fractally, optimizing interconnects from the microscopic 'Scale-In' level of chiplets, through rack-level 'Scale-Up' domains, out to the 'Scale-Out' Ethernet fabrics that bind clusters together.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

As AI models continue scaling toward trillion-parameter systems, intelligent workload-to-fabric mapping will be essential for realizing full infrastructure potential.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Compliance with Open Compute Project Tenets

Openness

This paper promotes openness by establishing a vendor-neutral framework for understanding and designing AI datacenter infrastructure, focusing on the specialized network fabrics required for large-scale AI training and inference. It introduces standardized terminology and characterization methods for "**AI Fabrics**" - purpose-built networks that connect thousands of GPUs and accelerators with ultra-low latency and extreme bandwidth.

Efficiency

The paper establishes a framework to optimize AI infrastructure by enabling informed decision-making through simulation and modeling. The modeling tool via InfraGraph provides a standardized way to describe, compare, and optimize infrastructure designs, reducing time-to-deployment and improving ROI through better resource utilization.

Impact

This paper serves as a reference with key insights for stakeholders in the data center ecosystem. It promotes the abstract, implementation independent description and modeling which allow better defining the view of the “scale-up” and “scale-out” fabrics as well as interaction between them. Through AI infrastructure modeling / categorization and workload characterization, trade-offs are able to be made on the technologies and topologies required to meet AI application requirements.

Scale

This paper explores the full spectrum of AI infrastructure, illustrating how co-design principles must be applied fractally, optimizing interconnects from the microscopic 'Scale-In' level of chiplets, through rack-level 'Scale-Up' domains, out to the 'Scale-Out' Ethernet fabrics that bind clusters together.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Sustainability

This contribution supports sustainability of AI infrastructure through continuous operational optimization and real-time simulation to predict performance and dynamically align resources for better compute efficiency as well as data movement efficiency. It also addresses the evolution of the physical layer (e.g., shift from copper to silicon photonics) as a critical enabler to tackle the hard constraints of the rising power cost and scaling of electrical transmission.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Table of Contents

1. Introduction **8**
2. AI Workloads, Applications, and Infrastructure **9**
2.1 Key AI Categories and Models **9**
2.2 AI Workloads: Training vs. Inference **10**
2.3 Parallelization Strategies in AI Compute and Communication 11
2.4 Data Movement, Management, and Network Implications **15**
2.5 Management and Orchestration **16**
3. AI Fabric Network Infrastructure **16**
3.1 End-to-End Data Center Network Architecture with AI Fabric 17
3.2 AI Fabric Terminologies 19
3.3 AI Fabric Technologies 21
3.3.3 Next-Generation Fabric Protocols 26
3.4 Converged AI Fabric: Connecting Scale-Up to Scale-Out/Scale-Across 28
3.5 Networks, Traffic Management, and AI Fabrics 31
4. Data Movement **34**
5. Distributed Data Services: The Role of Storage in Large-Scale AI **37**
5.1 Integrating Distributed Data Services with AI Fabrics **40**
5.2 Synchronous & Latency-Sensitive Traffic **42**
5.3 Asynchronous & Less Latency-Sensitive Traffic **43**
6. AI Infrastructure Modeling: A Standardized Schema with InfraGraph **44**
6.1 Why InfraGraph Matters for AI Infrastructure Modeling 45
6.2 Introduction to the InfraGraph Schema 46
6.3 Steps to Model Systems and Infrastructures 47
6.4 Framework and Tooling 48
6.5 Operationalizing the Model: The Digital Twin 49
7. AI Workload Categorization & Mapping **50**
7.1 The Co-Design Challenges 50
7.2 Impact of Workload Diversity on Network Utilization 51
7.3 Essential Components for Workload Mapping 51
7.4 Mapping Methodologies 53
7.5 Open Directions 54

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

8. Conclusion **55**
9. Reference **57**
Appendix A **58**
Appendix B **61**
B.1 Introduction to Computer Networks 61
B.2 Network Traffic Management 62
License **65**
About Open Compute Project **66**

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

1. Introduction

With the continued evolution of the Artificial Intelligence (AI) ecosystem, we have seen the growth in models and the associated compute-communication paradigm shift to train these models. Under this paradigm, there is a need to investigate new topologies for best accommodating the model's compute-communication pattern/distribution among the connected compute engines, enabling new communication methods and operationalizing the AI applications to transform, move and manage data appropriately across the connected AI datacenter.

To date a number of AI fabrics topology, terminology and operations have been associated with vendor specific implementations. The goal in building perspective for AI fabrics is to provide an abstracted view of the terminology in the compute-communication paradigm and leverage this perspective to characterize each of the elements as we work from chip to system, to rack, pod, cluster and to datacenter. Since each workload generates different demand patterns on data transformation and movement, we need to characterize the AI workload to identify these patterns and co-design the overall topology to collaboratively deliver the service level required for the workload or application. From the lowest level "memory semantic" operations to the higher level "memory based" network operations data are orchestrated and resources scheduled to meet the overall application/user needs. For this, boundaries are defined and the characteristics for each boundary in the topology are enumerated, abstracted from specific implementations to allow for customers to choose their specific technology to deploy at any layer in the network topology. However, topology alone cannot solve the looming physical bottlenecks. As lane rates approach 224 Gbps and 448 Gbps, the 'shoreline' limit of chip packaging and the rising power cost of electrical transmission create hard constraints for scaling. This paper therefore addresses the evolution of the physical layer (PHY), such as the shift from copper to Silicon Photonics, as a critical enabler for the next generation of high-radix, disaggregated AI fabrics.

These characteristics offer twofold benefits:

- First, the characteristics can be incorporated into a simulation platform for topology exploration, allowing for the characteristics to be clearly highlighted and trade-offs on implementation choices identified.
- Second, with the rapidly increasing number of compute elements, model types and communication operations during new AI trainings, the characterization provides better understanding of synchronous/asynchronous boundaries for data transfer and memory semantic operations. In addition, the tight coupling of hardware coherency may be able to be relaxed, allowing for a software coherent model for a number of AI operations to train and operationalize AI models.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The abstract, implementation independent description and modeling allows a better definition of the view of the “scale-up” and “scale-out” fabrics as well as interaction between them. Through AI infrastructure modeling / categorization and workload characterization, trade-offs are able to be made on the technologies and topologies required to meet AI application requirements.

This paper is organized as follows: Section 2 categorizes AI workloads with various parallelism mechanisms. Section 3 details AI Fabric technologies. Sections 4 and 5 characterize data services in AI Fabric. Section 6 presents an AI Infra modeling mechanism. Section 7 shows AI workload patterns with mapping to AI Fabric. The paper is then summarized in Section 8.

2. AI Workloads, Applications, and Infrastructure

AI has moved beyond research labs to become a transformative force across countless industries. At its heart, AI involves using algorithms (models) trained on data to perform tasks that typically require human intelligence, such as understanding language, recognizing images, making predictions, or generating new content. Building and deploying AI solutions involves a complex interplay of algorithms, massive datasets, powerful computing resources, and increasingly sophisticated network infrastructure.

2.1 Key AI Categories and Models

- **Generative AI (GenAI):** A category of AI focused on *creating* novel content. This includes text, images, music, code, and synthetic data. GenAI models learn patterns and structures from training data and use this knowledge to generate new, original outputs.
    - **Large Language Models (LLMs):** A prominent type of GenAI specifically designed for understanding, processing, and generating human-like text. Models like GPT, Claude, Llama, etc., often contain billions or even trillions of parameters, enabling complex tasks like translation, summarization, question answering, and creative writing.

- **Deep Learning Recommendation Models (DLRM):** Specialized models designed for personalization. They analyze user features, item features, and past interactions (clicks, purchases) to predict user preferences and recommend relevant items, such as products on e-commerce sites, movies on streaming services, or content in social media feeds.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **Other Models:** AI encompasses many other model types, including Computer Vision models for image/video analysis, predictive models for forecasting, and anomaly detection models for identifying unusual patterns.

2.2 AI Workloads: Training vs. Inference

AI models are not useful out-of-the-box; they must be developed through distinct workload phases:

1. **Training:** This is the process of teaching an AI model by feeding the model vast amounts of data.
   1.1. **Process:** The model makes predictions on the data, a *loss function* measures the error in these predictions, and an *optimization algorithm* (like stochastic gradient descent or SGD) adjusts the model's internal parameters (weights) to minimize this error iteratively. This often requires thousands or millions of iterations over the dataset.
   1.2. **Characteristics:** Highly compute-intensive. Require powerful GPUs (Graphics Processing Unit) or specialized accelerators, demands large storage for datasets, involves significant data movement (ingesting data, reading batches, checkpointing model state), and is sensitive to computation time.
   1.3. **Parallelism:** Due to the scale of models (especially LLMs) and datasets, training is almost always distributed across multiple accelerators using strategies like Data Parallelism (replicating the model, splitting data), Model Parallelism (splitting the model itself across devices), Pipeline Parallelism (staging model computations), and Tensor Parallelism (splitting operations within layers). The choice of parallelism directly impacts the communication patterns (All-Reduce, point-to-point transfers) and network requirements. More details on parallelization strategies and use cases are listed in Section 2.3.
   1.4. **Workload Profile:** Characterized by large, often bursty data transfers, potentially complex collective communication patterns, high computational demands, and sensitivity to network bandwidth and latency for efficient scaling.

2. **Inference:** This is the phase where a *trained* model is used to make predictions on new, unseen data.
   2.1. **Process:** Input data is fed into the trained model, which performs a forward pass to generate an output (a prediction, classification, generated text, recommendation score, etc.).
   2.2. **Characteristics:** These vary widely depending on the application. Real-time inference often prioritizes low latency, while batch inference may

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

prioritize throughput (predictions per second). Data movement per inference request is typically much smaller than during training, but there can be millions of concurrent requests.

2.3. **Workload Profile:** Ranges from latency-sensitive, single-request processing to high-throughput batch processing. Network needs depend heavily on the deployment model (e.g., requests coming over the internet vs. internal processing).

3. **Edge AI**: Bringing AI Closer to the Source - Edge AI refers to deploying and running AI models directly on end-user devices (smartphones, cameras, sensors) or local edge servers, rather than relying solely on centralized cloud data centers.
   3.1. **Motivations:** Reduces latency for real-time applications, saves network bandwidth by processing data locally, enhances privacy/security by keeping data local, and enables operation in disconnected environments.
   3.2. **Use Cases:** Real-time object recognition in smart cameras, voice assistants on smartphones, predictive maintenance alerts directly from industrial equipment, or autonomous vehicle decision-making.
   3.3. **Challenges:** Edge devices have limited compute, memory, and power resources, requiring highly optimized models (e.g., using quantization, pruning). Managing, deploying, and updating models across potentially thousands of distributed edge devices are complex. Though Edge AI does computation locally or with a much simplified AI fabric, it may involve heterogeneous computing via discrete GPUs / accelerators for more power-efficient computing. Similarly, it may leverage optimizations via hierarchical KV cache to reduce redundant computation for faster response or running larger models. Thus, AI fabric-like technologies are still relevant to Edge AI.

2.3 Parallelization Strategies in AI Compute and Communication

Training and running large-scale AI models, particularly deep learning models, often involves computations and memory requirements that exceed the capabilities of a single processing unit like a GPU. To overcome these limitations and accelerate the process, **parallelism** techniques are employed. These strategies distribute the computational workload and data across multiple devices (e.g., GPUs within a server or across multiple servers connected by a network).

The choice of parallelism strategy profoundly impacts how computation is divided and, crucially, the **communication patterns** required between the devices. Understanding these strategies is key to designing efficient AI systems and the underlying network fabrics that support them. As discussed in the section above, networks designed with a

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

specific focus on the communication patterns of AI workloads are also referred to as AI Fabrics.

Below is an overview of the common forms of parallelism [1,4] used in AI which determine the observed communication pattern:

Data Parallelism (DP)

- **Concept:** This is one of the most common strategies. The core idea is to replicate the *entire* AI model onto multiple devices. The large training dataset is then split into smaller chunks or mini-batches.
- **Mechanism:** Each device processes a different mini-batch of data using the local copy of the model parameters on that device. They compute the forward pass (generating predictions) and backward pass (calculating gradients – the updates needed for the model parameters) independently.
- **Communication:** After processing their mini-batch, all devices need to synchronize the calculated gradients. This typically involves an **All-Reduce** communication operation, where gradients from all devices are aggregated (e.g., averaged), and the updated gradients are distributed back to all devices so they can update their model copy consistently before processing the next mini-batch.
- **Use Case:** Effective when the model fits into the memory of a single device, but the dataset is very large, and computation time needs reduction.
- **Trade-offs:** Relatively straightforward to implement. Scales well if computation per batch is significantly larger than communication time. However, it is limited by the memory capacity of a single device (since each device holds the full model) and the All-Reduce step can become a bottleneck as the number of devices increases, demanding high network bandwidth and low latency.

Model Parallelism (MP) - General / Inter-Layer

- **Concept:** When a model is too large to fit into the memory of a single device, Model Parallelism splits the model across multiple devices.
- **Mechanism:** Different layers or components of the model are placed on different devices. Each device holds and computes only a *portion* of the model parameters. During the forward pass, the output (activations) from a layer on one device is passed as input to the next layer, potentially residing on another device. During the backward pass, gradients are passed in the reverse direction.
- **Communication:** Requires direct point-to-point communication between devices holding adjacent parts of the model to exchange activations (forward pass) and gradients (backward pass).
- **Use Case:** Necessary for models that exceed single-device memory.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **Trade-offs:** Allows training of extremely large models. However, it can be complex to partition the model effectively which often leads to lower hardware utilization, as some devices may wait for results from others. Communication overhead can be significant if layers are split inefficiently.

Pipeline Parallelism (PP)

- **Concept:** A more sophisticated form of model parallelism that aims to improve hardware utilization compared to basic model splitting. The model is divided into sequential stages, and these stages are placed on different devices.
- **Mechanism:** The input data (often split into micro-batches) is fed into the first stage. Once the first stage completes processing the first micro-batch, it passes the output to the second stage. Simultaneously, the first stage starts processing the *next* micro-batch. This creates a pipeline where multiple micro-batches are processed concurrently across the different stages/devices.
- **Communication:** Similar to basic Model Parallelism, activations are passed forward between stages, and gradients are passed backward. The pipelined nature allows for overlapping computation and communication.
- **Use Case:** Training very deep models where basic model parallelism might lead to significant idle time.
- **Trade-offs:** Can significantly improve device utilization and hide communication latency compared to naive model parallelism. However, PP introduces a "pipeline bubble" – periods at the start and end of processing where not all devices are active. Performance is limited by the slowest stage, requiring careful load balancing across stages.

Tensor Parallelism (TP) - Intra-Layer

- **Concept:** This strategy splits the computations *within* a single layer, often within large matrix multiplications or tensor operations common in transformer models, across multiple devices.
- **Mechanism:** Instead of placing entire layers on different devices (like MP/PP), TP splits the weight matrices and intermediate activation tensors of a specific layer. Each participating device computes only a part of the layer's operations. Results are then combined through communication. For example, a large matrix multiplication might be split so each device computes a portion of the output matrix.
- **Communication:** Requires frequent and high-bandwidth communication *within* the group of devices participating in the tensor parallel computation for a given layer. This often involves collective operations like All-Reduce or All-Gather specific to the mathematical operation being split.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **Use Case:** Useful for layers with massive parameters or activations that don't fit on a single device or when seeking to accelerate specific large operations. Often used *within* a node where GPUs are tightly connected (e.g., via NVIDIA's NVLink®).
- **Trade-offs:** Can reduce memory requirements per device for specific layers and potentially speed up computation for very large tensors. However, TP introduces significant communication overhead within the layer's computation, making it most effective with very high-speed, low-latency interconnects.

Expert Parallelism (EP):

- **Concept**: Expert Parallelism (EP) is a strategy to distribute individual "expert" neural network modules within a **Mixture-of-Experts** (MoE) model across different GPUs/compute nodes, rather than replicating the whole model.
- **Mechanisms**: It works by splitting the experts (often feed-forward networks) across devices, with a router directing each token to its selected expert(s) on specific GPUs, enabling massive model scaling.
- **Use Case**: Large MoE models. It allows larger models to fit in GPU memory and run efficiently by routing tokens to only the necessary experts, significantly reducing computation and memory load.
- **Trade-offs**: Expert routing creates dynamic, input-dependent communication patterns. The expert parallelism strategy introduces unique traffic that may benefit from in-network aggregation.

Hybrid Parallelism and Compute-Communication Interplay

In practice, especially for state-of-the-art large language models (LLMs) or complex vision models, these strategies are often combined. A common approach is hybrid parallelism:

- **Pipeline Parallelism (PP)** might be used *between* groups of devices (nodes).
- **Tensor Parallelism (TP)** might be used *within* a node across GPUs connected by high-speed links like NVLink.
- **Data Parallelism (DP)** might be applied *across* multiple pipelines or replicas.

The choice and combination of parallelism strategies directly dictate the demands placed on the underlying communication infrastructure (the AI Fabric).

- **DP** requires efficient All-Reduce across potentially many nodes.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **MP and PP** require efficient point-to-point communication between specific devices holding adjacent model parts or stages.
- **TP** demands extremely high bandwidth and low latency, usually within a tightly coupled set of devices.

Therefore, designing an effective AI system involves co-designing the parallelism strategy and the network fabric to ensure that communication does not become the primary bottleneck, allowing the powerful compute capabilities of modern accelerators to be fully utilized.

2.4 Data Movement, Management, and Network Implications

To complete the training or inferencing actions, complex pipelines are needed for data collection, cleaning, preprocessing, and feeding into training or inference systems.

- **Training Data Movement:** Involves transferring massive datasets to compute clusters, exchanging gradients/activations between nodes during distributed training (governed by parallelism strategy), and frequently saving model checkpoints for model versioning and/or fault tolerance (to quickly recover from the interrupted training process due to device or software errors).
- **Inference Data Movement:** Involves sending inference requests (input data) to the model and receiving predictions back. For Edge AI, this might be minimal, while cloud-based inference handles traffic from many users. In the case of GenAI with LLMs, inference can involve huge repetitive computation to generate keys and values from the input during the inference prefill phase. A common optimization is to store these keys/values in the cache, called KV Cache, and reuse the results during the decoding phase to accelerate the inference. Thus, this optimization introduces massive extra data movement besides the input data. Similarly, inference with RAG (Retrieval Augmented Generation) also introduces extra data movement in enterprise-level AI due to the input of enterprise domain-specific knowledge or experience.
- **AI Fabrics:** The intense communication demands of distributed training, particularly the need for high bandwidth, ultra-low latency, and lossless operation (to avoid performance degradation), have driven the development of specialized network infrastructures known as AI Fabrics. These often utilize technologies like InfiniBand® or Lossless Ethernet (using PFC, ECN/DCQCN) with Remote Direct Memory Access (RDMA), advanced load balancing (ECMP variants, adaptive routing, flowlet switching), and optimized topologies (e.g., Fat-Tree) to handle demanding AI workload profiles efficiently. More architectural details of AI Fabric are explained in Section 3.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

2.5 Management and Orchestration

The complexity of the AI lifecycle requires robust management tools and platforms (often part of MLOps - Machine Learning Operations):

- **Data Management:** Tools for data storage, versioning, retrieval, and processing pipelines.
- **Model Management:** Frameworks for tracking experiments, versioning models, and managing deployment.
- **Resource Orchestration:** Platforms like Kubernetes (with GPU operators) or Slurm schedule AI jobs, manage compute resources (CPUs, GPUs), and handle tidal variation of resources at scale.
- **Workload Orchestration:** As workloads are becoming increasingly complex and heterogeneous in terms of their dependency on computational resources (compute, memory, networking) and usage of operations, execution engines make the schedule when and in which order parts of the workload are to be executed across the duration of a compute job's execution.
- **Network Management:** Monitoring and configuring the network, especially critical for tuning high-performance AI fabrics to optimize data movement and prevent congestion.

The AI landscape is characterized by diverse workloads, ranging from compute-heavy, data-intensive training of massive models like LLMs and DLRMs to latency-sensitive inference tasks deployed in the cloud or at the edge. Understanding the specific workload profiles, applications, and associated data movement patterns is crucial. Efficiently handling these demands requires not only powerful compute but also sophisticated management practices and highly optimized network infrastructure, increasingly relying on specialized AI Fabrics to bridge the gap between distributed computation and communication, ultimately enabling the development and deployment of cutting-edge AI solutions.

3. AI Fabric Network Infrastructure

Large-scale distributed AI training and inference often make standard Ethernet networks (see details in Appendix B) insufficient to meet the extreme demands of high throughput, ultra low latency, and lossless performance. This section shows the necessity of purpose-built network infrastructures, AI Fabric, for AI workloads.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

3.1 End-to-End Data Center Network Architecture with AI Fabric

A. Front-end Network (Existing Data Center Fabric)

This is the part of network handling tasks that connect users, datasets, and services to GPU and AI accelerator resources. Its role is critical for seamless data flow and user interaction in AI-driven environments:

- User access to GPU and AI accelerator resources
- Dataset ingestion and preprocessing
- Model serving and inference requests
- Management and monitoring traffic
- Inter-service communication

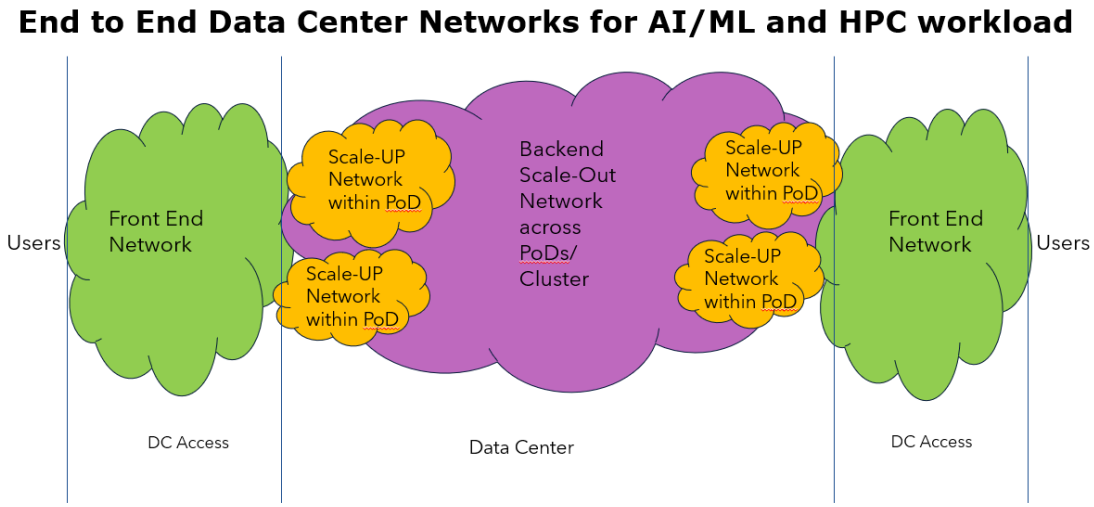


Figure 1. the end-to-end network architecture

The front-end network's quality of service (**QoS**) must adapt to the varied demands of AI workloads while maintaining flexibility and reliability:

- Flexible Bandwidth Allocation
- Support for burst traffic (up to 10x baseline)
- Standard TCP/IP optimization techniques

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- Multiple traffic classes (e.g., management, user data, inference)

Example use case for front end network: A machine learning platform handles traffic for 10,000 users, from data scientists uploading training datasets to services requesting real-time model inference. The front-end network ensures smooth and efficient handling of such diverse demands.

B. Back-end Networks (AI Fabric)

The back-end network, also known as the AI fabric, is the cornerstone of high-performance inter-device communication for AI-driven data centers. Its architecture supports the unique demands of distributed AI workloads, such as training and inference. To optimize performance, the back-end network is divided into specialized segments:

- **Scale-In Network (Die-to-Die):** Focuses on ultra-short-range communication within a single package or 'superchip' utilizing protocols like UCIe®. Crucially, this layer provides an abstraction that enables a modular 'chiplet' architecture. This allows designers to rapidly evolve specific functional blocks (e.g., swapping in next-gen AI compute tiles while retaining standard I/O) and permits software to treat these heterogeneous assemblies as a unified resource, significantly accelerating the hardware innovation cycle.

- **Scale-Up Network:** Focuses on accelerator-to-accelerator communication within a PoD (up to ~1,024 accelerators). Leverages direct load/store memory access for ultra-low latency operations. Emerging architectures extend this coherency beyond the server chassis to the rack level and beyond; examples include NVIDIA's NVL72, which connects 72 GPUs into a single rack-scale domain, and larger implementations like NVL576 or UAL1024, which aim to scale these unified memory domains across multiple racks to function as a single massive accelerator.

- **Scale-Out Network**: Facilitates communication across PoDs/clusters of thousands of accelerators. Utilizes RDMA for efficient data transfer across PoDs or accelerators.

- **Scale-Across Network**: Facilitates communication across Data Centers.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

At present, the typical technical requirements include low latency: <1 µs (Scale Up) and < 10 µs (Scale Out/Scale-Across), high radix connectivity, and guaranteed bandwidth up to 800 Gbps per device, lossless transmission.

Below is list of some of its key functions and features:

- Distributed memory sharing
- Gradient Aggregation for large language models
- Collective communications in Multi-Node AI training
- RDMA *and RMA (direct load/store)* operations requiring guaranteed delivery
- Precise congestion control

Back-end Network QoS Specification requirements include:

- Guaranteed Bandwidth: ~400/800 Gbps per device
- Low Latency: ~2 µs
- Lossless Transmission
- Predictable performance under loaded conditions

Example use case for backend/AI Fabric networks: Training a 175 billion parameter model across 1,024 GPUs or more involves handling approximately 350 GB of gradient data per training iteration. Delivering deterministic latency and bandwidth to ensure synchronized performance.

3.2 AI Fabric Terminologies

The Evolution of Compute Architecture Technologies for expanding computing capacity have evolved from centralized processing to massive distributed processing. This trajectory has shifted system design from classical CPU architectures (scalar, multi-core, SMP) to heterogeneous architectures (servers with Domain-Specific Accelerators or DSAs). Today, this evolution extends to rack-scale systems and multi-node clusters, connecting tens of thousands of nodes across large physical distances.

Defining Scale-Up vs. Scale-Out Within a server or rack, DSAs are often connected via specialized "southbound" interconnects (e.g., proprietary NVLink/NVSwitch®, or

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

open standards like Ultra Accelerator Link (UALink®) and ESUN). These bypass the CPU to remove performance bottlenecks and are classified as **Scale-Up** architectures. The defining characteristic of Scale-Up is the support for memory semantics (coherency) with high throughput and nanosecond-scale latency.

Scale-Out, by contrast, describes connections where the interconnect typically does not support direct memory semantics and operates at higher latencies (around 10s of microseconds). Standard network protocols, such as Ultra Ethernet or InfiniBand®, are the primary examples here. The boundary between Scale-Up and Scale-Out is defined by this capability to share memory space effectively across distance.

A. Definitions

1) **Accelerator**: Specialized compute components designed to accelerate AI workloads. Examples include: GPU (Graphics Processing Unit), TPU (Tensor Processing Unit), NPU (Neural Processing Unit)
2) **PoD** (Point of Delivery / Performance-Optimized Datacenter): A modular unit within a data center that integrates compute, storage, and networking resources to deliver localized services. PoDs are designed for scalability and performance optimization.
3) **AI Fabric/Back-end Network**: The cornerstone of high-performance inter-device communication for AI-driven data centers. Its architecture supports the unique demands of distributed AI workloads, such as training and inference.
4) **Scale-Up Network**: A segment of the data center backend network optimized for intra-PoD communication, typically among up to 1,024 accelerators. It supports ultra-low latency operations through direct load/store memory access semantics between accelerators.
5) **Scale-Out Network**: A segment of the data center backend network designed for inter-PoD communication across clusters of thousands of accelerators usually within the same Data Center or within the same Network Domain. It leverages RDMA for efficient, high-throughput data transfer between distributed compute nodes/PoDs.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

6) **Scale-Across Network**: Same as Scale-Out Network except it spans across multiple data centers or multiple network domains.
7) **INCC** (In-Network Collective Communication): A technique that offloads collective communication operations (such as AllReduce, Broadcast, Gather/Scatter, and AllToAll) to network switches to reduce data movement and improve performance. These operations are essential for synchronizing data across multiple compute nodes during distributed AI training.

B. Comparison of Scale-Up & Scale-Out/Scale-Across

Features	Scale-up	Scale-out/Scale-Across
Interconnect protocol	Memory semantics (Load/Store)	Network semantics (RDMA)
Throughput/Bandwidth	100s of Tbps	100s of Gbps
Latency	10s-100s of ns	10s-100s of us
Scope	Within node/pod	Across pods/clusters
Example Technologies	NVLink, UALink, SUE, CXL®, PCIe®, ESUN	Ethernet, InfiniBand, UEC®

Note that there have been a lot of parallel efforts in the community to enhance the existing interconnect protocols (e.g. ESUN) with new data semantics and boost throughput and/or reduce latency. Thus, the definitions of scale-up and scale-out may get a bit blur in the future.

3.3 AI Fabric Technologies

While Section 3.2 defined the AI Fabric by its *logical* role (connecting accelerators), this section addresses its *physical* reality. An AI Fabric is ultimately a massive system of I/O lanes, and today, those lanes are hitting a hard physical wall.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The Physics vs. Protocol Challenge: In standard server and rack architectures, components rely on **Physical Layer (PHY)** connections to exchange data. Historically, these have been copper-based electrical links. However, the physics of copper imposes a harsh penalty: as data rates increase, the effective signal reach dramatically decreases. Overcoming this requires complex equalization and power-hungry retimers, expensive countermeasures that eventually yield diminishing returns.

This physical limitation directly conflicts with the goals of modern interconnect standards like **PCIe, UCIe®, and CXL**. While these standards define both a protocol and a physical layer, their traditional copper-based PHYs cannot scale to the distance and density required for massive AI clusters.

To maintain the **upper-layer protocol semantics** (load/store, coherency) without being constrained by the **standard electrical PHY**, architects are decoupling the two—using Silicon Photonics as the physical transport for these established protocols.

C. The Geometric Bottleneck: SerDes and the Shoreline Limit

Inside the fundamental building block of high-speed I/O, **SerDes** (Serializer / Deserializer) is the foundational "PHY" component responsible for converting parallel on-chip data into high-speed serial streams for transmission.

SerDes acts as the universal engine supporting modern interconnects, from PCIe and Ethernet to proprietary fabrics like NVLink and Infinity Fabric. By standardizing this block, the industry allows component designers to focus on core logic rather than reinventing communication interfaces for every generation**.**

The Shoreline Problem: While SerDes speeds have scaled impressively to 224 Gbps, the challenge has shifted from speed to space. High-performance ASICs (XPUs and Switches) are limited by their "Shoreline" (or Beachfront), the physical perimeter of the chip die available for I/O pins.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

As transistor density (Area) scales quadratically, the available perimeter (Shoreline) only scales linearly. We are rapidly approaching a point where we physically cannot fit enough copper pins along the edge of the chip to feed the massive compute capacity of the core. This "Shoreline Bottleneck" forces designers to make impossible trade-offs between bandwidth, power, and complexity to maintain signal integrity.

The Photonic "Escape Valve": This is where Silicon Photonics fundamentally changes the equation. By moving the optical engine from the front panel to the package (Co-Packaged Optics), the electrical shoreline limitation can be effectively bypassed and any pins to/from the logic IC can be used for I/O. When the shoreline is no longer an architectural constraint, photonic I/O offers significantly higher bandwidth density (Gbps/mm) than electrical SerDes.

Furthermore, optical waveguides can escape the package with a fraction of the physical cross-section of copper bundles. This allows us to scale SerDes capacity far beyond the limits of electrical pins and lanes, enabling next-generation switch radices (e.g., 102.4T and beyond) that would be physically impossible to route with copper.

D. The Physical Layer Evolution: Silicon Photonics

The Evolution of the Copper PHY: As we push for higher bandwidth density, the physical layer (PHY) is undergoing a rapid evolution to squeeze the final capabilities out of copper. The industry has standardized on PAM4 (Pulse Amplitude Modulation 4-level) for 224 Gbps per lane SerDes to double the data rate compared to NRZ (Non-Return-to-Zero) without doubling the bandwidth requirement.

Looking forward, the path to 448 Gbps per lane is being charted, with standardization details currently under active discussion within the OIF (Optical Internetworking Forum). However, this speed increase imposes severe penalties on both signal integrity and power efficiency. To maintain signal quality over copper at these rates, complex

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

equalization and retiming are required, which drastically increases the power consumption per bit transferred.

While innovations like Co-Packaged Copper (CPC) help by minimizing the distance between the ASIC and the front panel, they only delay the inevitable. We are approaching a "power wall" where the energy required to drive electrical signals across the fabric (measured in picojoules per bit) begins to consume a disproportionate share of the total cluster power budget. This convergence of limited physical reach and unsustainable power scaling necessitates a fundamental shift in the transport medium itself.

Scaling, Complexity, DSPs, and the Path to SiPh: To understand the trajectory of AI fabrics, we must trace the evolution of the interconnect link itself, specifically the trade off between reach, capacity, and complexity.

- **Pluggable Copper (DAC):** The baseline for short-reach connectivity (in-rack). It is passive, simple, and low-power, relying on the host SerDes. However, physics strictly limits its reach and gauge at high speeds (224G+).
- **IM-DD Optics (The "DSP Tax"):** To extend reach for Scale-Out, the industry moved to Intensity Modulation-Direct Detection (IM-DD) optics. While effective, these modules typically require power-hungry **Digital Signal Processors (DSPs)** to recover signals, adding significant power and latency penalties to every hop.
- **Coherent Lite:** As we scale to **Scale-Across** architectures (campus/metro distances), simple IM-DD is lacking. The industry is adopting "Coherent Lite" technologies (derived from telecom) to maximize spectral efficiency. While powerful, coherent detection introduces even higher complexity and power consumption per bit.

The Silicon Photonics Simplification This escalating complexity creates an unsustainable power trajectory. Silicon Photonics (SiPh) disrupts this curve by integrating optical functions directly into silicon.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

This integration enables **Linear Drive Pluggable Optics (LPO)**, an architecture that removes the DSP entirely from the module for short-reach links, returning to the simplicity of copper but with the reach of optics. Ultimately, SiPh allows for **Co-Packaged Optics (CPO)**, where the optical engine is integrated with the ASIC, minimizing the electrical channel and drastically reducing the complexity required to drive high-bandwidth signals.

However, while it solves the power problem temporarily, LPO does not solve the **'Beachfront' density problem**. As switch bandwidths double again (to 102.4T and 204.8T), the physical faceplate area, not just the power, becomes the hard limit, necessitating the eventual move to Co-Packaged Optics.

Silicon Photonics: An Architectural Unlock: By integrating optical engines directly with the compute or switch ASIC (Co-Packaged Optics), SiPh moves beyond simple cable replacement to offer unique architectural advantages:

- **Higher Radix Switching:** The superior bandwidth density of photonics allows for a massive increase in I/O capacity per chip . This enables the construction of larger radix switches, which in turn allows for flatter network topologies with fewer hops and lower end-to-end latency.

- **Efficiency at Scale:** addressing the "power wall" described above, photonics bypasses the need for power-hungry retimers. It maintains signal integrity over the longer reaches required for **Scale-Out** and **Scale-Across** architectures without the exponential power penalty associated with copper.

- **Resource Disaggregation:** Ultimately, the reach and density of SiPh enable the physical disaggregation of compute, memory, and storage resources . This allows for 'rack-scale' or 'row-scale' architectures where resources are pooled and composed dynamically, breaking the rigid confines of the server chassis.

Beyond Electrical Switching: Optical Circuit Switches (OCS): While Silicon Photonics addresses the efficiency of *moving* data, the industry is also confronting the limits of *switching* data electrically. Traditional Electrical Packet Switches (EPS) convert

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

optical signals to electrical, process headers, buffer packets, and re-transmit—a process that consumes massive power and adds latency.

Optical Circuit Switches (OCS) break this barrier by switching light directly (typically via MEMS mirrors or advanced non-linear materials) without any O-E-O conversion.

- **Protocol Agnostic:** Because OCS switches pure light, it is transparent to the data rate and protocol. It scales seamlessly from 800G to 1.6T and beyond without requiring a hardware upgrade, carrying Ethernet, UALink, or any other signal with equal ease.
- **Dynamic Topology:** Unlike fixed electrical cabling, OCS allows the network topology to be reconfigured in software. This makes it the physical actuator for the **Digital Twin** (discussed in Section 6.5); the orchestrator can analyze a workload and physically "rewire" the cluster, changing from a hierarchical tree to a high-dimensional torus, to match the specific traffic pattern of a massive training job.

3.3.3 Next-Generation Fabric Protocols

With the physical layer constraints addressed by SiPh, specific protocols can be designed to run on top of a high-performance PHY fabric.

E. UALink for Scale-up Networking

UALink is an open, standardized interconnect designed for direct accelerator-to-accelerator communication, focused on memory sharing among GPUs. It supports simple load, store, and atomic operations with software-managed coherency, enabling low-latency, high-bandwidth connectivity for hundreds to thousands of accelerators within a pod. Built on the proven Infinity Fabric protocol, UALink delivers Ethernet-level speed with PCIe-like latency while reducing power and complexity through efficient switch design, small packets, fixed FLIT sizes, and ID-based routing. Its compact link stack lowers die area and acquisition costs, while improved bandwidth

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

efficiency reduces total cost of ownership. By having an option of Ethernet at PHY level, it leverages existing Ethernet infrastructure for cables, connectors, and management tools. UALink fosters interoperability and innovation through collaboration among member companies.

F. Scale-Up Ethernet (SUE)

The intent of Scale Up Ethernet (SUE) is to provide the transport and ethernet datalink to move memory transactions among Accelerators/xPUs. SUE provides a framework to provide low-latency, high-bandwidth connectivity for accelerators in scale up networks based on Ethernet.

G. Ethernet for Scale-up Networking (ESUN)

Proposed recently at the OCP Global Summit, ESUN is a complementary, open standard for AI interconnect to address/accommodate different technologies (UAlink, SUE, NVLink etc) in the networking stack for **"scale-up"** connectivity (within a single cluster/pod of accelerators). Its focus is to define network functionality at the L2/L3 (Link/Network) layers, including switching, framing, and reliable transport mechanisms over standard Ethernet. The goal is to provide the physical and link-level foundation such that protocols like UALink and SUE can utilize for broad interoperability, including accelerator / xPU network interfaces and Ethernet switch ASICs.

ESUN will actively engage with organizations such as UEC (Ultra-Ethernet Consortium) and IEEE 802.3 (Ethernet) to align with open standards, incorporate best practices, and accelerate innovation. By leveraging Ethernet's mature hardware and software ecosystem, ESUN will encourage diverse implementations and drive rapid adoption across the industry.

ESUN is one of the exemplary interconnect technologies which make the boundaries between scale-up and scale-out get blurrier.

H. Ultra Ethernet (UE) for Scale-Out/Scale-Across Networking

The Ultra Ethernet Consortium (UEC) is enhancing Ethernet’s capabilities through a collaborative, standards-driven approach, with the goal of standardizing a high-performance Ethernet stack purpose-built for the unique demands of AI & HPC.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

UEC provides an open, interoperable, and scalable Ethernet-based Scale-Out fabric that enables data-intensive applications to fully utilize compute and network resources.

There are many working groups within UEC focusing on specific parts of this technology, but the most important deliverable for UEC is the specification of a new transport protocol. This new protocol provides the ability to deliver data straight from the network and into application memory and vice versa, without software involvement, using RDMA. Ultra Ethernet Transport (UET) protocol incorporates multiple elements that distinguish it from any RDMA protocol that has come before.

3.4 Converged AI Fabric: Connecting Scale-Up to Scale-Out/Scale-Across

Converging scale-up, scale-out and scale-across back-end networks with simplified, unified management would be a game-changing approach for modern AI data centers with GPU/accelerator PoDs (collection of server nodes). By tailoring intra-PoD (scale-up) and inter-PoD (scale-out) architectures to AI's unique demands, organizations can unlock transformational improvements in speed, efficiency, and reliability with adaptability to different use-cases. In large-scale deployments, converged network strategy optimizes performance for AI/ML workloads, ensures predictable QoS, enhances security, and lays the groundwork for next-generation AI infrastructure. Crucially, this convergence is being driven by the standardization of Ethernet as the universal medium. By adopting Ethernet-based physical layers and protocols (such as UEC for Scale-Out and UALink/ESUN for Scale-Up), infrastructure architects can build these distinct network domains upon a common physical foundation, streamlining supply chains and enabling a unified management plane.

A. Unified Back End AI Fabric Implementation Strategy

Designing an efficient and scalable back-end network requires meticulous consideration of hardware and software components to meet the demands of AI workloads. Below are key aspects of the flexible implementation with converged Scale-In, Scale-up & Scale-out AI Fabric Technology:

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- To fully exploit modular 'Scale-In' architectures, the software stack, specifically compilers and runtime schedulers, must evolve to be 'topology-aware' at the package level. While protocols like UCIe® abstract the physical connection, intelligent software should optimize tensor placement to minimize cross-die traffic, treating the chiplet assembly not just as a single device, but as a micro-network of compute resources.

- High-Performance Interconnects Options: Leverage RDMA/RMA, HDR/NDR, RoCE v2 etc., to achieve high bandwidth and low latency for AI workloads. Configure RDMA for high-speed, low-latency data transfer with fine-tuned flow control mechanisms.

- Topology Design: Use non-blocking CLOS or single-layer topologies to maximize data throughput and minimize congestion as needed.

- AI Fabric Technology Options: Employ dedicated Ethernet-based or PCIe-based switches optimized for GPU and AI accelerator traffic as needed.

- Converged Routing: Use efficient IP routing algorithms or direct routing techniques per destination to streamline data movement across nodes.

- Load Balancing: Deploy Equal-Cost Multi-Path (ECMP) or packet spraying to distribute traffic evenly and prevent bottlenecks.

- Flow Control Options: Implement PFC (Priority-based Flow control) or credit-based flow control (CBFC) as needed to avoid packet loss during high traffic bursts, ensuring smooth operation under load.

- Congestion Management: Use Explicit Congestion Notification (ECN) or signal-based controls to maintain network stability.

- Dynamic Connected Transport (DCT): Adopt DCT for scalable connections in distributed environments, balancing performance and resource usage

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- Cost-Benefit Consideration: While deploying separate scale-up and scale-out networks involves additional CAPEX, the return on investment is significant due to enhanced performance and reduced downtime, optimized resource utilization and operational efficiency, greater flexibility needed to adapt to evolving AI workloads and infrastructure requirements.

The synergy between Scale-up focus on intra-Pod connectivity and Scale-out emphasis on inter-Pod connectivity with scalability enables a comprehensive approach to AI data center design. This integration ensures flexibility, scalability, and performance, catering to the unique data and traffic characteristics of diverse AI/ML models. By addressing both local and distributed cluster connectivity challenges, this converged Scale-up and Scale-out architecture poised to set the foundation for the next generation of AI infrastructure.

B. Converged Scale Up & Scale Out network architecture of AI Fabric

The example of “Converged Scale-Up and Scale-Out AI/ML Infrastructure” as shown in Figure 2 leverages the complementary strengths of scale-up and scale-out under one architecture. This architecture is designed to adapt seamlessly to a wide range of application needs, from training large-scale models to handling diverse organizational workloads. Converged architecture offers a robust framework for building scalable and flexible AI systems that can evolve with future demands.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

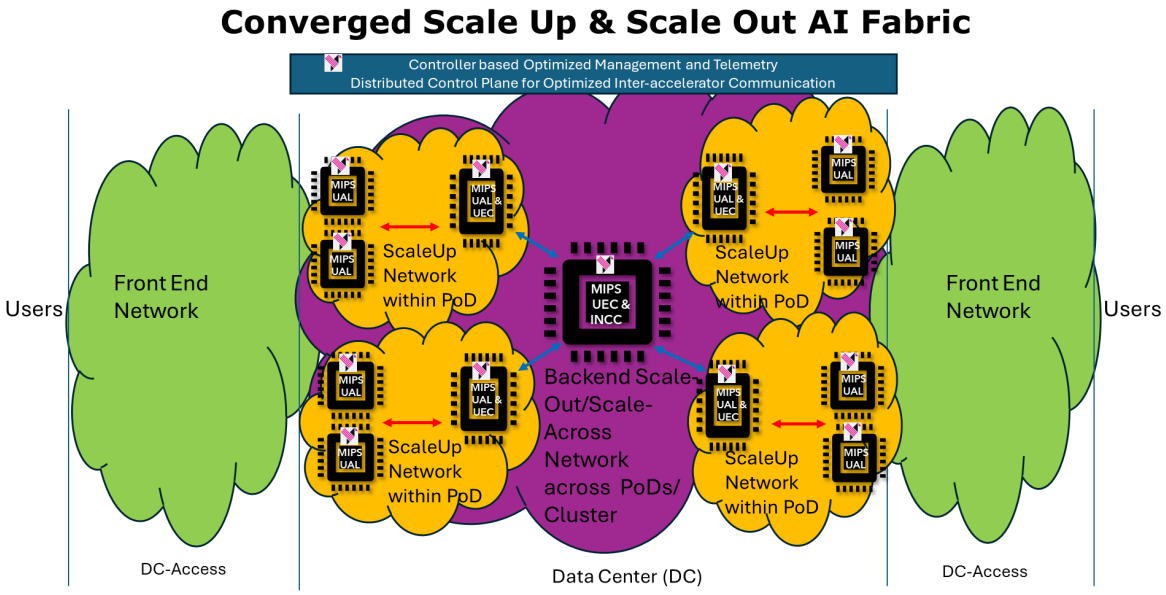


Figure 2. Converged Scale-up and Scale-out AI Fabric

Converged architecture also offers a unified, simple single pane of glass management of all different components of scale-up and scale-out back-end networks. This is very critical for easy operation, administration, and maintenance of data center AI infrastructure.

C. Advantages of Converged Architecture

The distinct but unified scale-up and scale-out network architecture in GPU and AI accelerator-based data centers is critical for optimizing AI infrastructure. This distinction enables predictable performance by addressing the differing requirements of compute-intensive AI workloads and distributed data flow, which are essential for achieving high efficiency and scalability. This architectural approach enhances security by isolating traffic types, simplifying operations through more manageable network configurations, and ensuring future scalability to accommodate the growing demands of AI workloads.

As AI workloads evolve in complexity, the flexibility to adapt the infrastructure to diverse organizational needs, AI/ML model types, and traffic patterns becomes paramount. The

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

implementation of a converged AI infrastructure — such as the Converged Scale-Up & Scale-Out AI/ML Infrastructure — offers the necessary adaptability to maintain high performance across different use cases. Since the complexity of managing separate scale-up and scale-out networks is challenging, the converged architecture brings in the needed simplification with the long-term benefits of performance, security, and scalability justifying this approach. This ultimately supports the seamless integration of diverse AI/ML applications.

Integrating the management and control planes for scale-up and scale-out networks in a converged architecture provides significant advantages in simplifying network operations and enhancing overall operational efficiency. A unified plane streamlines monitoring, configuration, and orchestration processes, reducing the operational overhead associated with managing separate networks. This integration enables consistent policy enforcement, accelerates troubleshooting, and enhances real-time adaptability to dynamic AI/ML workloads. Furthermore, a single pane of glass management and unified control plane also fosters better resource utilization, improved security, and scalability by offering a cohesive framework supporting seamless integration of diverse applications and evolving traffic patterns.

3.5 Networks, Traffic Management, and AI Fabrics

Effective networking and traffic management are vital for modern computing. While traditional techniques like load balancing, QoS, and basic congestion control are essential for general-purpose networks, standard Ethernet networks (see details in Appendix B) often struggle to meet the extreme demands of large-scale distributed AI training and inference. These unique demands necessitate specialized **AI Fabrics** – network infrastructures purpose-built for AI workloads. These fabrics integrate high-speed hardware, optimized protocols like InfiniBand® or lossless Ethernet/RoCEv2, RDMA, and advanced traffic management strategies (PFC, DCQCN, Adaptive Routing, Flowlet/Packet Balancing) to deliver the ultra-low latency, high bandwidth, and lossless performance required to efficiently train and deploy complex AI models.

Why Specialized Fabrics?

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **Massive Data Movement:** Training large models involves exchanging huge amounts of data (parameters, gradients, optimizer states) between hundreds or thousands of GPUs/accelerators.
- **Unique Traffic Patterns:** AI workloads often generate bursty, synchronized communication patterns (e.g., All-Reduce, All-to-All) that can overwhelm traditional networks (incast congestion). Not only do different AI workloads exhibit different data communication patterns, different execution phases within a single workload also show variational behaviors, such as the compute-bound prefill phase and the memory-bound decoding phase. This phased behavior often demands dynamic adaptation of AI fabrics.
- **Performance Sensitivity:** AI training performance is highly sensitive to network latency and packet loss. Even minor delays or drops can significantly slow down computation as GPUs wait for data. Shortening communication time is critical.

Key Requirements:

- **High Bandwidth:** State-of-the-art AI fabrics utilize high-speed links (e.g., 400Gbps, 800Gbps, and beyond per connection).
- **Ultra-Low Latency:** Minimizing communication delay can be paramount for efficient synchronization between distributed compute units. However, most AI workloads are feed-forward networks with little immediate feed-back, so low-latency is mostly required to avoid processors stalling and wasting power. It isn't required if the network is buffering enough data that processors don't stall.
- **Lossless Operation:** Packet loss is highly detrimental; fabrics must guarantee reliable delivery, typically using mechanisms like PFC (for Ethernet/RoCE) or built-in mechanisms (InfiniBand). Inference engines may be less sensitive to packet loss, since dropping a word from a sentence or a few pixels from a picture often doesn't change the meaning, and in training data may be cycled through repeatedly, so this can be traded off against latency. Photonic/mmWave broadcast mechanisms are likely to be lossy, but lower latency over distance.
- **Scalability:** Must efficiently scale to support clusters with thousands of interconnected accelerators. Fault tolerance becomes more essential at scale (self healing).
- **Efficient Load Balancing:** Needs to handle AI's often large, long-lived "elephant flows" effectively across available paths.

Enabling Technologies:

- **Protocols:**
  - **InfiniBand (IB):** A high-performance interconnect common in HPC and AI. Features inherent lossless operation (credit-based link-level flow control), very low latency, and support for RDMA. Often managed by a centralized Subnet Manager.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

    - **Ethernet with RoCEv2 (RDMA over Converged Ethernet):** Allows RDMA operations over standard Ethernet infrastructure. Relies heavily on a *lossless* underlying network configuration using PFC and ECN/DCQCN. Offers potential cost savings and leverages existing Ethernet expertise but requires careful network design and tuning.
    - **RDMA:** Allows network adapters to transfer data directly to/from application memory on remote hosts without involving the CPU/OS kernel, significantly reducing latency and CPU overhead. Crucial for both IB and RoCEv2 performance. SmartNICs may be considered as another type of XPU and can offer RDMA services without being specific about the mechanisms.
- **Hardware:** Switches with high port speeds, deep buffers, and hardware optimized for low latency forwarding; advanced NICs/DPUs capable of RDMA and offloading network tasks.
- **Topologies:** Network designs like **Fat-Tree (or Clos)** are common, providing high *bisection bandwidth* (aggregate bandwidth between two halves of the network) to support all-to-all communication patterns efficiently.

Note that the choice between technologies like InfiniBand and Ethernet/RoCE often involves trade-offs in cost, existing infrastructure, required expertise, and specific performance characteristics.

Traffic Management in AI Fabrics: Basic techniques like PFC and ECMP are necessary but often insufficient. Advanced methods are employed:

- **PFC/DCQCN:** As mentioned, crucial for lossless RoCEv2 fabrics.
- **Advanced Load Balancing:** To overcome standard ECMP limitations with AI's low-entropy/elephant flows:
    - **Adaptive Routing:** Switches dynamically select output paths based on real-time load or congestion information, rather than static hashing. This helps route traffic around transient hotspots. (Used in both IB and advanced Ethernet solutions like NVIDIA Spectrum-X).
    - **Flowlet Switching/Load Balancing:** Breaks large flows into smaller chunks ("flowlets") based on idle time between packets. Switches can then route different flowlets from the *same original flow* over different paths, improving path utilization compared to strict per-flow ECMP hashing.
    - **Packet Spraying (Per-Packet Load Balancing):** Distributes individual packets from a single flow across multiple available paths. This offers the most granular load balancing but can cause packet reordering, which must be handled by the end nodes (RDMA protocols can often manage this).

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Protocols like IPv6 are independent of actual hardware, and physical routing may change over time, application layer connections are through abstract mechanisms like sockets and MPI.

4. Data Movement

Data provisioning is critical to achieve the ultimate performance of AI training and inference. Efficient data provisioning has strong dependency on computing parallelism and the topology of AI fabric.

Imagine the **thousands of GPUs** are organized hierarchically:

1. **Tensor Parallelism (TP) Group:** A small group of GPUs (e.g., 4 or 8) are tightly connected, often within the same server node using high-speed interconnects like NVLink/NVSwitch. This group works together to compute a *single model layer* or operation.
2. **Pipeline Parallelism (PP) Stage:** One or more TP groups form a single stage in the pipeline. Each PP stage holds a sequential chunk of the model's layers. For example, a 100-layer model might be split into 10 stages, each holding 10 layers, with each stage running on, say, 8 GPUs (forming one TP group).
3. **Pipeline Replica:** A complete set of PP stages that holds the entire model constitutes one pipeline replica.
4. **Data Parallelism (DP) Group:** Multiple identical Pipeline Replicas work in parallel, each processing a different batch of data. The total number of GPUs is (GPUs per TP group) * (TP groups per PP stage) * (Number of PP stages) * (Number of DP replicas).

With the parallelisms above, here is how data may move during one training iteration (forward and backward pass) for a single mini-batch:

Forward Pass (Calculating Activations and Loss):

1. **Data Loading (DP Level):** The training dataset is divided into mini-batches. Each DP replica receives a *unique* mini-batch of input data (e.g., text sequences). This data is typically loaded from storage to the GPUs in the *first* PP stage of that replica. To improve efficiency, the mini-batch is often further divided into smaller *micro-batches*.
2. **Pipeline Execution (PP Level):**
    - The *first* PP stage processes the first micro-batch.
    - **Intra-Stage Communication (TP Level):** Within the first stage, the GPUs in the TP group(s) collaborate to execute their assigned layers. This

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

involves frequent, high-bandwidth communication *within* the TP group. For example, processing a large matrix multiplication might require:
    - Broadcasting or gathering parts of the input activation tensor to all GPUs in the TP group.
    - Performing partial computations locally.
    - Using collective operations (like All-Reduce or Reduce-Scatter) *within the TP group* to combine partial results and form the final output activation for that layer/operation.
  - **Inter-Stage Communication (PP Level):** Once the first stage completes processing the micro-batch, its final output activation tensor is sent point-to-point across the network (inter-node communication) to the *second* PP stage.
  - **Pipeline Fill:** As the second stage starts working on the first micro-batch, the first stage begins processing the *second* micro-batch, creating the pipeline effect. This continues across all stages.
  - **Final Stage:** The last PP stage computes the final output (logits) for the micro-batch. Often, the loss calculation (comparing logits to the target sequence) is also performed on this last stage.

Backward Pass (Calculating Gradients):

1. **Gradient Initiation (PP Level):** The backward pass starts from the calculated loss on the *last* PP stage. Gradients are computed with respect to the outputs of this stage.
2. **Pipeline Execution (PP Level - Reversed):**
  - The *last* PP stage processes the gradients for its micro-batch.
  - **Intra-Stage Communication (TP Level):** Similar to the forward pass, GPUs within the last stage's TP group(s) collaborate to compute gradients for their local portion of the weights and gradients with respect to the stage's input activations. This again involves frequent, high-bandwidth collective communication *within* the TP group (often complementary operations to the forward pass, like All-Gather or Reduce-Scatter depending on the layer type and how TP was applied).
  - **Inter-Stage Communication (PP Level):** Once the last stage completes its backward computation, the gradients corresponding to its *input activations* (which were the *output activations* of the second-to-last stage) are sent point-to-point across the network back to the *second-to-last* stage.
  - **Pipeline Flush:** This backward flow continues, stage by stage, towards the first stage. Each stage computes gradients for its weights and passes the necessary gradients backward.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Gradient Synchronization and Parameter Update (DP Level):

1. **Gradient Accumulation:** As each **DP replica** completes the backward pass for one (or more, in gradient accumulation scenarios) micro-batch, it has computed the gradients for *all* the model parameters held within that replica.
2. **Inter-Replica Communication (DP Level - All-Reduce):** Before the model parameters can be updated, the gradients computed independently by each DP replica (based on its unique data batch) must be averaged. This requires a massive All-Reduce collective communication operation that spans *all GPUs across all pipeline replicas* in the DP group. Each GPU sends the calculated gradients for its portion of the model into the All-Reduce operation, and receives back the averaged gradients for those same parameters. This is typically the most network-intensive communication step in terms of total data volume moved across the entire cluster.
3. **Parameter Update:** Finally, each GPU updates its local copy of the model parameters (or the portion it is responsible for under TP/PP) using the globally averaged gradients obtained from the All-Reduce step.

Summary of Data Movement Types:

- **TP Communication:** Extremely frequent, low latency, high bandwidth, within-node collectives (All-Reduce, All-Gather, Reduce-Scatter). Moves partial activations/gradients for layer computations.
- **PP Communication:** Less frequent than TP (once per micro-batch per stage boundary), point-to-point between nodes/stages, latency-sensitive. Moves full activation tensors (forward) and corresponding gradients (backward).
- **DP Communication:** Least frequent (once per effective batch), large-scale All-Reduce across many/all nodes, highest total volume, bandwidth-bound. Moves gradients for parameter synchronization.

5. Distributed Data Services: The Role of Storage in Large-Scale AI

In large-scale AI clusters involving thousands of GPUs, storage transcends its traditional role as a passive data repository. It functions as an active, distributed **data service** layer, deeply integrated with the compute and network fabrics. To keep pace with the immense data appetite and parallel processing capabilities of modern AI workloads (like training LLMs or DLRMs), these storage services must deliver extreme performance, massive scalability, high parallelism, and a rich set of features beyond simple reads and writes. This often involves moving away from traditional Network Attached Storage (NAS) and Storage Area Network (SAN) towards high-performance

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

distributed file systems (e.g., Lustre, IBM Spectrum Scale/GPFS) or scale-out object storage platforms (e.g., Ceph, or S3-compatible systems optimized for performance) accessible concurrently across the entire compute cluster.

In the context of large-scale AI, storage must be viewed as a foundational, high-performance distributed data service layer. Its ability to provide fast, parallel data access, manage massive embedding tables, rapidly save and load multi-terabyte checkpoints, and offer integrated services like transformation, encryption, and compression is paramount. The performance, scalability, and RAS characteristics of these storage services are inextricably linked to the overall efficiency, robustness, and success of demanding AI training and inference workloads, functioning as a critical element alongside the compute and network infrastructure

Distributed storage services can address the extreme compute needs above by attacking the following technology areas [2,3]:

A. **High-Performance Data Movement & Access:**
    a. **Parallel Access:** Distributed storage systems provide parallel data paths, allowing thousands of GPU nodes to simultaneously access different parts of massive training datasets (often petabytes in size) without creating I/O bottlenecks.
    b. **Proximity and Caching:** Data can be staged or cached closer to the compute nodes (e.g., on node-local NVMe drives managed as part of the distributed system or via client-side caching) to reduce read latency for frequently accessed data blocks.
    c. **Direct Access Technologies:** Technologies like NVIDIA's GPUDirect Storage (GDS) enable GPUs to directly read and write data from compatible distributed storage systems over the network fabric (e.g., using RDMA), bypassing the CPU and system memory. This significantly reduces latency and CPU overhead for data loading and checkpointing, keeping the GPUs fed more efficiently.
    d. **AI-native Inference Context Memory Storage:** This storage extends GPU memory with a cluster-level key-value (KV) cache, enabling high-speed sharing of context across nodes for multi-turn, long-context agentic AI. It leverages programmable data processors to offload infrastructure services (including networking, storage, and security) from the host CPU, reducing overhead and enabling secure, high-performance operation within orchestrated control frameworks. Hardware acceleration for KV cache placement and a high-performance RDMA network fabric provides low-latency access, improving responsiveness, throughput, and

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

power efficiency for scalable inference workloads while simplifying lifecycle management of the offload infrastructure.

B. **Data Transformation Services & Offloading:**
While GPUs handle the core model computations, some data preprocessing (like image decoding, data augmentation, text tokenization, or feature extraction) can be also computationally intensive. Thus, distributed data service platforms can integrate with compute resources (potentially dedicated nodes near storage, or even computational storage devices themselves) to perform these transformations *before* the data hits the primary training GPUs. This offloads work from the expensive GPU resources, reduces data movement (if transformations expand data size), and avoids redundant preprocessing across different jobs or epochs.

C. **Encryption and Security:**
AI datasets can be highly sensitive (e.g., medical images, financial data, personal user data). Distributed storage services provide essential security features:
   a. **Encryption at Rest:** Automatically encrypting data stored on the physical media (SSDs, HDDs) protects against physical theft or unauthorized access to the drives. Transparent Data Encryption (TDE) handled by the storage layer simplifies deployment.
   b. **Secure Access:** Robust authentication and authorization mechanisms ensure that only permitted users and applications can access specific datasets or parts of the storage namespace.

D. **Compression:**
Datasets and model checkpoints can consume vast amounts of storage capacity. Distributed storage systems often offer transparent compression, automatically compressing data when written and decompressing that data when read. This saves valuable storage space. Hardware-accelerated compression on storage nodes or DPUs/SmartNICs can minimize performance impact. Compression also reduces the amount of data that needs to be transferred over the network, particularly beneficial when saving/loading large multi-terabyte checkpoints.

E. **Handling Massive Embedding Tables:**
Recommendation models (DLRM) and some generative models utilize huge embedding tables (potentially terabytes) that far exceed the memory capacity (HBM) of individual GPUs or even entire nodes. The distributed storage system serves as a high-capacity, low-latency backing store for these tables. Specialized AI frameworks often manage data movement, loading only the necessary



This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

embedding vectors into GPU HBM or system DRAM just-in-time for computation, relying on the storage service for fast, random access to the rest of the table.

F. **Checkpointing and Resilience:**
Training large AI models can take weeks or months, making it essential to save progress regularly via **checkpointing**. Distributed data services provide a high-bandwidth, parallel destination allowing the state of the model (weights, optimizer states, etc.) from thousands of GPUs to be written concurrently and quickly, minimizing the pause in training (e.g., saving multi-terabyte checkpoints in minutes). Conversely, fast parallel reads from the distributed storage are crucial for rapidly restoring state and resuming training after planned restarts or unexpected hardware failures, maximizing cluster utilization.

G. **Distributed State Management:**
Beyond model checkpoints, complex distributed AI workflows might require managing other forms of shared state (e.g., distributed data structures, coordination flags, job progress indicators). While often handled by specialized coordination services (like etcd) or built into the compute framework (like Ray), the underlying persistent distributed storage system can be leveraged for reliably snapshotting or persisting certain types of state needed for broader application-level fault tolerance or coordination.

H. **Hardware Considerations:**
Effective distributed data services typically employ a tiered hardware strategy:
   a. **Flash Tier (NVMe SSDs):** For metadata operations, storing small files, caching hot data, and providing low-latency access for embedding tables or small reads/writes.
   b. **Capacity Tier (HDDs/QLC SSDs):** For storing large datasets, checkpoints, or less frequently accessed data where sequential bandwidth is important.
   c. **Networking:** High-speed network interfaces (e.g., 200/400 Gbps Ethernet or Infiniband) are essential on storage nodes to match the aggregate bandwidth of the compute fabric.
   d. **DPUs/SmartNICs:** Increasingly used on storage nodes to offload storage protocols (NVMe-oF), networking tasks, and potentially data services like compression, encryption, or erasure coding calculations, freeing up CPU resources.

I. **Reliability, Availability, and Serviceability (RAS):**
Distributed storage architectures inherently enhance RAS:

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

a. **Reliability/Availability:** Techniques like data replication or erasure coding ensure data durability and availability even if individual disks or entire storage nodes fail. This prevents data loss and allows training jobs to continue or restart quickly.
b. **Serviceability:** The modular design allows for non-disruptive maintenance, capacity scaling by adding more nodes, and rolling hardware/software upgrades, minimizing interruptions to critical AI workloads. Integrated monitoring and fault detection simplify operations.

In the context of large-scale AI, storage must be viewed as a foundational, high-performance distributed data service layer. Its ability to provide fast, parallel data access, manage massive embedding tables, rapidly save and load multi-terabyte checkpoints, and offer integrated services like transformation, encryption, and compression is paramount. The performance, scalability, and RAS characteristics of these storage services are inextricably linked to the overall efficiency, robustness, and success of demanding AI training and inference workloads, functioning as a critical element alongside the compute and network infrastructure.

5.1 Integrating Distributed Data Services with AI Fabrics

Making the distributed data service layer as an integral part of AI Fabrics can **leverage and influence the compute-communication paradigm** inherent in large-scale AI training and inference. The following strategies are the major design considerations of high-performance AI Fabrics.

1. **Storage I/O as Managed Communication:** In large-scale ecosystems, accessing storage (reading datasets/embeddings, writing checkpoints) should be viewed as another critical **communication flow**, alongside the GPU-to-GPU communication (gradients, activations) dictated by parallelism strategies (TP, PP, DP). The AI Fabric must be designed to efficiently handle *both* compute-to-compute and compute-to-storage communication patterns.

2. **Exploiting AI Fabric Capabilities for Data Movement:**
    - **High Throughput & Low Latency:** Distributed storage systems connect directly to the same AI Fabric (e.g., InfiniBand, lossless RoCE Ethernet) used by the GPUs. This allows I/O operations to benefit from the fabric's raw speed, minimizing the time GPUs are stalled waiting for data or saving state.
    - **RDMA Integration (e.g., GPUDirect Storage):** Technologies like GDS allow GPUs to communicate *directly* with storage over the AI Fabric using RDMA, bypassing the CPU and kernel network stack. This drastically

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

reduces the latency and overhead of data movement, making storage access behave more like a direct memory operation within the compute-communication flow rather than a slow, external I/O call.

3. **Optimizing the Compute-Communication Balance During Training:**
    - **Reducing I/O Wait Time:** By accelerating data loading (parallel reads) and checkpointing (parallel writes) over the AI Fabric, the distributed data service minimizes the communication component associated with I/O wait times. This allows the compute-communication balance to be dominated by the inherent needs of the algorithm and parallelism strategy (e.g., the ratio of computation time within a layer vs. the time spent on gradient All-Reduce or activation passing). Fast storage access prevents I/O from becoming the primary bottleneck that masks potential compute or inter-GPU communication inefficiencies.
    - **Enabling Larger Models/Embeddings:** Low-latency access over the fabric to embedding tables or model parameters stored externally allows models to exceed GPU memory limits without meaningfully reducing performance. The communication cost of accessing these off-chip parameters via the fabric becomes a manageable part of the overall compute-communication cycle.

4. **Optimizing Compute-Communication During Inference:**
    - **Fast Model Loading:** Inference servers can rapidly load large, multi-terabyte models from the distributed storage service over the AI Fabric, minimizing downtime or latency for model updates and scaling.
    - **Real-time Access:** Applications requiring real-time inference against large datasets or knowledge bases stored externally rely on the low-latency communication provided by the fabric between the inference compute nodes and the data services.

5. **Synergy: Compute Offload and Fabric Efficiency:** By performing data transformations, compression/decompression, or encryption within the distributed data service layer (potentially using dedicated compute or DPUs near storage), the system reduces the volume or complexity of data that needs to traverse the AI Fabric to the primary compute GPUs. This optimizes fabric bandwidth usage and offloads tasks from the expensive GPU resources, further refining the compute-communication balance. Compute offloading can be further exploited by leveraging built-in compute capability within AI Fabrics itself (i.e., in-network computing) to accelerate AI specific operations, such as collectives or model protections.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

6. **Co-Design Requirement:** The effectiveness of this integration hinges on **co-design**. The bandwidth and latency characteristics of the AI Fabric must be provisioned to handle the *aggregate* load of both inter-GPU communication (TP, PP, DP patterns) *and* the peak I/O demands from the distributed data services (concurrent reads for data loading, concurrent writes for checkpointing). The storage system's internal architecture and network interfaces must be capable of saturating its link to the fabric during these peak demands. The parallelism strategy itself might even be influenced by the achievable latency to externalized parameters or embeddings.

7. **Fault Tolerance:** Compute nodes or accelerator cards within a node can fail frequently, especially with tens of thousands of cards during large-scale training. To quickly recover from failures, it requires AI Fabrics to have the ability to detect / isolate the failed nodes/devices, and readjust the AI Fabric to accommodate reduced resources with new (or even irregular) topology. Then, the previous checkpoint will be reloaded and training is restarted with new optimized compute-communication patterns.

Distributed data services in large-scale AI ecosystems are not separate from the compute-communication paradigm – they are integral to it. These services leverage the high-performance AI Fabric to minimize I/O bottlenecks, effectively transforming storage access into another managed, high-speed communication flow. This deep integration ensures that data availability and movement do not throttle the expensive compute resources, allowing the system's efficiency to be governed primarily by the interplay between parallel computation and the inherent communication requirements (gradients, activations) of the AI training or inference algorithms themselves.

5.2 Synchronous & Latency-Sensitive Traffic

Data movement in and out of distributed data services has the same impact on the training / inference performance as **the compute communication paradigm** described in the previous sections. Thus, data servers need to take various parallelisms (DP, TP, PP, etc.) into design considerations of high-performance AI Fabrics. For example, critical embedding table lookups during training often leverage RDMA/GPUDirect Storage to achieve low latency access from compute nodes to the memory/storage subsystem over the AI Fabric.

Note that recent advances in tensor-parallel communication emphasize **algorithmic latency reduction** rather than faster interconnects alone. New approaches restructure collectives using direct data access and fused all-reduce kernels, reducing synchronization and launch overhead on the critical path. These are complemented by runtime and library improvements — such as direct NIC access, alternative

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

high-performance collective libraries, and techniques that partially relax synchronization in tensor-parallel activation exchange without accuracy loss — fundamentally reshaping latency behavior.

5.3 Asynchronous & Less Latency-Sensitive Traffic

Asynchronous traffic often happens in the background and tolerates more variability in completion time without immediately halting the primary computation. Aggregate bandwidth (i.e., throughput) is often more critical than per-operation latency.

For example, during training and batch inference, data loading and prefetching read training batches or batch inference data from the distributed storage system into GPU/system memory. Data loading is typically done asynchronously. Data loaders prefetch upcoming batches while the GPUs are busy processing the current one. The key is ensuring the average data loading *throughput* matches the GPU consumption rate to avoid starvation, rather than minimizing the latency of every single read. Similarly, checkpointing (training) saves the model's state (weights, optimizer parameters) to distributed storage periodically. It is a background task initiated periodically and doesn't need microsecond-level latency. High *bandwidth* to storage is the primary requirement to complete the potentially multi-terabyte write quickly. Additionally, logging, monitoring, telemetry also fall into the asynchronous I/O operation category as they often use separate management networks or lower priority queues on the AI Fabric. The data is typically not on the critical path for computation or request processing. It can tolerate reasonable delays.

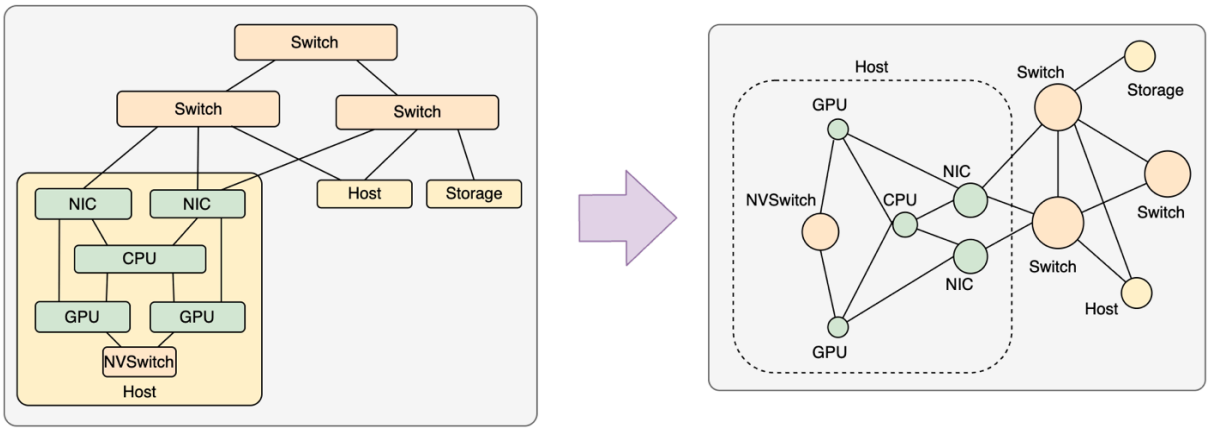
Implications for AI Fabrics: AI Fabrics must concurrently provide ultra-low, predictable latency for synchronous, sensitive flows (TP, PP, real-time inference) and massive aggregate bandwidth for asynchronous, high-volume flows (DP All-Reduce, checkpointing, data loading). Thus, sophisticated AI Fabric management mechanisms (PFC, congestion control, QoS prioritization), as listed in Section 3.5, are needed to manage these diverse flows. Advanced Load Balancing techniques beyond simple hashing (adaptive routing, flowlet switching) also help distribute traffic evenly to avoid hotspots that could increase latency for critical flows.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

6. AI Infrastructure Modeling: A Standardized Schema with InfraGraph

As AI tasks and their infrastructure become more sophisticated, it's increasingly important to accurately model the compute, network, and storage systems. Creating advanced AI platforms—such as extensive GPU arrays or custom-built accelerator units—requires engineers to balance computing strength with network bandwidth and layout. Optimizing how GPUs, ASICs, and fast interconnects work together is challenging, especially for those lacking deep expertise in networking. One major hurdle is the lack of a standard, vendor-independent method for describing these complex "**systems of systems**," which slows down collaborative design and complicates the identification of performance issues or architectural improvements.

InfraGraph bridges this divide by offering a model-based, vendor-neutral interface that represents entire infrastructures as graphs consisting of interconnected parts. As shown in Figure 3, in InfraGraph, system architects can define servers, GPUs, network cards, switches, and other components as vertices or nodes, while depicting links between them as standardized edges. This graph structure aligns with the way data actually moves: vertices stand for computing devices, and edges mark high-speed connections. By using a unified graph language, InfraGraph promotes better collaboration between hardware and networking teams, supporting more accurate modeling and clearer communication at the system level.



Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Figure 3. An example of InfraGraph

6.1 Why InfraGraph Matters for AI Infrastructure Modeling

InfraGraph provides a unified, extensible, and machine-interpretable way to describe modern AI infrastructure—spanning compute, storage, networking, and system-level structure. Its value comes from enabling researchers, architects, and operators to explore, compare, and manage complex infrastructure designs with precision and consistency.

A. Unified and Abstracted System Representation

InfraGraph offers a single, standardized schema for modeling diverse hardware components—CPUs, GPUs, accelerators—and the networks that bind them. This provides a single source of truth for system design. Topologies of any kind (scale-up, scale-out, or scale-across) can be expressed uniformly, allowing systems to be analyzed consistently across simulation, emulation, and real deployments. The abstraction layer also enables organizations to share structural models without revealing sensitive implementation details, promoting transparency and collaborative benchmarking.

B. Rich Extensibility Through Annotations

A key strength of InfraGraph is its annotation mechanism, which allows unlimited custom attributes to be attached to nodes and edges—for example, bandwidth, latency, power usage, reliability metrics, or vendor-specific characteristics. Multiple analytical views (performance, cost, resiliency, energy) can be built on top of the same base description, eliminating the need to maintain multiple variants of the model.

C. Topology Exploration, Boundaries, and What-If Analysis

InfraGraph supports detailed exploration of fabric and system designs, including explicit modeling of boundaries such as memory-semantic vs. network-semantic domains. These boundaries and their properties allow engineers to reason about communication patterns, scaling limits, and placement constraints. InfraGraph can act as the common, structured input to external tools where users can run **what-if studies** to compare

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

performance, resiliency, and cost across candidate architectures and even attach workload descriptions for AI/HPC use cases.

D. Reusability and Composability

The modular nature of InfraGraph makes it easy to define reusable components—standard servers, accelerator pods, network switches, or entire topologies like Clos or Dragonfly. Vendors can contribute component definitions independently, and these modules can be composed to form larger systems. This approach hides internal complexity while enabling repeatable, faster experimentation.

E. Integration and Tooling Ecosystem

InfraGraph's machine-parseable schema acts as a common language for management, orchestration, and automation frameworks. Annotations can bridge vendor-specific tools, reducing manual interventions. Being graph-based by design, InfraGraph integrates naturally with visualization, analysis, and graph-processing tools, enabling intuitive inspection of relationships, dependencies, and bottlenecks.

6.2 Introduction to the InfraGraph Schema

InfraGraph uses basic graph principles—vertices, edges, links, and properties—to illustrate systems and their supporting network infrastructure. The definitions below establish the essential concepts that underpin the InfraGraph schema.

- **Vertices** or **Nodes** represent components within a device or individual device instances.
- **Devices** may themselves be modeled as nested subgraphs that capture their internal components and interconnections.
- **Links** describe the logical definition of a connection between components or devices, while **edges** represent the actual one-to-one physical or direct connections.
- **Edges** consist of two endpoints, each corresponding to a specific component of a device instance, along with its index.
- **Paths** are sequences of device-edges and infrastructure-edges that connect a defined source to a destination.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

6.3 Steps to Model Systems and Infrastructures

InfraGraph provides a systematic way to construct reusable device models and assemble them into full infrastructure descriptions. As illustrated in Figure 4, the modeling workflow typically involves the following steps:

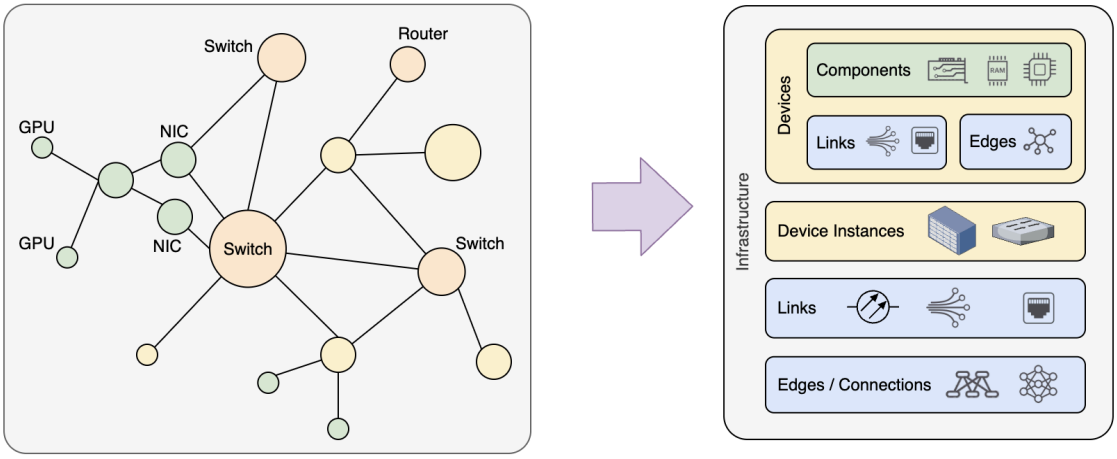


Figure 4. Illustration of InfraGraph modeling workflow

1) Create the Device Inventory

Define reusable device models using InfraGraph. A device can represent any simple or complex system—such as a server, switch, storage node, or accelerator chassis.

a) **Define the Device Structure:** Complex devices consist of multiple internal components. For example, an HPC server may include GPUs, CPUs, memory modules, and NICs. Each of these elements is modeled as a component within the device.
b) **Define Link Types Within the Device:** Links describe the kinds of connections that can exist between components, such as PCIe, NVLink, or other internal interconnects. Each link type may carry properties like bandwidth, latency, or error rate.
c) **Define Edges Within the Device:** Edges represent concrete instances of links, connecting two specific component instances inside the device. For example, an edge may connect a GPU instance to a NIC port.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

2) **Instantiate Devices in the Infrastructure**

After defining device models, one or more instances of each device type can be instantiated and placed within the overall infrastructure graph.

3) **Define Infrastructure-Level Link Types**

Similar to intra-device links, infrastructure-level link types—such as Ethernet or InfiniBand—must be defined. These specify how device instances can be connected across the system.

4) **Create Infrastructure Edges**

Finally, connect device instances using edges based on the predefined infrastructure link types. Each edge maps two endpoints, each corresponding to a specific component of a device instance.

6.4 Framework and Tooling

The InfraGraph ecosystem includes APIs, schema models, supporting libraries, and example blueprints that help users define, validate, and work with infrastructure graphs effectively.

A. API and Schema Models

InfraGraph schemas are defined using OpenapiArt**,** which generates declarative, intent-based models and APIs. This framework produces fluent Python and Go SDKs for both server and client implementations over REST or protobuf.

Core data models—such as components, devices, links, and edges—are expressed as YAML-based schema definitions. These definitions act as the foundation for object creation and also provide built-in validation to ensure correctness across all parts of the schema.

B. Programmatic Creation Using the Fluent SDK

InfraGraph favors programmatic generation over manual JSON or YAML system descriptions. With the fluent SDK, users define devices, components, and link types as

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Python or Go objects to build infrastructure models. These objects can be converted to YAML for easier storage, sharing, or automation.

C. Integration with Graph-Based Tools

Being fundamentally graph-driven, InfraGraph integrates naturally with existing graph libraries and tooling. Internally, it uses **NetworkX,** enabling seamless use of graph algorithms, visualization tools, and query frameworks for analysis and experimentation.

D. InfraGraph Service

The InfraGraph Service provides an interface for defining, validating, and querying infrastructure graphs. It enables tasks such as validating a device model, retrieving connectivity information based on specific criteria, or computing shortest paths between components or devices.

E. Example Blueprints

The repository includes example blueprints of well-known systems and topologies, offering quick references and reusable patterns for building new models or experimenting with different fabric designs.

F. References

- Documentation: https://infragraph.dev/
- Source Code: https://github.com/Keysight/infragraph
- Schema Specification: https://infragraph.dev/openapi.html

6.5 Operationalizing the Model: The Digital Twin

While a standardized schema like InfraGraph is essential for design-time planning, its value extends significantly into the operational phase. By integrating the InfraGraph model directly into the cluster orchestrator, we propose to create an **Operational Digital Twin**, a live, software-defined mirror of the physical fabric state.

Real Time Simulation and "What-If" Analysis In a traditional network, the orchestrator reacts to telemetry. In an AI fabric powered by a Digital Twin, the orchestrator *predicts*

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

the future state. Before committing to a complex operation, such as scheduling a massive training job, re-routing a flow around a failure, or changing the topology of an optical circuit switch, the system first runs a millisecond-scale simulation on the model.

This "Online Simulation" capability allows operators to:

- **Validate Configurations:** Ensure that a proposed topology change will yield the expected bandwidth improvement without violating SLAs or causing congestion.
- **Optimize Job Placement:** Run rapid Monte Carlo simulations to determine the mathematically optimal placement of a workload across fragmented GPU resources.
- **Predictive Maintenance:** Simulate the impact of taking a specific spine switch offline for maintenance, automatically identifying the optimal alternative paths before a single packet is dropped.

By transforming the infrastructure model from a static documentation artifact into a dynamic operational tool, we close the loop between physical reality and logical intent.

7. AI Workload Categorization & Mapping

7.1 The Co-Design Challenges

Previous sections explored innovations in AI fabric infrastructure, from multi-dimensional topologies and hierarchical network architectures to memory disaggregation and advanced traffic management strategies. These networking advances provide the foundation for high-performance distributed training and inferencing. However, even optimal network fabrics cannot achieve their potential without intelligent workload-to-hardware mapping.

As AI systems scale to thousands of heterogeneous devices and incorporate complex multi-dimensional fabrics, the mapping problem becomes increasingly critical. The

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

question is no longer just "what network topology should we build?" but rather "given our workload's communication patterns and our fabric's capabilities, how do we optimally place operations across devices?" This represents a fundamental shift toward true compute-communication co-design, where network topology, workload characteristics, and device placement must be jointly optimized.

7.2 Impact of Workload Diversity on Network Utilization

Different AI workloads exhibit fundamentally different communication patterns, each with distinct implications for network fabric utilization. As shown in Section 2.3, Expert Parallelism is a good example which demonstrates dynamic, input-dependent communication patterns when running large Mixture-of-Experts models.

Poor mapping has significant consequences. Placing tensor-parallel operations across high-latency fabric dimensions leaves fast interconnects underutilized. Misaligning data-parallel groups with network topology creates congestion hotspots. With memory disaggregation, the placement problem extends beyond compute devices to include decisions about which data resides in local HBM versus remote memory pools, directly impacting fabric load.

Parallelization strategies must align with fabric hierarchy. Tensor parallelism should exploit scale-up networks with high bandwidth and low latency, while data parallelism can effectively utilize scale-out networks. Hybrid parallelism strategies that ignore these fabric characteristics will suffer from poor network utilization regardless of the underlying hardware quality.

7.3 Essential Components for Workload Mapping

Successful workload mapping requires three core components:

1. Hardware Representation

A structured description of the target infrastructure must capture topology structure, bandwidth hierarchy across dimensions, memory organization (local vs. remote), and connectivity constraints. The representation should be rich enough to inform mapping decisions. For instance, distinguishing between scale-up links (such as NVLink at 900

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

GB/s) and scale-out links (such as Ethernet at 400 Gbps) is essential for proper placement of communication-intensive operations.

The InfraGraph schema (Section 6) provides such a graph-based infrastructure description where devices and components are nodes, connections are edges, and annotations capture performance characteristics. This machine-readable representation enables algorithmic reasoning about placement constraints and communication costs.

2. Workload Characterization

The workload must be represented to expose compute operations, memory accesses, and communication collectives along with their dependencies and execution order. This characterization should reveal the communication pattern: which operations participate in which collectives, the data volumes involved, and the sensitivity to latency versus bandwidth.

Execution traces provide a standardized approach. For example, **Chakra** [5] execution traces represent workloads as directed acyclic graphs where nodes encode operations (compute with FLOPs and tensor sizes, memory loads/stores, or collective communications with type and size) and edges encode dependencies. This representation is framework-agnostic and can be generated from PyTorch, TensorFlow, or other training frameworks.

3. Mapping Decision Engine

Given hardware and workload representations, a decision engine must determine which operations execute on which devices. The typical objective is to minimize end-to-end training time by balancing compute utilization, network load across fabric dimensions, and memory capacity constraints (both local HBM and remote pools).

The decision engine must respect hard constraints: topology limitations, collective algorithm requirements (topology-aware hierarchical collectives depend on device arrangement), and co-location rules (certain operations may need to remain together to avoid communication overhead or maintain correctness).

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

7.4 Mapping Methodologies

Multiple methodologies exist for translating hardware and workload representations into mapping decisions:

Simulation-Based Search

A simulator evaluates candidate mappings by modeling execution on the target infrastructure. For instance, ASTRA-sim can consume InfraGraph hardware descriptions and Chakra execution traces to estimate end-to-end training time for a given mapping. Search algorithms (evolutionary algorithms, simulated annealing, or reinforcement learning) then explore the space of possible placements, using simulation results as the objective function.

This approach requires no training data and can directly optimize for the specific hardware-workload combination of interest. The downside is computational cost: each candidate mapping requires a full simulation. However, since mapping is typically performed once (or infrequently) for a given model-hardware pair, this overhead is often acceptable.

Learning-Based Prediction

Learning-based approaches train models to predict good placements directly from workload structure. Graph neural networks are particularly well-suited: they encode the computation graph structure (operations as nodes, dependencies as edges) along with node features (operation type, tensor sizes, compute requirements) and learn to predict optimal device assignments.

These models require substantial training data, which can come from recorded executions on real hardware or from large-scale simulation campaigns. Once trained, inference is fast (milliseconds to make placement decisions for new workloads), enabling real-time or interactive use cases. The challenge is generalization: models must work across workload variations and potentially different hardware configurations than seen during training.

Hybrid Strategies

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Practical systems often combine approaches. A common pattern is coarse-to-fine mapping: first, use domain knowledge or heuristics to assign workload stages to network dimensions (for example, place tensor parallelism on scale-up fabric, data parallelism on scale-out), then apply simulation-based search or learning to optimize placement within each dimension. This hierarchical decomposition reduces the search space while respecting the fabric hierarchy.

Another hybrid approach uses heuristics for initialization followed by refinement via search or learning. For instance, a simple rule might place communication-heavy operations close together in the topology, then refine this initial mapping using more expensive optimization. Dynamic remapping based on runtime telemetry (detecting congestion or stragglers) can further adapt to changing conditions.

7.5 Open Directions

Workload mapping for AI fabrics remains an active research area with several open challenges:

Cross-layer co-optimization extends beyond fixed workload mapping to jointly optimize model architecture, parallelization strategy, and hardware placement. Emerging auto-tuning frameworks attempt to navigate this expanded design space, but the combinatorial complexity is significant.

Multi-tenancy introduces new constraints: multiple workloads may share fabric resources, requiring mapping decisions that avoid interference while maintaining fairness. Dynamic workload arrival and departure further complicate the problem.

Scalability of mapping algorithms becomes a concern at extreme scale. With 10,000+ devices, even evaluating a single candidate mapping can be expensive. Hierarchical decomposition and incremental search techniques are necessary but not yet fully mature.

Finally, **integration with production systems** requires low-overhead profiling to gather the data needed for mapping decisions, mechanisms for safe exploration of new mappings without disrupting training, and automated workflows that make co-design accessible to

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

practitioners rather than requiring expert knowledge of both AI algorithms and networking.

As AI models and infrastructure continue to evolve rapidly, the co-design of workloads and fabrics through intelligent mapping will be essential to realizing the full potential of next-generation training platforms.

8. Conclusion

This paper presents a comprehensive framework for AI datacenter infrastructure, addressing the compute-communication paradigm needed for modern AI model training and deployment. The framework aims to establish vendor-neutral terminology for AI fabrics, spanning chip to system to datacenter scale.

Effective AI systems require holistic co-design of parallelization strategies, network fabrics, and storage services to balance compute and communication demands. The framework categorizes AI workloads by parallelization strategies including Data Parallelism (gradient synchronization via All-Reduce), Model/Pipeline Parallelism (splitting models across devices), Tensor Parallelism (intra-layer operations), and Expert Parallelism (dynamic routing in Mixture-of-Experts models) such that distinct network demands on the AI Fabric can be created for each strategy. The framework also provides a standardized, graph-based schema, called InfraGraph, to model AI Fabric and enable topology exploration. InfraGraph's machine-readable representation enables algorithmic reasoning about placement constraints and communication costs according to AI application's characterization, achieving the optimal workload-to-hardware mapping.

In addition, this paper advocates for converged AI fabric architecture with four network segments: Scale-In (die-to-die modularity via UCIe®), Scale-Up (memory semantics, <1µs latency, 100s Tbps bandwidth within pods), Scale-Out (RDMA semantics, <10µs latency, 400-800 Gbps across pods), and Scale-Across (inter-datacenter communication). Key technologies include UCIe®, UALink, ESUN, and Ultra Ethernet, with advanced traffic management using PFC, ECN, and adaptive routing for lossless operation. Distributed data services can be integrated directly with AI fabrics via RDMA for high-performance parallel access, checkpointing, and embedding table management.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Ultimately, the convergence of standardized protocols, Silicon Photonics, and the InfraGraph modeling schema paves the way for the 'Predictive AI Fabric.' By transitioning from static documentation to operational digital twins, organizations can move from reactive troubleshooting to proactive optimization, building infrastructure that not only meets today's demands but adapts autonomously to the innovations of tomorrow.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

9. Reference

1. Context parallelism, https://mlsys.org/media/mlsys-2025/Slides/3255.pdf
2. https://nvidianews.nvidia.com/news/nvidia-bluefield-4-powers-new-class-of-ai-native-storage-infrastructure-for-the-next-frontier-of-ai
3. https://ubuntu.com/blog/canonical-kubernetes-meets-nvidia-doca-platform-framework-dpf-building-the-future-of-dpu-driven-infrastructure
4. https://engineering.fb.com/2025/10/17/ai-research/scaling-llm-inference-innovations-tensor-parallelism-context-parallelism-expert-parallelism/
5. MLCommons, Chakra, https://mlcommons.org/working-groups/research/chakra/

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Appendix A

Glossary

- **Asynchronous**: Refers to operations or communication where the initiating process does not need to wait for the operation to complete before continuing with other tasks. In AI fabrics, this often applies to background data transfers like dataset prefetching, checkpoint writing, or sending telemetry data, which can tolerate higher or more variable latency and are often optimized for throughput.
- **Synchronous**: Refers to operations or communication where the initiating process must wait for the operation or response to complete before proceeding. This implies a tight dependency and sensitivity to latency. In AI fabrics, examples include Tensor Parallelism communications within a layer, Pipeline Parallelism transfers between stages, and the request-response path for real-time inference, all requiring low, predictable latency.
- **Scale-up**: Increasing the performance and capacity of a single compute node by adding more resources, such as more powerful CPUs, additional GPUs/accelerators connected via high-speed internal interconnects (like NVLink), or more memory. Within AI fabrics, scale-up often refers to maximizing the capability within a node before distributing work across multiple nodes (scaling out).
- **Scale-out**: Increasing the performance and capacity of a system by adding more independent nodes (e.g., servers with GPUs) and distributing the workload across them. This is the primary method for building large AI training clusters with thousands of accelerators. Scale-out architectures rely heavily on the performance (bandwidth, latency, scalability) of the interconnecting network or AI Fabric.
- **Coherency**: Generally, maintaining consistency of data shared across multiple locations (like caches or memories). In large-scale AI, ensuring data coherency, particularly for model parameters across different GPUs/nodes, is crucial. While hardware cache coherency is typically limited to within a node (e.g., via NVLink), consistency across nodes in distributed training is usually achieved through explicit synchronous communication steps, such as the All-Reduce collective used in Data Parallelism to average gradients before parameter updates.
- **Data Services**: Viewing the storage infrastructure not just as passive repositories but as an integrated, distributed system providing essential services for large-scale AI workloads. These services, accessed via the AI fabric, include high-performance parallel data access (reads/writes), data transformation (preprocessing), security (encryption, access control), efficiency features (compression), specialized functions (embedding table hosting), and critical

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

operational support (checkpointing, state management), all requiring high reliability and availability (RAS).

- **Collective (Communication**): A communication pattern involving a coordinated data exchange among a group of processes or devices working in parallel. Collectives are fundamental building blocks for distributed AI training algorithms, particularly for synchronization in Tensor Parallelism (intra-layer updates) and Data Parallelism (gradient aggregation). Common examples include All-Reduce, Broadcast, Reduce, All-Gather, and Reduce-Scatter, each imposing specific communication patterns and demands (latency, bandwidth) on the AI Fabric.

- **Cluster (AI Cluster):** A collection of interconnected computing nodes (servers typically containing multiple GPUs or other accelerators), high-performance storage systems (distributed data services), and a specialized network (the AI Fabric), all working together cohesively to execute large-scale AI workloads like distributed training or high-throughput inference.
- **Pod (AI Infrastructure Pod):** In the context of large-scale AI infrastructure (distinct from Kubernetes pods), a Pod typically refers to a standardized, modular unit or building block of the cluster. It often comprises a defined set of racks containing compute nodes, network switches (e.g., leaf switches), and associated power/cooling, interconnected with specific internal bandwidth characteristics. Multiple pods are interconnected via the core/spine layer of the AI Fabric to form a larger cluster. Examples include Google's TPU Pods or similar conceptual units in large GPU cluster designs.
- **Domain**:
    - **Failure Domain:** A group of infrastructure components (e.g., servers in a rack, racks connected to a single leaf switch, components powered by one PDU) that share a common point of failure. Designing AI clusters with awareness of failure domains (e.g., distributing stages of a pipeline or data replicas across different domains) enhances overall system resilience.
    - **Communication Domain / Locality Domain (e.g., NVLink Domain):** A set of processors (typically GPUs) tightly interconnected with very high-speed, low-latency links, often within a single physical node (e.g., GPUs connected via NVLink/NVSwitch). Communication within such a domain is significantly faster than communication across the general network fabric, making it ideal for latency-sensitive operations like Tensor Parallelism.
- **AI Fabric**: The underlying high-performance network infrastructure (switches, NICs/DPUs, optical transceivers, cables, protocols) specifically designed and configured to handle the demanding communication patterns of large-scale distributed AI workloads, characterized by requirements for high bandwidth, ultra-low latency, lossless data transport, and scalability.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **RDMA (Remote Direct Memory Access):** A technology that allows a network interface card (NIC) in one computer to directly access the main memory (RAM or GPU memory) of another computer across the network fabric without involving the operating system or CPU of either computer. Essential for low-latency, high-throughput communication in AI fabrics, used by protocols like InfiniBand and RoCEv2.
- **RoCEv2 (RDMA over Converged Ethernet v2):** A network protocol standard that enables RDMA communication directly over Ethernet networks (specifically using UDP/IP for Layer 3 routability). Requires a carefully configured, lossless Ethernet fabric (using PFC, ECN) to operate efficiently and reliably.
- **InfiniBand (IB):** A high-performance computing network interconnect standard offering high bandwidth and very low latency. It natively supports RDMA and typically ensures lossless transport through a credit-based link-level flow control mechanism. Commonly used in HPC and large AI clusters.
- **Topology (Network Topology):** The physical or logical arrangement and interconnection pattern of network devices (switches, nodes) within a data center or AI fabric. The choice of topology significantly impacts network properties like bandwidth, latency, path diversity, and cost. Examples relevant to AI fabrics include Fat-Tree (or Clos) networks, Dragonfly, and Torus.
- **Fat-Tree (Clos Network)**: A popular multi-stage switching topology used in data centers and AI fabrics designed to provide high bisection bandwidth (bandwidth between any two halves of the network) and multiple paths between endpoints, making it well-suited for the demanding, often all-to-all communication patterns found in AI workloads.
- **NVLink / NVSwitch:** NVIDIA's proprietary high-speed, direct GPU-to-GPU interconnect technology (NVLink) and associated switching system (NVSwitch). Provides significantly higher bandwidth and lower latency than standard PCIe or Ethernet for communication between GPUs within a node or within a tightly coupled multi-GPU complex, commonly used for Tensor Parallelism.
- **DPU (Data Processing Unit) / SmartNIC:** Advanced Network Interface Cards equipped with their own programmable processor cores (often ARM-based) and acceleration engines. They can offload tasks from the host CPU, such as network protocol processing (TCP/IP, RoCE), storage protocols (NVMe-oF), security functions (encryption, firewalling), virtualization networking (OVS), or even elements of AI data services or communication libraries.
- **GPUDirect Storage (GDS):** A technology enabling GPUs to initiate direct data transfers (reads and writes) to and from NVMe storage devices (local or over the network via NVMe-oF) using RDMA, bypassing the host CPU and system memory bounce buffers. This reduces I/O latency and frees up CPU resources, accelerating data loading and checkpointing.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Appendix B

Network Basics

B.1 Introduction to Computer Networks

At its core, a computer network connects two or more devices (computers, servers, printers, etc.) so they can share resources and communicate. Networks are fundamental to modern computing, enabling everything from simple file sharing to complex, distributed applications like large-scale Artificial Intelligence (AI).

- **Basic Components:**
  - **End Devices:** Computers, servers, GPUs, smartphones.
  - **Intermediary Devices:** Switches (connect devices within a local network), Routers (connect different networks, direct traffic), Firewalls (provide security).
  - **Network Interface Cards (NICs):** Hardware allowing devices to connect to the network.
  - **Network Media:** Cables (e.g., Ethernet copper, fiber optic) or wireless signals (Wi-Fi) that carry data.
- **Network Models:** Frameworks like the OSI model or the more practical TCP/IP model describe how different networking tasks are layered, from the physical connection (Layer 1) to application protocols (e.g., HTTP, Layer 7). Key layers include the Network Layer (IP addressing, routing) and Transport Layer (TCP/UDP, ensuring data delivery).
- **Network Types:** Common types include Local Area Networks (LANs) connecting devices in a limited area (like an office) and Wide Area Networks (WANs) connecting networks over larger geographical distances (like the Internet).
- **Network segmentation**: virtual networks (**VLANs**) may be used to isolate particular workloads and hardware. In a virtual DC environment a workload may be launched as a group of networked containers or VMs in its own VLAN such that it can be migrated about depending on current needs. Particular VLANs may be used for XPU discovery.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

B.2 Network Traffic Management

As networks grow and carry more data, managing the flow of traffic becomes critical. Network Traffic Management encompasses techniques to control and optimize data movement, ensuring network efficiency, performance, reliability, and security.

Goals:

- Maximize throughput (data transfer rate).
- Minimize latency (delay) and jitter (variation in delay).
- Ensure fairness among different users or applications.
- Prevent network congestion (overload).
- Provide reliable data delivery (minimize packet loss).
- Enhance security by identifying anomalous traffic.

Key Techniques:

- **Load Balancing:** Distributes incoming network traffic across multiple paths or servers to prevent any single resource from becoming a bottleneck. This enhances performance, scalability, and availability.
    - **Equal-Cost Multi-Path (ECMP):** ECMP is a routing strategy allowing traffic to be distributed over multiple paths that have the same routing cost (e.g., same number of hops or bandwidth).
        - *How it works:* Switches maintain routing tables with multiple next-hops for a destination. Incoming packets are typically assigned to a path using a *hash function* based on packet headers (Source/Destination IP, Source/Destination Port, Protocol).
        - *Challenge:* Standard hashing can lead to uneven load distribution, especially if many flows share similar header information (low entropy), which is common in specific workloads like AI.
    - **Other Algorithms:** Round Robin, Least Connections, weighted variations, etc.
    - **Global Server Load Balancing (GSLB):** Distributes traffic across geographically dispersed data centers.
    - **Process migration**: If network links become saturated or underperforming it may be necessary to move processing (when the workload is movable containers and/or VMs). Some workloads will expand over time.
- **Quality of Service (QoS):** A set of technologies used to manage network resources and prioritize traffic based on application requirements. This ensures critical or latency-sensitive applications (like VoIP or video conferencing) receive adequate performance even when the network is busy.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

    - **Mechanisms:**
        - *Classification & Marking:* Identifying traffic types and marking packets (e.g., using DSCP values) for differentiated treatment.
        - *Queuing:* Managing how packets are held and forwarded when congestion occurs (e.g., Priority Queuing gives preference to high-priority traffic; Weighted Fair Queuing (WFQ) allocates bandwidth proportionally).
        - *Traffic Shaping:* Delaying excess traffic (buffering) to smooth out bursts and conform to predefined rates.
        - *Policing:* Dropping or re-marking traffic that exceeds predefined rates.
- **Congestion Management & Avoidance:** Deals with situations where network links or nodes are overloaded.
    - **Flow Control:** Mechanisms preventing a sender from overwhelming a receiver.
        - **Priority Flow Control (PFC) (IEEE 802.1Qbb):** PFC is a Layer 2 mechanism operating on Ethernet networks.
            - *Overview:* Prevents packet loss due to congestion on a *per-priority* basis. Essential for protocols like RoCEv2 that are sensitive to loss.
            - *How it works:* When a switch buffer for a specific priority class nears capacity, it sends a PAUSE frame to the upstream sender, specifying which priority level(s) should be paused. The sender halts transmission for that priority until congestion eases.
            - *Key Features:* Supports multiple priority levels; per-priority control.
            - *Advantages:* Enables lossless Ethernet for specific traffic classes; improves reliability for sensitive applications.
            - *Disadvantages:* Requires PFC-capable switches and NICs; can introduce Head-of-Line (HOL) blocking where pausing one priority can inadvertently block lower priorities on the same physical link; requires careful configuration.
    - **Congestion Notification/Control:** Mechanisms signaling congestion back to the sending endpoints, allowing them to adjust their sending rate.
        - **Explicit Congestion Notification (ECN):** An extension to IP and TCP. Instead of dropping packets to signal congestion, routers can *mark* packets (in the IP header) when queues start building. The receiver echoes this mark back to the sender (in the TCP header), prompting the sender to reduce its rate proactively, thus avoiding packet loss.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

- **Data Center Quantized Congestion Notification (DCQCN):** A widely used algorithm, especially with RoCEv2. It combines ECN marking by the switch with rate adjustments at the sender and PFC as a safety net. Switches mark packets using ECN when congestion is detected. The receiver sends Congestion Notification Packets (CNPs) back to the sender, which then reduces its sending rate. PFC is used as a last resort to prevent buffer overflows if rate reduction isn't fast enough. Requires careful tuning of parameters.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

License

Creative Commons

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at http://www.opencompute.org/participate/legal-documents/ . If you have any questions, please contact OCP.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

About Open Compute Project

The Open Compute Project (OCP) brings at-scale innovations and hyperscaler best practices to all, spanning technology domains from the data center to the edge, and the technology stack from silicon, to systems, to site facilities and services. The international OCP Community is made up of organizations and people from hyperscale and tier-2 cloud data center operators, communications providers, colocation providers, diverse enterprises, and technology vendors. The OCP Foundation fosters collaboration within the Community to tackle market challenges in an array of OCP Projects that create open contributions such as specifications, designs, and more. The OCP Solution Provider Program then recognizes the application of those contributions in compliant products, solutions, and data center facilities and services with designations like OCP Inspired, OCP Accepted and OCP Ready. With the tenets of openness, impact, efficiency, scale and sustainability, the OCP engages and educates thousands of engineers every year through many events and webinars. Across many initiatives the OCP Foundation and Community are meeting the market today and shaping the future.

Learn more at: www.opencompute.org.

Date: July 2026

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.